49th International Symposium on Mathematical Foundations of Computer Science

MFCS 2024, August 26–30, 2024, Bratislava, Slovakia

Edited by

Rastislav Královič
Antonín Kučera

LIPICS

LIPIcs – Vol. 306 – MFCS 2024 www.dagstuhl.de/lipics

Editors

Rastislav Královič
Comenius University, Faculty of Mathematics, Physics, and Informatics, Bratislava, Slovakia
kralovic@dcs.fmph.uniba.sk

Antonín Kučera
Masaryk University, Faculty of Informatics, Brno, Czechia
tony@fi.muni.cz

ACM Classification 2012
Theory of computation; Mathematics of computing

ISBN 978-3-95977-335-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-335-5.

Publication date
August, 2024

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

Digital Object Identifier: 10.4230/LIPIcs.MFCS.2024.0

ISBN 978-3-95977-335-5 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The series of International Symposia on Mathematical Foundations of Computer Science (MFCS), organized since 1972, invites high-quality contributions in all branches of theoretical computer science. The broad scope of MFCS provides an opportunity to bring together researchers who do not usually meet at specialized conferences.

In the year 2024, the 49th edition of MFCS was held in Bratislava, on August 26–30. This volume contains 80 contributions selected out of 239 full submissions. The programme was further enriched by five invited talks given by Wojciech Czerwiński (University of Warsaw), Jarkko Kari (University of Turku), Kasper Green Larsen (Aarhus University), Rupak Majumdar (Max Planck Institute for Software Systems), and David Peleg (Weizmann Institude of Science).

We express our deep gratitude to all the members of the program committee and the reviewers. Due the the high number of submissions, the load of work was higher than anticipated. Still, the merits of all papers were considered very carefully not only in the reviews but also in subsequent discussions. We also warmly thank the invited speakers, as well as the authors of the submitted papers. We also thank the local organizing team chaired by Dana Pardubská.

MFCS proceedings have been published in the Dagstuhl/LIPIcs series since 2016. We thank Michael Wagner and the LIPIcs team for their kind help and support.

Rastislav Královič
Antonín Kučera

Organization

Program Committee

- Christel Baier (TU Dresden)
- Petra Berenbrink (Universität Hamburg)
- Christoph Berkholz (TU Ilmenau)
- Michael Blondin (Université de Sherbrooke)
- Mikołaj Bojańczyk (University of Warsaw)
- Joan Boyar (University of Southern Denmark)
- Broňa Brejová (Comenius University in Bratislava)
- Jean Cardinal (Université libre de Bruxelles)
- Pavol Cerny (TU Wien)
- Krishnendu Chatterjee (IST Austria)
- Ugo Dal Lago (University of Bologna)
- Stefan Dobrev (Slovak Academy of Sciences)
- Robert Elsässer (University of Salzburg)
- Leah Epstein (University of Haifa)
- Henning Fernau (Trier University)
- Fedor Fomin (University of Bergen)
- Pierre Fraigniaud (IRIF Université Paris Cité)
- Jean Goubault-Larrecq (CNRS and ENS Paris-Saclay)
- Kristoffer Arnsfelt Hansen (Aarhus University)
- Lane Hemaspaandra (University of Rochester)
- Petr Jančar (Palacký University Olomouc)
- Christos Kapoutsis (Carnegie Mellon University in Qatar)
- Stefan Kiefer (University of Oxford)
- Ralf Klasing (CNRS, LaBRI, University of Bordeaux)
- Naoki Kobayashi (University of Tokyo)
- Barbara König (University of Duisburg-Essen)
- Martin Koutecký (Charles University, Prague)
- Rastislav Královič (Comenius University in Bratislava, **co-Chair**)
- Antonín Kučera (Masaryk University, **co-Chair**)
- Tobias Momke (University of Augsburg)
- Madhavan Mukund (Chennai Mathematical Institute)
- Daniel Paulusma (Durham University)
- Giovanni Pighizzini (University of Milan)
- Alexander Rabinovich (Tel Aviv University)
- Peter Rossmanith (RWTH Aachen)
- Christian Scheideler (Paderborn University)
- Sebastian Siebertz (University of Bremen)
- Martin Škoviera (Comenius University in Bratislava)
- Bettina Speckmann (TU Eindhoven)
- Paul Spirakis (University of Liverpool)
- Daniel Stefankovic (University of Rochester)
- Till Tantau (University of Lübeck)
- Takeshi Tsukada (Chiba University)
- Ugo Vaccaro (University of Salerno)
- Igor Walukiewicz (LaBRI, Université de Bordeaux)

External Reviewers

- Samy Abbesc
- Duncan Adamson
- Jungho Ahn
- C Aiswarya
- Spyros Angelopoulos
- Atul Singh Arora
- Danilo Artigas
- Ali Asadi
- Tian Bai
- Paolo Baldan
- Sayan Bandyapadhyay
- Joergen Bang-Jensen
- Gregor Bankhamer
- Max Bannach
- Davide Barbarossa
- Benoit Barbot
- Mohammed Barhoush
- Georgios Barmpalias
- Alexey Barsukov
- Eduard Bartl
- Paul Bell
- Matthias Bentert
- Ioana Bercea
- Benjamin Bergougnoux
- Anna Bernasconi
- Antonio Bernini
- Nathalie Bertrand
- Marc Bezem
- Koustav Bhanja
- Mats Bierwirth
- Ivan Bliznets
- Karl Boddy
- Tobias Boege
- Narek Bojikian
- Jan Bok
- Édouard Bonnet
- Olivier Bournez
- Nicolas Bousquet
- Vladimir Boza
- Laura Bozzelli
- Dennis Breutigam
- Gerth Stølting Brodal
- Roberto Bruno
- Damien Busatto-Gaston
- Michaël Cadilhac
- Cezar Campeanu
- Olivier Carton
- Antonio Casares
- Arnaud Casteigts
- Arcangelo Castiglione
- Dibyayan Chakraborty
- Calvin Chau
- Michael C. Chavrimootoo
- James Cheney
- Janka Chlebikova
- Keerti Choudhary
- Florian Chudigiewitsch
- Ferdinando Cicalese
- Amanda Clare
- Ilan Cohen
- Vincent Cohen-Addad
- Gennaro Cordasco
- Arthur Correnson
- Emilio Cruciani
- Gianluca Curzi
- Manuel Cáceres
- Francesco Dagnino
- Antoine Dailly
- Avinandan Das
- Peter Davies
- Anuj Dawar
- Abhishek De
- Mark de Berg
- Argyrios Deligkas
- Vincent Despré
- Sanjana Dey
- Thomas C. Van Dijk
- Ruiwen Dong
- Vinicius F. dos Santos
- Pål Grønås Drange
- Benoît Dubus
- Maël Dumas
- Jérôme Durand-Lose
- Pavol Duris
- Alon Efrat
- Kord Eickmeyer
- Michael Elberfeld
- David Eppstein
- Funda Ergun
- Thomas Erlebach
- Solène Esnay
- Lene M. Favrholdt
- Stefan Felsner
- Zhidan Feng
- Diodato Ferraioli
- Nathan Flaherty
- Pamela Fleischmann
- Florent Foucaud
- Fabian Frei
- Dominik D. Freydenberger
- Akio Fujiyoshi
- Eric Fusy
- Askar Gafurov
- Travis Gagie
- Luisa Gargano
- Matthias Gehnen
- Abheek Ghosh
- Rohan Ghuge
- Tatsuya Gima
- Hugo Gimbert
- Lior Gishboliner
- Adrián Goga
- Ehsan Goharshady
- Petr Golovach
- Dmitriy Gorovoy
- Luciano Grippo
- Jan Friso Groote
- Christoph Grüne
- Pierre Guillon
- Manoj Gupta
- Venkatesan Guruswami
- Jens Oliver Gutsfeld
- Waldo Gálvez

- Thorsten Götte
- Anselm Haak
- Christoph Haase
- Christopher Hahn
- Niklas Hahn
- Masahiro Hamano
- Yassine Hamoudi
- Yo-Sub Han
- Zohair Raza Hassan
- Meng He
- Yumeng He
- Hsi-Ming Ho
- Hung Hoang
- Martin Hoefer
- Lukáš Holík
- Rostislav Horcik
- Kaave Hosseini
- Mathieu Hoyrup
- Petra Hozzová
- Annika Huch
- Ling-Ju Hung
- Peter Høyer
- Neil Immerman
- Radu Iosif
- Takehiro Ito
- Tomáš Jakl
- Tom Janßen
- Eva Jelinkova
- Gejza Jenca
- Emil Jeřábek
- Wanchote Po Jiamjitrak
- Dominik Kaaser
- Lawqueen Kanesh
- Frantisek Kardos
- Mehrdad Karrabi
- Leon Kellerhals
- Sandra Kiefer
- Rafael Kiesel
- Namjoong Kim
- Peter Kiss
- Nina Klobas
- Fabian Klute
- Dušan Knop
- Balagopal Komarath
- Christian Komusiewicz
- Athanasios Konstantinidis
- Juha Kontinen
- Peter Kostolányi
- Matan Kotler-Berkowitz
- Michal Koucky
- Dexter Kozen
- Laszlo Kozma
- Robert Krauthgamer
- Mario Krenn
- Kirsti Kuenzel
- Pascal Kunz
- Ondřej Kuželka
- Abhiruk Lahiri
- Tero Laihonen
- Jourdain Lamperski
- Michael Lampis
- Martin Lange
- John Lapinskas
- Laura Larios-Jones
- Olivier Laurent
- Brian Lavallee
- Tran Duy Anh Le
- Chia-Wei Lee
- Russell Lee
- Tuomo Lehtilä
- Louis Lemonnier
- Jérôme Leroux
- Asaf Levin
- Paul Blain Levy
- Jason Li
- Zsuzsanna Liptak
- Mingmou Liu
- Harry Liuson
- Aliaume Lopez
- Felicia Lucke
- Kelin Luo
- Carsten Lutz
- Neil Lutz
- Andreas Lööw
- Ramanujan M. Sridharan
- Marten Maack
- Martin Macaj
- Atrayee Majumder
- Andreas Malcher
- Kevin Mann
- Bubai Manna
- Alessio Mansutti
- Amaldev Manuel
- Giovanni Manzini
- Yaping Mao
- Andrea Marino
- Florian Andreas
- Tomas Masopust
- Marios Mavronicolas
- Richard Mayr
- Pierre McKenzie
- Alexander Melnikov
- Hammurabi Mendes
- Stefan Mengel
- Wolfgang Merkle
- Maximilian Merz
- Mehdi Mhalla
- Matthäus Micun
- Martin Milanič
- Kevin Milans
- Anders Miltner
- Pranabendu Misra
- Kenshi Miyabe
- Daniel Mock
- Hendrik Molter
- Benjamin Monmege
- Pedro Montealegre
- Nils Morawietz
- Nelma Moreira
- Laure Morelle
- Ryuhei Mori
- Amer Mouawad
- Loay Mualem
- Partha Mukhopadhyay
- Wolfgang Mulzer
- Andrea Munaro
- Takao Murakami

- Stefnaie Muroya Lei
- Judit Nagy-György
- Koji Nakazawa
- Timothy Ng
- Lê Thành Dũng Nguyêñ
- Zeyu Nie
- Nicolas Nisse
- Petr Novotný
- Pascal Ochem
- Jakub Opršal
- Sebastian Ordyniak
- Petr Osička
- Yota Otachi
- Youssouf Oualhadj
- Giacomo Paesani
- Sukanya Pandey
- Subhasree Patro
- Erik Paul
- Ami Paz
- David Peleg
- Ray Perlner
- Alessia Petescia
- Sebastian Pfau
- Michael Pinsker
- Jakob Piribauer
- Paolo Pistone
- Igor Potapov
- M. Praveen
- Nicola Prezza
- Wojciech Przybyszewski
- David Purser
- Stanislaw Radziszowski
- Md. Saidur Rahman
- Rajmohan Rajaraman
- Jozef Rajník
- Vijayaragunathan Ramamoorthi
- R. Ramanujam
- Narad Rampersad
- Malin Rau
- Michael Reidy
- Rogério Reis
- Adele Rescigno
- Emanuele Rodaro
- Benjamin Rossman
- Wojciech Rozowski
- Rubén Rubio
- Claudio Sacerdoti Coen
- Silas Sacher
- Toshiki Saitoh
- Prakash Saivasan
- Ken Sakayori
- Abhisekh Sankaran
- Raimundo Saona
- Ege Sarac
- Saket Saurabh
- Zdeněk Sawa
- Benjamin Scheidt
- Philipp Schepper
- Sven Schewe
- Patrick Schnider
- Jason Schoeters
- Christian Schwarz
- Jonas Schweichhart
- Ayumi Shinohara
- Anil Shukla
- Florian Sikora
- Pedro V. Silva
- Meera Sitharam
- George Skretas
- Michał Skrzypczak
- Daniel Slamanig
- Friedrich Slivovsky
- Siani Smith
- Frank Sommer
- Willem Sonke
- Petr Sosik
- Uéverton Souza
- A V Sreejith
- Srikanth Srinivasan
- B Srivathsan
- Frank Staals
- Rafał Stefański
- Tomasz Steifer
- Lara Stoltenow
- Jan Strejček
- Georg Struth
- Blair D. Sullivan
- S P Suresh
- Jakub Svoboda
- Yuta Takahashi
- Toru Takisaka
- Prafullkumar Tale
- Sébastien Tavenas
- Jan Arne Telle
- Soeren Terziadis
- Dimitrios Thilikos
- Sam Thompson
- Josef Tkadlec
- Konstantinos Tsakalidis
- Yiannis Tselekounis
- Andrea Turrini
- Benito van der Zander
- Marc van Kreveld
- Joseph Vandehey
- Manolis Vasilakis
- Alexandre Vigny
- Renaud Vilmart
- Harry Vinall-Smeeth
- Tomas Vinar
- N. V. Vinodchandran
- Jonni Virtema
- Tung Anh Vu
- Kazuki Watanabe
- Yohei Watanabe
- Simon Weber
- Dominik Wehr
- Pascal Weil
- Maximilian Weininger
- Philip Wellnitz
- Marcus Wilhelm
- Petra Wolf
- Prudence Wong
- Jules Wulms
- Tomoyuki Yamakami
- Takashi Yamakawa
- Geva Yashfe
- Ryo Yoshinaka
- Francesca Zaffora Blando
- Mohammad Zakzok
- Viktor Zamaraev
- Noam Zilberstein
- Marius Zimand

Invited Speakers

- Wojciech Czerwiński (University of Warsaw)
- Jarkko Kari (University of Turku)
- Kasper Green Larsen (Aarhus University)
- Rupak Majumdar (Max Planck Institute for Software Systems)
- David Peleg (Weizmann Institute of Science)

Organizing Committee

- Jana Kostičová, Comenius University in Bratislava
- Peter Kostolányi, Comenius University in Bratislava
- Dana Pardubská, Comenius University in Bratislava

On Key Parameters Affecting the Realizability of Degree Sequences

Amotz Bar-Noy
City University of New York (CUNY), NY, USA

Toni Böhnlein
Huawei, Zurich, Switzerland

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Yingli Ran
Zhejiang Normal University, Jinhua, Zhejiang, China

Dror Rawitz
Bar Ilan University, Ramat-Gan, Israel

Abstract

Call a sequence $d = (d_1, d_2, \ldots, d_n)$ of positive integers *graphic*, *planaric*, *outer-planaric*, or *forestic* if it is the degree sequence of some arbitrary, planar, outer-planar, or cycle-free graph G, respectively. The two extreme classes of graphic and forestic sequences were given full characterizations. (The latter has a particularly simple criterion: d is forestic if and only if its *volume*, $\sum d \equiv \sum_i d_i$, satisfies $\sum d \leq 2n - 2$.) In contrast, the problems of fully characterizing planaric and outer-planaric degree sequences are still open.

In this paper, we discuss the parameters affecting the realizability of degree sequences by restricted classes of sparse graph, including planar graphs, outerplanar graphs, and some of their subclasses (e.g., 2-trees and cactus graphs). A key parameter is the *volume* of the sequence d, namely, $\sum d$ which is twice the number of edges in the realizing graph. For planar graphs, for example, an obvious consequence of Euler's theorem is that an n-element sequence d satisfying $\sum d > 4n - 6$ cannot be planaric. Hence, $\sum d \leq 4n - 6$ is a necessary condition for d to be planaric. What about the opposite direction? Is there an *upper* bound on $\sum d$ that guarantees that if d is graphic then it is also planaric. Does the answer depend on additional parameters? The same questions apply also to sub-classes of the planar graphs.

A concrete example that is illustrated in the technical part of the paper is the class of outer-planaric degree sequences. Denoting the number of 1's in d by ω_1, we show that for a graphic sequence d, if $\omega_1 = 0$ then d is outer-planaric when $\sum d \leq 3n - 3$, and if $\omega_1 > 0$ then d is outer-planaric when $\sum d \leq 3n - \omega_1 - 2$. Conversely, we show that there are graphic sequences that are not outer-planaric with $\omega_1 = 0$ and $\sum d = 3n - 2$, as well as ones with $\omega_1 > 0$ and $\sum d = 3n - \omega_1 - 1$.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Degree Sequences, Graph Algorithms, Graph Realization, Outer-planar Graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.1

Category Invited Paper

Funding Supported in part by a US-Israel BSF grant (2022205).

1 Introduction

Background. Throughout, we consider a a nonincreasing sequence $d = (d_1, \ldots, d_n)$ of n nonnegative integers. The sequence d is *graphic* if it is the degree sequence of some graph G. The graph realization problem concerns deciding, for a given d, if it is graphic, and if so -

constructing a realizing graph for it. The problem has been studied extensively over the past 60 years, and was given both combinatorial characterizations and construction algorithms, cf. [9, 13, 14]. In particular, by [9], a nonincreasing sequence d is graphic if and only if

$$\sum_{i=1}^{\ell} d_i \le \ell(\ell-1) + \sum_{i=\ell+1}^{n} \min\{\ell, d_i\}, \qquad \text{for } \ell = 1, \ldots, n. \tag{1}$$

Beyond its theoretical interest, realization questions are also relevant in the context of specification-based network design, where the users specify the desired network properties, as well as in some scientific contexts where it is required to discover the unknown structure of a network based on measurements on its various parameters.

The question can be asked also in the context of special graph families. In particular, the realizability of a given sequence d by a *planar* graph was studied in [23] and some of the sequences whose status was left undetermined in [23] were later resolved in [10, 11]. In addition, regular planar graphic sequences were classified in [15], and planar bipartite biregular degree sequences were studied in [1]. Still, the realizability of a given sequence d by a planar graph was not given a complete solution so far.

The same question can be asked with respect to other restricted classes of graphs. One extreme example is that of *trees* and more generally *forests*. Whereas the characterization (1) for graphic degree sequences (realizable by general graphs) is somewhat involved, composed of n different conditions and affected individually by each of the degrees in the sequence, the characterization for *forestic* sequences (namely, ones realizable by a forest) is very simple: an n-element sequence d is forestic if and only if its *volume*, defined as $\sum d \equiv \sum_{i=1}^{n} d_i$, satisfies $\sum d \le 2n-2$ (with equality if and only if the sequence is realizable by a tree). One might conjecture that restricted graph classes should tend to have simpler characterizations, or ones that depend on fewer parameters.

The current paper is motivated by the (intuitively clear) fact that the volume parameter of the sequence d plays a significant role in its realizability by classes of *sparse* graphs. One direction is easy: if the class $\mathcal{G}$ contains only graphs of M or fewer edges, then clearly, sequences of volume larger than $2M$ cannot have a realization from $\mathcal{G}$. Hence $\sum d \le 2M$ is a *necessary* condition for realizability by a graph from $\mathcal{G}$. Here, we discuss the converse question: Can one derive *sufficient* conditions for realizability by $\mathcal{G}$ based on the volume parameter?

Indeed, for various low-volume graph classes $\mathcal{G}$, there are known results, guaranteeing that if d is graphic and $\sum d$ is sufficiently small, then d has a realizing graph in $\mathcal{G}$. In particular, consider the following hierarchy of graph classes and corresponding hierarchy of volume bounds.

- Forests [12]: A sequence d is forestic if and only if $\sum d \le 2n-2$ and $\sum d$ is even.
- Uni-cyclic graphs [6]: These are connected graphs with precisely one cycle. A sequence d is uni-cyclic if and only if $\sum d = 2n$ and $d_1 \le n-1$.
- Bi-cactus graphs [2]: Here, there is a full characterization that involves two additional parameters, ω_1 and ω_{odd}, where ω_i is the number of i's in d and ω_{odd} is the number of odd degrees in d. In particular, a necessary condition for a sequence d to be realizable by a bi-cactus is that $\sum d < 8n/3$. The sufficient condition is more complex, utilizing also ω_1 and ω_{odd}.
- Cactus graphs [20]: Here, again, there is a full characterization that involves ω_1 and ω_{odd}. In particular, a necessary condition for a sequence d to be realizable by a cactus graph is that $\sum d \le 3(n-1)$, and the sufficient condition utilizes also ω_1 and ω_{odd}.
- Outer-planar graphs: This is the main technical contribution of the current paper, to be described shortly.

- 2-trees [7]: This class was given a full characterization, where the volume requirement is $\sum d = 4n - 6$ but the precise conditions depend also on ω_1 and ω_2, and involve also some exceptions.
- Planar graphs [4]: Full characterization is still out of reach. The natural necessary condition based on volume is $\sum d \leq 6n - 12$. The sufficient condition given in [4] depends also on ω_1.

Our main technical contribution concerns sequences that can be realized by outer-planar graphs. A graph is *outer-planar* if it can be embedded in the plane such that edges do not intersect each other and additionally, each vertex lies on the outer face of the embedding, i.e., no vertex is fully surrounded inside an internal face (cf. [24]). Call a sequence d *planaric* (respectively, *outer-planaric*) if it is the degree sequence of some planar (resp., outer-planar) graph G.

Since it is known that every outer-planar graph has at most $2n - 3$ edges, it follows that every sequence d with $\sum d > 4n - 6$ cannot be outer-planaric. In fact, as claimed in [3], this bound can be improved if ω_1, the multiplicity of degree 1, is taken into consideration. Specifically, if the sequence d satisfies $\sum d > 4n - 6 - 2\omega_1$ and $d \neq (n - 1, 1, \ldots, 1)$, then it is not outer-planaric (the exceptional sequence has $n - 1$ leaves and volume $2n - 2$, and is realizable by a star graph). It is also easy to show that this bound is tight, in the sense that there are outerplanar sequences d with $\sum d = 4n - 6 - 2\omega_1$, for ω_1 values in the range $[1, n - 2]$.

Focusing on the converse question, we look for a function $f(n, \omega_1)$ guaranteeing that if d is graphic and $\sum d \leq f(n, \omega_1)$, then d is *always* outer-planaric. A straightforward such bound is $f(n, \omega_1) = 2n - 2$, since as mentioned above, if $\sum d \leq 2n - 2$ then d has a realization by a forest, hence it is trivially outer-planaric. Here, we present a tight answer to the question, separated into two cases, depending on whether or not the sequence d contains 1's. Specifically, we show that if $\omega_1 = 0$ then the desired property holds when $\sum d \leq 3n - 3$, and if $\omega_1 > 0$ then the desired property holds when $\sum d \leq 3n - \omega_1 - 2$. Conversely, we show that the set of graphic sequences that are not outer-planaric includes:

(1) sequences with $\omega_1 = 0$ and $\sum d = 3n - 2$;

(2) sequences with $\omega_1 > 0$ and $\sum d = 3n - \omega_1 - 1$.

Related Work. A number of papers studied the outerplanar degree realization problem in the past. *Forcibly* outerplanar graphic sequences, i.e., sequences *all* of whose realizations are outerplanar, were given a full characterization in [8]. The degree sequences of maximal outerplanar graphs with exactly two 2-degree nodes were characterized in [19]. This was also mentioned in [7] independently. The degree sequences of maximal outerplanar graphs with at most four vertices of degree 2 were characterized in [17].

The special class *2-trees* has also received some attention. A graph G is a 2-tree if $G = K_3$ or G has a vertex v with degree 2, whose neighbors are adjacent and $G[V \setminus \{v\}]$ is a 2-tree. Sufficient conditions for a sequence d to have a realization by a 2-tree were given in [18]. The degree sequences of 2-trees were fully characterized in [7]. (The conditions are surprisingly rather involved, and include a number of specific exceptions.) Note that 2-trees have $\sum d = 4n - 6$. This implies, using Theorem 1 of [7], that if a sequence d satisfies some specific conditions, then d has a realizing 2-tree. Rengarajan and Veni Madhavan [22] have shown that every 2-tree has a 2-page book embedding. Unfortunately, the class of 2-trees is not hereditary (meaning that a subgraph of a 2-tree is not necessarily a 2-tree), so the result of [7] does not extend to non-maximal degree sequences.

The planar degree realization problem also has a long history. Regular planar graphic sequences and planar bipartite biregular degree sequences were given a classification in [15] and [1] respectively. Schmeichel and Hakimi [23] identified the graphic sequences with $d_1 - d_n = 1$ that are planaric, and did the same for $d_1 - d_n = 2$ with a small number of unresolved exceptions, some of which were later resolved in [10, 11]. Additional studies on special cases of the planaric degree realization problem are discussed in [21].

In [4] it is shown that for every sequence d with $\sum d \leq 4n - 4 - 2\omega_1$, if d is graphic then it is also planaric. Conversely, there are graphic sequences with $\sum d = 4n - 2\omega_1$ that are non-planaric. For the case $\omega_1 = 0$, it is shown therein that d is planaric when $\sum d \leq 4n - 4$, and conversely, there is a graphic sequence with $\sum d = 4n - 2$ that is non-planaric.

A cactus graph is a connected graph in which any edge may be a member of at most one cycle, which means that different cycles do not share edges, but may share one vertex. Rao [20] provided a full characterization for degree sequences realizable by cactus graphs, and also gave a characterization for degree sequences realizable by cactus graphs whose cycles are triangles and for degree sequences realizable by connected graphs whose blocks are cycles of k vertices. Beineke and Schmeichel [5] characterized cacti degree sequences with up to four cycles and also provided a sufficient condition for cactus realization. A characterization for degree sequences realizable by bipartite cactus graphs was shown in [2].

2 Preliminaries and Known Results

Some Terminology. Two necessary (but not sufficient) conditions for a non-increasing sequence $d = (d_1, \ldots, d_n)$ to be graphic are that $\sum d$ is even and $d_1 \leq n - 1$. We refer to sequences that satisfy these two conditions as *standard* sequences. Hereafter, we consider only standard sequences.

For a nonincreasing sequence d of n nonnegative integers, let $\mathsf{pos}(d)$ denote the prefix consisting of the positive integers of d. We use the shorthand a^k to denote a subsequence of k consecutive a's. Denote the *volume* of sequence $d = (d_1, \ldots, d_n)$ by $\sum d \equiv \sum_{i=1}^{n} d_i$.

Trees and Forests. Consider a sequence $d = (d_1, \ldots, d_n)$ of positive integers such that $\sum d$ is an even number. It is known that if $\sum d \leq 2n - 2$, then d is graphic and moreover, $\mathcal{G}(d)$ contains an acyclic graph (forest). In this case, we say that d is *forestic*. If $\sum d = 2n - 2$, then d can be realized by a tree and we say that d is *treeic*. We refer to a vertex of degree one as a *leaf*.

We make use of a special type of realizations for treeic and forestic sequences, known as *caterpillar graphs*. In a caterpillar graph $G = (V, E)$, all non-leaves are arranged on a path which we call the *spine* of G, i.e., the spine $S = (x_1, \ldots, x_s) \subset V$ is an ordered sequence where $(x_i, x_{i+1}) \in E$, for $i = 1, \ldots, s - 1$ (see Figure 1 for an example).

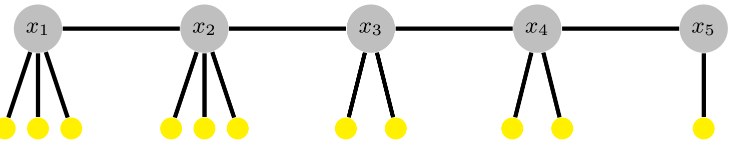


Figure 1 Caterpillar graph with degree sequence $(5, 4^3, 2, 1^{11})$. Leaves are depicted in yellow.

The following (possibly folklore) observation appears in [3]. We provide a proof, since its method will be instrumental in what follows.

▶ **Observation 1** ([3])**.** *Any forestic sequence d can be realized by a forest composed of the union of a caterpillar graph and a matching.*

Proof. First consider a treeic sequence d (such that $\sum d = 2n - 2$). Assume there are $n - s$ vertices of degree 1 in d. Denote by d^* the prefix of d that contains all the degrees $d_i > 1$. It follows that $\sum d^* + (n - s) = \sum d = 2n - 2$, or equivalently $\sum d^* - 2s + 2 = n - s$. To get the caterpillar realization of d, first arrange s vertices corresponding to the degrees of d^* in a path. The path edges contribute $2s - 2$ to the volume $\sum d^*$. The missing $\sum d^* - 2s + 2 = n - s$ degrees are satisfied by attaching the $n - s$ leaves to the path, which now forms the spine of the caterpillar realization. Note that the order of the vertices on the spine can be arbitrary.

Now assume that d is forestic but not treeic, i.e., $\sum d < 2n - 2$. In this case, remove pairs of 1 degrees from d until the volume of the reaming sequence d' with n' vertices is $2(n' - 2)$. This must happen because each pair removal reduces the volume by 2 while $2n - 2$ decreases by 4. Let G' be the caterpilar realization of d' as implied by the first part of the proof. To get the realization of d, add $(n - n')/2$ edges to G' to satisfy the $n - n'$ removed 1 degrees by a matching. ◀

Necessary Conditions for Outer-Planaric Sequences.

▶ **Lemma 2** ([25])**.** *If $d = (d_1, \ldots, d_n)$ is an outer-planaric degree sequence where $n \geq 2$, then $\sum d \leq 4n - 6$.*

It is known that every outer-planar graph has at least two vertices of degree two or less, and at least three vertices of degree three or less, see for example Sysło [24]. This implies the following necessary condition for outer-planaric sequences.

▶ **Lemma 3** ([24])**.** *If $d = (d_1, \ldots, d_n)$ is an outer-planaric degree sequence, then*
(i) $d_{n-1} \leq 2$, *and*
(ii) $d_{n-2} \leq 3$.

The Havel-Hakimi Algorithm and Outer-planaric Sequences. The lay-off techniques developed by Havel and Hakimi, and subsequently extended by Kleitmann and Wang, are among the fundamental tools for constructing realizations for degree sequences. One might hope to be able to use such techniques for finding (outer-)planar realizations as well. Unfortunately, it turns out that the lay-off operation does not, in general, preserve planarity or outer-planarity (only graphicity), hence it cannot be applied directly to generate a realizing outer-planar graph for a given outer-planaric sequence.

Nevertheless, we do make use of the lay-off technique, and specifically the *minimum pivot* version of the Havel-Hakimi algorithm [13, 14]. It is used for realizing a nonincreasing degree sequence $d = (d_1, d_2, \cdots, d_n)$ associated with the vertices $v_1, v_2, \ldots, v_n$, and is based on repeatedly performing the following operation, hereafter referred to as the *MP-step*, until all the vertices reach their required degree. Suppose the current sequence of residual nonzero degrees is $d' = (d'_1 \geq d'_2 \geq \cdots \geq d'_h)$ and the corresponding vertices are $v_{i_1}, v_{i_2}, \ldots, v_{i_h}$.

- Pick the vertex $v = v_{i_h}$ with degree d'_h as pivot.
- Set v's neighbors to be the d'_h vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_{d'_h}}$.
- Delete the pivot from d', reduce by 1 the residual degrees of its selected neighbors, and delete from d every vertex whose residual degree became zero.

The key observation is that, in case the MP-step transforms the residual degree sequence d into d', the following holds: d is graphic if and only if d' is graphic.

3 Outer-Planaric Degree Sequences

3.1 Necessary conditions

We first recall that Lemma 2 can be improved if ω_1, the multiplicity of degree 1, is taken into consideration. We add the proof for completeness.

▶ **Lemma 4** ([3]). *If d is an outer-planaric sequence such that $\sum d > 2n - 2$, then $\sum d \leq 4n - 6 - 2\omega_1$.*

Proof. Let d be as in the lemma and let $G = (V, E)$ be an outerplanar graph realizing d. Construct a graph G' by deleting from G all the ω_1 vertices of degree 1 together with their incident edges, and subsequently deleting the vertices whose degree was reduced to zero. Notice that the graph G' may contain vertices whose degree became 1. Observe that G' is outerplanar, and let $d' = \deg(G')$. Note that $n' \leq n - \omega_1$, where n' the number of vertices in G'. We have that $\sum d' \leq 4n' - 6$ due to Lemma 2. Hence,

$$\sum d = \sum d' + 2\omega_1 \leq 4(n - \omega_1) - 6 + 2\omega_1 = 4n - 6 - 2\omega_1 \ ,$$

where the first equality comes from the fact that ω_1 vertices of degree 1 together with their incident edges are deleted. ◀

We remark that Lemma 4 is false if we drop the assumption $2n - 2 < \sum d$. To see this, consider the sequence $d = (a, 1^a)$ which is (uniquely) realized by a star graph and therefore outer-planaric. The construction of G' in the proof above, applied to d, yields an isolated vertex. Note that Lemma 2 cannot be applied in this case. Indeed, we have that

$$\sum d = 2a > 2a - 2 = 4(a + 1) - 6 - 2a = 4n - 6 - 2\omega_1.$$

Observe that sequences of the form $d = (a, 1^{a+t})$ are the only sequences where G' has order one or less. If $t \geq 2$ (note that t must be even), the bound can be shown directly with the above calculation. We state the following corollary.

▶ **Corollary 5.** *If $d = (d_1, \ldots, d_n)$ is an outer-planaric sequence such that $d \neq (a, 1^a)$, then $\sum d \leq 4n - 6 - 2\omega_1$.*

The next corollary gives us a useful necessary condition for outer-planaric sequences.

▶ **Corollary 6** ([16]). *If $d = (d_1, \ldots, d_n)$ is an outer-planaric sequences, then $d_1 + d_2 \leq n + 2$.*

3.2 Sufficient Conditions

In this section, we show sufficient condition for outer-planaric sequences.

The collection of *low-volume* (standard) sequences is defined as

$$LV \ = \ LV_1 \cup LV_2 \ ,$$

where

$$LV_1 = \left\{ d \mid \sum d \leq 3n - \omega_1 - 2, \ \ d_n = 1, \ \ d \text{ is graphic} \right\}$$
$$LV_2 = \left\{ d \mid \sum d \leq 3n - 3, \ \ d_n = 2 \right\}$$

Notice that by assuming that the sequences are standard, we implicitly require that $\sum d$ is even and that $d_1 \leq n - 1$.

We now show that all sequences in LV are outer-planaric. For the proof we analyze a slightly large collection than LV_2, namely,

$$LV_2' = \left\{ d \mid \sum d \leq 3n-2, \quad d_n = 2 \right\},$$

Notice that $LV_2 \subset LV_2'$. We prove that all the sequences in LV_2' except for a small set EX are outer-planaric. As $LV_2 \subset LV_2' \setminus EX$, we get that all the sequences in LV_2 are outer-planaric. Moreover, we also use LV_2' to prove that all the sequences in LV_1 are outer-planaric.

In order to analyze the collection LV_2', we break it into four sub-collections A, B, C, D, and handle each of them separately. Specifically, we define the following sets.

$$\begin{aligned}
A &= \left\{ (3^s, 2^{n-s}) \mid n \geq 3, \quad 0 \leq s \leq n-2, \quad \text{and } s \text{ is even} \right\}, \\
B &= \left\{ (d_1, 2^{n-1}) \mid d_1 \geq 4 \quad \text{and} \quad d_1 \text{ is even} \right\}, \\
C^M_{even} &= \left\{ (d_1, d_2, 2^{n-2}) \mid d_1, d_2 \geq 4, \quad d_1 + d_2 = M \quad \text{and} \quad d_1, d_2 \text{ are even} \right\}, \\
C^M_{odd} &= \left\{ (d_1, d_2, 2^{n-2}) \mid d_1 \geq 5, \quad d_2 \geq 3, \quad d_1 + d_2 = M \quad \text{and} \quad d_1, d_2 \text{ are odd} \right\}, \\
C_{even} &= \bigcup_{M \leq n+2} C^M_{even}, \\
C_{odd} &= \bigcup_{M \leq n+2} C^M_{odd}, \\
C &= C_{even} \cup C_{odd}, \\
D &= \left\{ d \mid d_1 \geq 4, \quad d_3 \geq 3, \quad d_n = 2, \quad \sum d \leq 3n-2 \right\}.
\end{aligned}$$

Note that all sequences in A, B, C, D are standard.

▶ **Observation 7.** *$LV_2' = A \cup B \cup C \cup D$.*

Proof. One can check directly that $A, B, C, D \subseteq LV_2'$, and hence $A \cup B \cup C \cup D \subseteq LV_2'$. For the converse, consider some $d \in LV_2'$. If $d \in LV_2' \setminus D$, then $d_3 = 2$ or $d_1 \leq 3$, so d has the form $(d_1, d_2, 2^{n-2})$ or $(3^s, 2^{n-s})$. First assume that $d_3 = 2$ and consider d of the form $(d_1, d_2, 2^{n-2})$. If $d_2 = 2$, then $d \in B$. If $d_1 = d_2 = 3$, then $d \in A$. Otherwise $d_1 \geq 4$ and $d_2 \geq 3$, we have $d_1 + d_2 \leq n + 2$ since $\sum d \leq 3n-2$. In this case, $d \in C$. Next consider d of the form $(3^s, 2^{n-s})$. Since $\sum d \leq 3n-2$, $s \leq n-2$. In this case, $d \in A$. Combining these cases, we have $LV_2' \subseteq A \cup B \cup C \cup D$. The observation follows. ◀

▶ **Lemma 8.** *Every sequence $d \in A$ is outer-planaric.*

Proof. Let d be as in the lemma. By the definition of A, $d = (3^s, 2^{n-s})$ where s is even and $0 \leq s \leq n-2$. We construct an outer-planar realization G of d as follows. First, arrange n vertices in a cycle, i.e., let $G = C_n$. If $s = 0$, then G is a valid realization of d (noting that $n \geq 3$ so G is a simple graph). Now suppose $s > 0$. Select one vertex y on the cycle and denote its clockwise (respectively, counter-clockwise) neighbor of distance x by u_x (resp., v_x), for $x = 1, \ldots, s/2$. Observe that the vertices $u_{s/2}$ and $v_{s/2}$ cannot be connected by an edge in C_n, since $s \leq n-2$. To complete our construction, we add to $E(G)$ a matching consisting of the edges (v_x, u_x), for $x = 1, \ldots, s/2$ Since these new edges can be placed *inside* the cycle, G has an outer-planar embedding as shown in Figure 2. Verifying that $\deg(G) = d$, the claim follows. ◀

▶ **Lemma 9.** *Every sequence $d \in B$ is outer-planaric.*

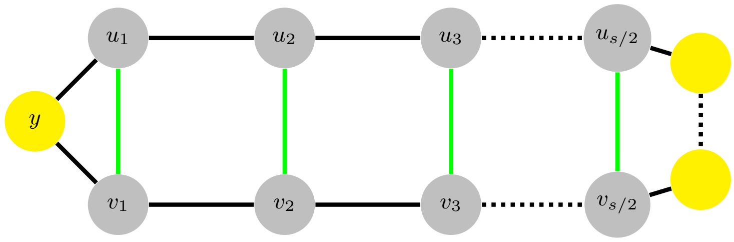


Figure 2 Schematic outer-planar realization of a sequence $d = (3^s, 2^{n-s})$ where s is even and $0 \le s \le n-2$. The yellow vertices have degree two and the gray vertices have degree three. Cycle edges are drawn in black and matching edges are drawn in green.

Proof. Since d_1 is even, d can be realized by the graph $G = (V, E)$ where $V = \{v_1, \ldots, v_n\}$ and the edge set E is constructed by the following steps:

(1) $E_1 \leftarrow \{(v_1, v_i) \mid i = 2, \ldots, d_1 + 1\}$ connects v_1 to d_1 other vertices.

(2) $E_2 \leftarrow \{(v_{2j}, v_{2j+1}) \mid j = 2, \ldots, d_1/2\}$ is a perfect matching among the neighbors of v_1.

(3) Let $W \leftarrow V \setminus \{v_1, \ldots, v_{d_1+1}\}$ be the set of remaining vertices. If $W = \emptyset$, then set $E_3 \leftarrow \{(v_2, v_3)\}$. Otherwise (i.e., if $W = \{v_j \mid j = d_1 + 2, \ldots, n\}$ where $d_1 + 2 \le n$), $E_3 \leftarrow \{(v_2, v_{d_1+2}), (v_{d_1+2}, v_{d_1+3}), \ldots, (v_n, v_3)\}$.

(4) Set $E \leftarrow E_1 \cup E_2 \cup E_3$.

Note that Steps (1) and (3) can be performed based on d since $d_1 \le n-1$. More specifically, Step (1) yields well-defined edges, i.e., $E_1 \subseteq V \times V$, and in Step (3), W is well-defined. Step (2) can be performed based on d i.e., $E_2 \subseteq V \times V$, since d_1 is even. Figure 3 illustrates these steps. Note that every vertex is located on the outer face, so G is outerplanar. As G realizes the degree sequence d, it follows that d is outer-planaric. ◀

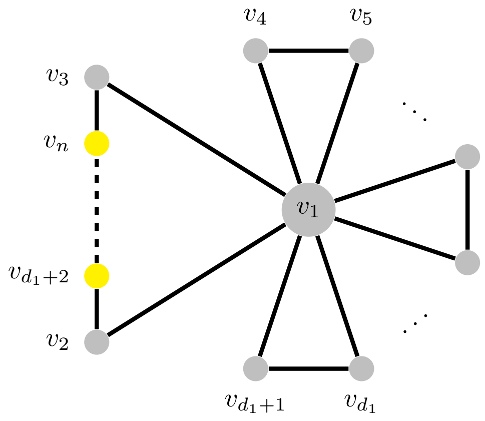


Figure 3 An illustration of the graph G constructed in the proof of Lemma 9 where the set W (vertices in yellow) is not empty.

Define $EX = C^{n+2}_{even} = \{(d_1, d_2, 2^{n-2}) \mid d_1, d_2 \ge 4, \;\; d_1 + d_2 = n+2 \;\text{ and }\; d_1, d_2 \text{ are even}\}$.

▶ **Lemma 10.** *Let $d \in C \setminus EX$. Then d is outer-planaric.*

Proof. Suppose $d \in C \setminus EX$. Since $\sum d \le 3n-2$, necessarily $d_1 + d_2 \le n+2$.

First suppose $d \in C_{odd}$. Construct a realizing graph G on the vertex set $V = \{v_1, \ldots, v_n\}$ by taking the following steps. Let v_1 and v_2 be the vertices with degree d_1 and d_2 respectively.

(1) $E_1 = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$. These edges connect v_1 to v_2 and also connect v_1 and v_2 to two common neighbors, v_3 and v_4.

(2) $E_2 \leftarrow \{(v_1, v_i) \mid i = 5, \ldots, d_1 + 1\}$ connects $d_1 - 3$ extra neighbors from the remaining vertices to v_1, and $E_3 \leftarrow \{(v_{5+2j}, v_{6+2j}) \mid j = 0, \ldots, \frac{d_1-5}{2}\}$ sets a perfect matching among the vertices $\{v_i \mid i = 5, \ldots, d_1 + 1\}$.

(3) If $d_2 = 3$, then $E_4, E_5 \leftarrow \emptyset$.
Otherwise, $E_4 \leftarrow \{(v_2, v_i) \mid i = d_1 + 2, \ldots, d_1 + d_2 - 2\}$ connects $d_2 - 3$ extra neighbors from the remaining vertices to v_2, and $E_5 \leftarrow \{(v_{d_1+2+2j}, v_{d_1+3+2j}) \mid j = 0, \ldots, \frac{d_2-5}{2}\}$ is a perfect matching among the vertices $\{v_i \mid i = d_1 + 2, \ldots, d_1 + d_2 - 2\}$.

(4) Let $W \leftarrow V \setminus \{v_1, \ldots, v_{d_1+d_2-2}\}$ be the set of remaining vertices. If $W = \emptyset$, then set $E_6 \leftarrow \{(v_1, v_3)\}$
Otherwise, plant the vertices of W on the edge (v_1, v_3), or more formally, replace (v_1, v_3) with the edges in the path $(v_1, v_{d_1+d_2-1}, \ldots, v_n, v_3)$, namely with the edge set $E_6 \leftarrow \{(v_1, v_{d_1+d_2-1}), (v_{d_1+d_2-1}, v_{d_1+d_2}), \ldots, (v_n, v_3)\}$

(5) Set $E \leftarrow (E_1 \setminus \{(v_1, v_3)\} \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6$.

Note that Step (1) can be performed based on d, in the sense that no vertex is assigned more edges than its degree, since $d_1, d_2 \geq 3$ and $d_3 = d_4 = 2$. Steps (2) and (3) use $d_1 + d_2 - 6$ distinct vertices (from v_5 to $v_{d_1+d_2-2}$) to serve as the remaining neighbors of v_1 and v_2. Since $d_1 + d_2 \leq n + 2$, Steps (2) and (3) can be performed based on d, i.e., $d_1 + d_2 - 2 \leq n$ so $E_2 \cup E_3 \subseteq V \times V$. Besides, the matchings in Steps (2) and (3) can be performed based on d, i.e., $E_4 \subseteq V \times V$, since d_1 and d_2 are both odd. Figure 4 illustrates the steps of the construction. One can check that constructed graph G realizes d and is outer-planaric, since all vertices are located on the outer face.

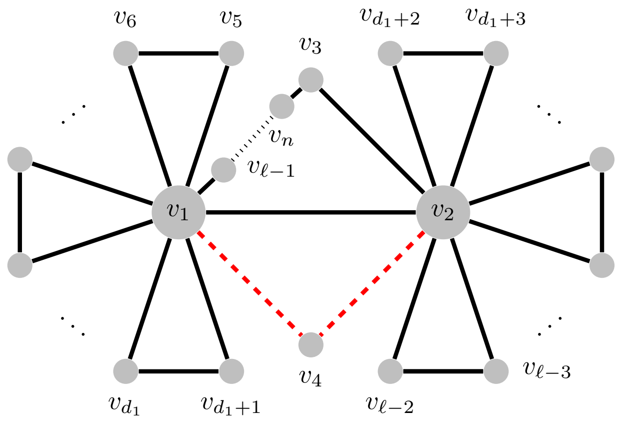


Figure 4 An illustration of the graph G constructed in the proof of Lemma 10 for the case where $d \in C_{odd}$. Let $\ell = d_1 + d_2$. The graph for the case where $d \in C_{even}$ is obtained by removing the red dashed edges.

Now suppose $d \in C_{even} \setminus EX$. Hence, $d_1 + d_2 \leq n + 1$. Let $d' = (d_1 + 1, d_2 + 1, 2^{n-1})$. Notice that $n' = n + 1$ and that $d'_1 + d'_2 = d_1 + d_2 + 2 \leq n + 1 + 2 = n' + 2$. Hence, $d' \in C_{odd}$. Construct a realizing outerplanar graph G' for the sequence d'. Let G be the graph we get by removing v_4 and its two edges the edges (v_1, v_4) and (v_2, v_4) from G'. (The dashed red edges in Figure 4.) Observe that G realizes d and it is outer-planar. ◀

▶ **Lemma 11.** *Let $d \in EX$. Then d is not outer-planaric.*

Proof. Recall that $d = (d_1, d_2, 2^{n-2})$, where $4 \le d_1 \le n-1$, $d_2 \ge 3$, $d_1 + d_2 = n+2$, and both d_1 and d_2 are even. Assume towards contradiction that d is outer-planaric, and let $G = (V, E)$ be a realizing outerplanar graph for it. Let v_1 and v_2 be the vertices of degree d_1 and d_2 in G, respectively.

Let $k \le d_2$ be the number of common neighbors of v_1 and v_2. If v_1 and v_2 are not adjacent, then

$$n - 2 = (d_1 - k) + (d_2 - k) + k = d_1 + d_2 - k = n + 2 - k \; .$$

Hence, $k = 4$. In this case, G contains a subgraph homeomorphic to $K_{2,3}$, and therefore it is not outer-planar [24], leading to a contradiction.

Otherwise, if v_1 and v_2 are adjacent, then

$$n - 2 = (d_1 - k - 1) + (d_2 - k - 1) + k = d_1 + d_2 - 2 - k = n - k \; .$$

Hence, $k = 2$. Consequently, denoting the two common neighbors of v_1 and v_2 in G by v_3 and v_4, let $N'[v_1]$ (respectively, $N'[v_2]$) be the set of neighbours of v_1 (resp., v_2) except for v_3, v_4 and v_2 (resp., v_1). See Figure 5. $|N'[v_1]|$ and $|N'[v_2]|$ are odd, since d_1, d_2 are even. Since the vertices in $N'[v_1] \cup N'[v_2]$ all have degree 2, there must exist (at least) one vertex in $N'[v_1]$ and one vertex in $N'[v_2]$ that are connected. (In Figure 5, these are marked as v_5 and v_6.) It follows that there are three paths from v_1 to v_2, the first contains v_3, the second contains v_4, and the third contains v_5 and v_6. In this situation, in any planar embedding of G, at least one of the common neighbors v_3 or v_4 or the pair v_5 and v_6 is not in the outer face. (In Figure 5, the vertex v_3 is blocked.) This leads to a contradiction. ◀

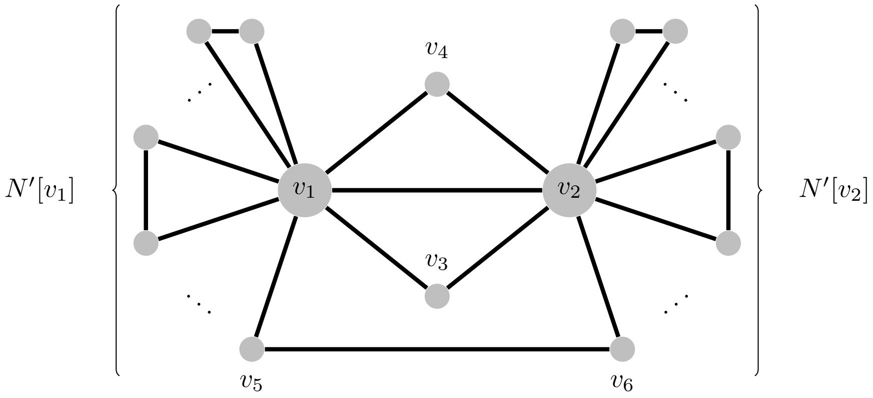


Figure 5 The non outer-planar graph G constructed in the proof of Lemma 11. $N'[v_1]$ and $N'[v_2]$ are, respectively, the vertex sets to the left of v_1 and to the right of v_2.

To prove the next lemma, we construct an outer-planar graph by starting from a caterpillar graph and adding a matching, increasing the degree of each vertex by one. The next observation describes a part of the construction used repetitively.

▶ **Observation 12.** *Let $G = (V, E)$ be a caterpillar graph with an outer-planar embedding as depicted in Figure 1. If $L \subseteq V$ is an even set of leaf vertices that appear consecutively in the embedding, then one can add a matching between the vertices of L such that the resulting graph has an outer-planar embedding.*

Proof. Let $L = \{\ell_1, \ldots, \ell_h\}$ be the leaf vertices in consecutive order, for even h. Note that vertices in L are pairwise non-adjacent. We add the matching edges (ℓ_{2i-1}, ℓ_{2i}), for $i = 1, \ldots, h/2$, i.e., we add every second edge of the path $(\ell_1, \ldots, \ell_h)$. ◀

The next lemma proves the most general case where $d_n \geq 2$.

▶ **Lemma 13.** *Every sequence $d \in D$ is outer-planaric.*

Proof. Consider $d \in D$. Denote $k = \sum d - (2n-2)$. Observe that k is even and that $k \geq 2$, since $\sum d \geq 2n$. Also, $k \leq 3n - 2 - (2n-2) = n$.

Our proof consists of two major steps:

(1) Create a treeic sequence d' from d and find an (outer-planar) caterpillar realization G of d' as described in Observation 1.

(2) Modify G by adding edges such that $\deg(G) = d$ and G remains outer-planar.

For step (1), construct the sequence $d' = (d'_1, d'_2, \ldots, d'_n)$ as follows:

$$d'_i = \begin{cases} d_i, & i \leq n-k, \\ d_i - 1, & i > n-k. \end{cases}$$

d' is well defined since $k \leq n$, and also note that $\sum d' = \sum d - k = 2n - 2$.

We first assume that $\omega'_2 \leq 2$.

Relying on Observation 1, we find a caterpillar realization $G = (V, E)$ of d'. Let $S = \{v_1, \ldots, v_s\} \subseteq V$ be the vertices on the spine of G. Note that $|S| \geq 3$ since $d_3 \geq 3$, implying that (at least) the three highest-degree vertices are non-leaves. Let ℓ denote the number of leaves in the caterpillar. Notice that $\ell + s = n$. Also, observe that $k \geq \ell$. Out goal is to increase the degree of all leaves and of $k - \ell$ vertices in the spine by 1 while maintaining outer-planarity.

As a preliminary step we rearrange the spine such that vertices high degree vertices are closer to the center of the spine. Specifically, if s is even, the order is $v_{s-1}, \ldots, v_3, v_1, v_2, \ldots, v_s$, and if s is odd, the order is $v_{s-1}, \ldots, v_2, v_1, v_3, \ldots, v_s$. Notice that only v_s and v_{s-1} may be of degree 2. We refer to such a construction as an *ordered caterpillar.*

For step (2), we consider two cases:

Case A: $k < n$.

Case A.1: ℓ and $k - \ell$ are even.

We add a perfect matching of consecutive leaves as described in Observation 12. We also add the (matching) edges (v_i, v_{i-1}), for $i = s, s-2, \ldots, s-(k-\ell)+2$ between the vertices on the spine. The latter is feasible since $k < n$ (or $k - \ell < s$).

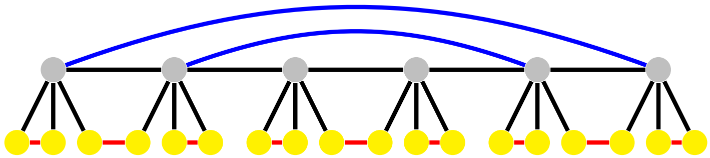


Figure 6 Illustration of the construction in Case A.1.

Case A.2: If ℓ and $k - \ell$ are odd, we add one leaf of v_{s-1} to the spine. Since $\ell - 1$ is even, we can add edges as in case A.1.

Case B: $k = n$.

Observe that n must be even, since k is even. Also, in this case, $s = k - \ell$. Let m be the number of leaves connected to spine vertices to the left of v_1 up to v_1 (including v_1).

Case B.1: s and ℓ are odd.

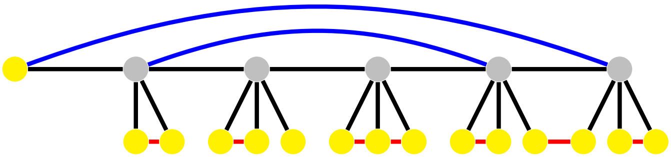


Figure 7 Illustration of the construction in Case A.2.

Case B.1(i): m is even.
Add a perfect matching of the consecutive leaves connected to $v_{s-1}, v_{s-3}, \ldots, v_1$ as described in Observation 12, add an edge connecting v_1 to the leftmost leaf of the vertex to its right (v_2 or v_3), and then add perfect matching of the rest of the leaves. In addition, add the (matching) edges (v_i, v_{i-1}), for $i = s, s-2, \ldots, 3$ between the vertices on the spine.

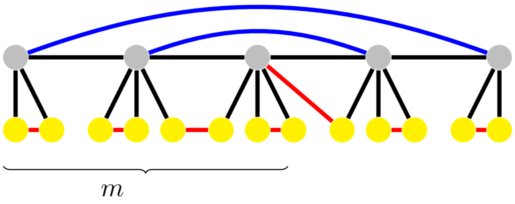


Figure 8 Illustration of the construction in Case B.1(i).

Case B.1(ii): m is odd.
Add the leftmost leaf of v_{s-1} and the rightmost leaf of v_s to the spine. Now use the same construction as in the case B.1(i).

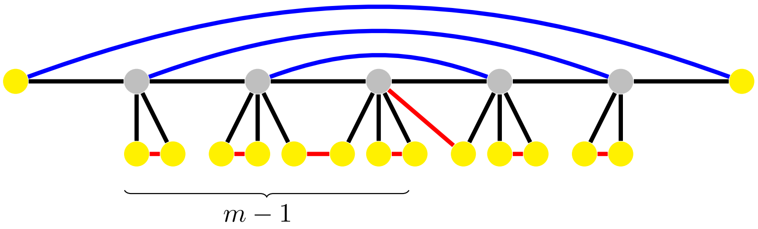


Figure 9 Illustration of the construction in Case B.1(ii).

Case B.2: s and ℓ are even.

Case B.2(i): m is odd.
Add the leftmost leaf of v_{s-1} to the spine, and use the construction of case B.1(i).

Case B.2(ii): m is even.
Add the edges (v_i, v_{i-1}), for $i = s-1, s-3, \ldots, 3$ between the vertices on the spine. Also, add the edge (v_{s-1}, v_s). In addition, replace the edge (v_{s-1}, u) with the edge (v_1, u), where u be the leftmost leaf of v_{s-1}. Add a perfect matching of consecutive leaves as described in Observation 12.

It remains to consider the case where $\omega'_2 > 2$. There are two options regarding the 2's in d'. Either all of them appear in d, or all of them originate from 3's in d.

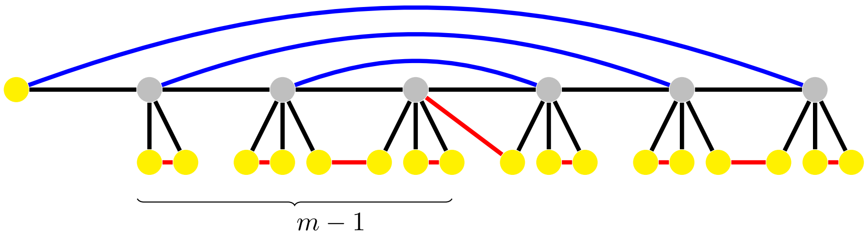


Figure 10 Illustration of the construction in Case B.2(i).

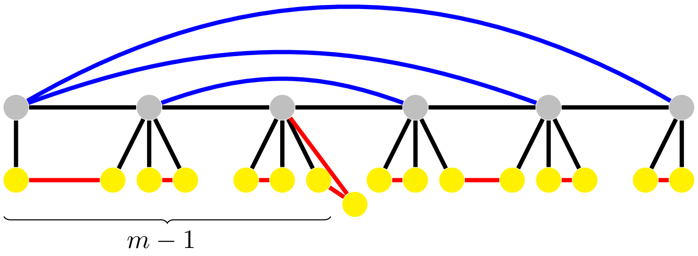


Figure 11 Illustration of the construction in Case B.2(ii).

In the first case, we obtain a sequence d'' by removing all 2's from d', and construct an outer-planar embedding for the first n'' entries by using the construction for case where $\omega_2'' \leq 2$. Consider any edge (x, y) connecting two former leaves in the above construction. We replace the edge with a path of length $\omega_2' + 1$.

In the second case, we remove $t = 2\lfloor\omega_2'/2\rfloor$ 2's from d', and construct an outer-planar embedding for the first n'' entries by using the construction for case where $\omega_2'' \leq 2$. Consider any edge (x, y) connecting two former leaves in the above construction. We replace (x, y) with a path $(x = u_0, u_1, \ldots, u_t, u_{t+1} = y)$. Then, we add the edges $\{(u_i, u_{t+1-i}) : i = 0, 1, \ldots, t/2 - 1\}$, but not in the outer-face. ◀

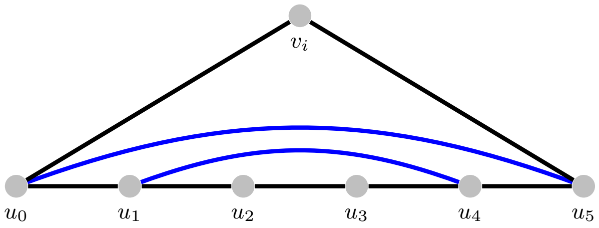


Figure 12 Illustration for the case where $\omega_2' > 2$, where $t = 4$.

Combining Lemmas 8, 9, 10, 11, 13 and Observation 7, we have the following corollary.

▶ **Corollary 14.** *Let $d \in LV_2'$.*

1. *The sequence d is outer-planaric except for $d \in EX$.*
2. *If $d \in LV_2$, then d is outer-planaric.*

Finally, we deal with sequences where $d_n = 1$.

▶ **Lemma 15.** *Every sequence $d \in LV_1$ is outer-planaric.*

Proof. Consider $d \in LV_1$. If $\sum d \leq 2n-2$, then the claim follows from Observation 1. So now assume that $\sum d > 2n-2$. Applying the MP-step of the HH method on d (see Section 2) with pivot 1 connecting to the largest degree ω_1 times yields a new sequence d', with $n' := |\mathsf{pos}(d')| = n - \omega_1$ nonzero degrees.

We claim that all the degrees in $\mathsf{pos}(d')$ are at least 2. If $d'_{n'} = 1$, then the other degrees in $\mathsf{pos}(d')$ have value 1 or 2 by the MP-step of the HH method. Therefore, $\sum \mathsf{pos}(d') \leq 2(n'-1)+1 = 2n-2\omega_1-1$, and consequently $\sum d = \sum d' + 2\omega_1 \leq 2n-1$, contradicting the assumption on $\sum d$. Furthermore, by the MP-step of the HH method, $\mathsf{pos}(d')$ is graphic, so $d'_1 \leq n'-1$. Also, $\sum \mathsf{pos}(d') = \sum d - 2\omega_1 \leq 3(n-\omega_1) - 2 = 3n'-2$. Hence, $\mathsf{pos}(d') \in LV'_2$. This and Corollary 14 imply that $\mathsf{pos}(d')$ is outer-planaric except for $\mathsf{pos}(d') \in EX$.

Notice that if $\mathsf{pos}(d')$ can be realized by an outer-planar graph G', then d is also outer-planaric, since a realizing graph G can be obtained from G' by inserting $d_i - d'_i$ new leaves to each vertex of degree d'_i. This completes the proof of the lemma for all cases except when $\mathsf{pos}(d') \in EX$, in which case it may be that d is outer-planaric yet $\mathsf{pos}(d')$ is not. For example, the sequence $d = (8, 4, 2^6, 1^2)$ is outer-planaric, as illustrated in Figure 13, but for the sub-sequence d' obtained by applying the MP-step of the HH method on d, the positive prefix $\mathsf{pos}(d') = (6, 4, 2^6)$ is not outer-planaric, by Lemma 11. So it remains to show that d is outer-planaric even if $\mathsf{pos}(d') \in EX$.

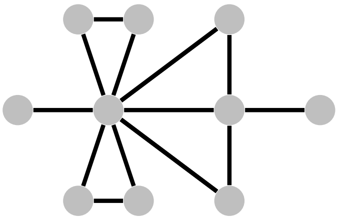


Figure 13 Outer-planar realization of the sequence $d = (8, 4, 2^6, 1^2)$.

A sequence $\mathsf{pos}(d') \in EX$ has the form $\mathsf{pos}(d') = (d'_1, d'_2, 2^{n'-2})$ where $d'_1 + d'_2 = n'+2$ and d'_1 and d'_2 are even. Since $d'_2 \geq 3$, we have $d'_2 \geq 4$.

Convert d' to $d'' = (d'_1+1, d'_2-1, 2^{n'-2})$. Note that d'' is non-increasing since $d'_2 \geq 4$. Then $d''_1 + d''_2 = d'_1 + d'_2 = n'+2$. As $d''_2 \geq 3$, we have $d''_1 \leq n'-1 = n''-1$. Also, $\sum d'' = \sum \mathsf{pos}(d') \leq 3n'-2 = 3n''-2$, so $d'' \in LV'_2$. One can check that $d'' \notin EX$, since d''_1, d''_2 are odd. By Corollary 14 (1), d'' is outer-planaric. Returning to our example of the outer-planaric $d = (8, 4, 2^6, 1^2)$ where $\mathsf{pos}(d') = (6, 4, 2^6)$ is not outer-planaric, the conversion yields $d'' = (7, 3, 2^6)$, which is outer-planaric by the construction in Lemma 10.

By the construction of d' and d'', $d_i \geq d'_i$ for $1 \leq i \leq n'$ and $d'_i \geq d''_i$ for $2 \leq i \leq n'$. As $d_1 > d'_1$, we have $d_i \geq d''_i$ for any $1 \leq i \leq n'$. Let G'' be an outer-planaric realizing graph for d''. Insert $d_i - d''_i$ leaves to the vertex with degree d''_i in G'' for any $1 \leq i \leq n'$. This yields an outer-planar graph G with degree sequence d. The lemma follows. ◀

In summary, combining Corollary 14 and Lemma 15, we get the following.

▶ **Theorem 16.** *Consider a nonincreasing n-integer graphic sequence d. If*
(1) $\omega_1 = 0$ *and* $\sum d \leq 3n-3$, *or*
(2) $\omega_1 > 0$ *and* $\sum d \leq 3n - \omega_1 - 2$,
then d is outer-planaric.

Finally, we complement the positive results of Corollary 14 and Lemma 15 by tight negative examples. Let us first consider the case of $\omega_1 = 0$. A tight example is any sequence in EX, which is not outer-planaric by Lemma 11. The volume of any sequence $d \in EX$ is

$\sum d = 3n - 2$. Next consider the case of $\omega_1 > 0$. A tight example is $d' = (3^{n-1}, 1)$ for even $n \geq 6$, which is planaric (See Figure 14) but not outer-planaric by Lemma 3. This sequence satisfies that $\sum d' = 3n - \omega_1 - 1$.

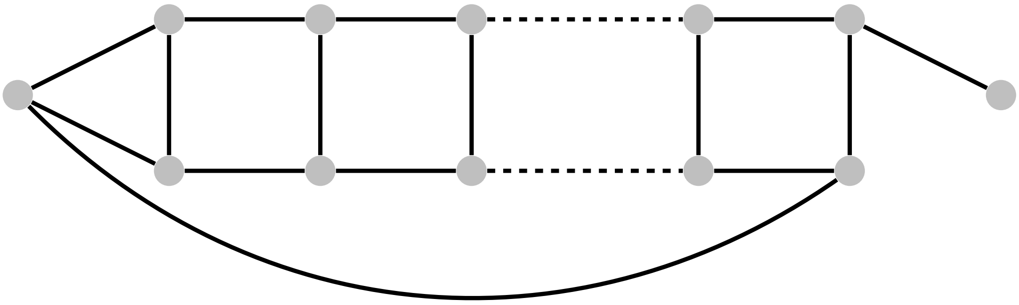


Figure 14 A non-outer-planar realization of the sequence $d = (3^{n-1}, 1)$.

References

1 Patrick Adams and Yuri Nikolayevsky. Planar bipartite biregular degree sequences. *Discr. Math.*, 342:433–440, 2019.

2 Amotz Bar-Noy, Toni Böhnlein, David Peleg, Yingli Ran, and Dror Rawitz. Degree realization by bipartite cactus graphs. Unpublished Manuscript.

3 Amotz Bar-Noy, Toni Böhnlein, David Peleg, Yingli Ran, and Dror Rawitz. Approximate realizations for outerplanaric degree sequences. In *Proc. 35th IWOCA*, 2024.

4 Amotz Bar-Noy, Toni Böhnlein, David Peleg, Yingli Ran, and Dror Rawitz. Sparse graphic degree sequences have planar realizations. In *Proc. 49th MFCS*, 2024.

5 Lowell W Beineke and Edward F Schmeichel. Degrees and cycles in graphs. *Annals of the New York Academy of Sciences*, 319(1):64–70, 1979.

6 F. T. Boesch and F. Harary. Unicyclic realizability of a degree list. *Networks*, 8:93–96, 1978.

7 Prosenjit Bose, Vida Dujmović, Danny Krizanc, Stefan Langerman, Pat Morin, David R Wood, and Stefanie Wuhrer. A characterization of the degree sequences of 2-trees. *JGT*, 58:191–209, 2008.

8 SA Choudum. Characterization of forcibly outerplanar graphic sequences. In *Combinatorics and Graph Theory*, pages 203–211. Springer, 1981.

9 Paul Erdös and Tibor Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.

10 Stefano Fanelli. On a conjecture on maximal planar sequences. *Journal of Graph Theory*, 4(4):371–375, 1980.

11 Stefano Fanelli. An unresolved conjecture on nonmaximal planar graphical sequences. *Discrete Mathematics*, 36(1):109–112, 1981.

12 Gautam Gupta, Puneet Joshi, and Amitabha Tripathi. Graphic sequences of trees and a problem of Frobenius. *Czechoslovak Math. J.*, 57:49–52, 2007.

13 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph –I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.

14 V. Havel. A remark on the existence of finite graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.

15 AF Hawkins, AC Hill, JE Reeve, and JA Tyrrell. On certain polyhedra. *The Mathematical Gazette*, 50(372):140–144, 1966.

16 Kyle F. Jao and Douglas B. West. Vertex degrees in outerplanar graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 82:229–239, 2012.

17 Zepeng Li and Yang Zuo. On the degree sequences of maximal outerplanar graphs. *Ars Comb.*, 140:237–250, 2018.

18 Zvi Lotker, Debapriyo Majumdar, N. S. Narayanaswamy, and Ingmar Weber. Sequences characterizing k-trees. In *12th COCOON*, volume 4112 of *LNCS*, pages 216–225, 2006.

19 Md Tahmidur Rafid, Rabeeb Ibrat, and Md Saidur Rahman. Generating scale-free outerplanar networks. In *Int. Computer Symp.*, pages 156–166. Springer, 2022.

20 A. Ramachandra Rao. Degree sequences of cacti. In *Combinatorics and Graph Theory*, LNM, pages 410–416, 1981.

21 S. B. Rao. A survey of the theory of potentially p-graphic and forcibly p-graphic degree sequences. In *Combinatorics and graph theory*, volume 885 of *LNM*, pages 417–440, 1981.

22 S. Rengarajan and C. E. Veni Madhavan. Stack and queue number of 2-trees. In *1st COCOON*, volume 959 of *LNCS*, pages 203–212, 1995.

23 E. F. Schmeichel and S. L. Hakimi. On planar graphical degree sequences. *SIAM J. Applied Math.*, 32:598–609, 1977.

24 Maciej M Sysło. Characterizations of outerplanar graphs. *Discrete Mathematics*, 26(1):47–53, 1979.

25 D.B. West. *Introduction to graph theory.* Prentice Hall, 2001.

Challenges of the Reachability Problem in Infinite-State Systems

Wojciech Czerwiński
University of Warsaw, Poland

Abstract

The reachability problem is a central problem for various infinite state systems like automata with pushdown, with different kinds of counters or combinations thereof. Despite its centrality and decades of research the community still lacks a lot of answers for fundamental and basic questions of that type. I briefly describe my personal viewpoint on the current state of art and emphasise interesting directions, which are worth investigating in my opinion. I also formulate several easy to formulate and understand challenges, which might be pretty hard to solve but at the same time illustrate fundamental lack of our understanding in the area.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases reachability problem, infinite-state systems, vector addition systems, pushdown

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.2

Category Invited Paper

Funding *Wojciech Czerwiński*: Supported by the ERC grant INFSYS, agreement no. 950398.

Introduction

Since the early days of computer science the reachability problem is in its core interest. One of the first known undecidable problems was the halting problem for Turing Machines (TM), which is actually exactly the reachability problem for TM. We are given an initial configuration and final configuration and we ask whether there is a path of the TM between them. It is quite easy to come up with simpler automata models for which the reachability problem is also undecidable. The undecidability will follow from the fact that the simpler models can actually simulate TM, so are Turing powerful.

The first such a model is automaton with two pushdowns or in other words with two stacks. It has two stacks and each transition can change a state and push or pop on the top of the stacks. One can easily see that the transition of a TM can be simulated by automaton with two stacks. Let us say that the TM has one tape. One stack keeps the part of the tape of the TM to the left of the head and the other stack keeps the part of the tape to the right of the head. The top elements of both stacks are the cells of the TM, which are close to its head. To simulate a transition of a TM, say one step of the head to the left, the first stack pops the top symbol and the second stack pushes the same symbol. It is rather straightforward to see that this simulation indeed works well, which shows that automata with two stacks have undecidable reachability problem.

One can however simplify further and show that the reachability problem is undecidable already for the automaton with two counters, which can be zero-tested. Configuration of such an automaton consists of a finite state and two nonnegative integer counters. A transition can change the state and increase or decrease the counters. One can assume that increases and decreases are just by one or by some arbitrary value, it does not change the strength of the model. There are also special transitions called zero-tests. Such a transition can be fired only if a specific counter (the first or the second one) equals exactly zero. This model was shown to be undecidable by Marvin Minsky [19], so it is sometimes called a Minsky machine.

The reduction from the automaton with two stacks to the automaton with two zero-tested counters is not very complicated. A very brief sketch is the following: one stack can be simulated easily by two counters with zero-tests, one keeping value of the stack as a B-ary number (where B is the number of different stack symbols) and the other one being auxiliary. Indeed, with the help of an auxiliary counter with zero-tests one can multiply or divide the value of the first counter by B and this roughly speaking corresponds to a push or pop of a symbol from the top of the stack. Thus one can simulate automaton with two stacks by an automaton with four counters with zero-tests. Further, one can observe that the auxiliary counter can be shared, so only three counters are needed for the simulation of two stacks. Finally, one can simulate values of three counters x, y and z on two counters, one counter keeping value $2^x 3^y 5^z$ and the other one being an auxiliary counter, which helps simulating incrementing and decrementing of the three counters. The above construction can be easily made precise, and it shows that the reachability problem for automata with two zero-tested counters is indeed undecidable.

Decidable models

Automata models with undecidable reachability problem are of not much use: even the simplest fundamental problem cannot be decided for them. Therefore in order to understand the computation better, the infinite-state community searched for decidable restrictions of the above models. Of course there is plenty of ways one can restrict pushdowns or counters, but there are a few natural ones, which were considered for decades. Quite natural restrictions are in my opinion the following:

(1) disallow zero-tests and consider just nonnegative integer counters without zero-tests
(2) simplify the counters even further and allow them go to below zero
(3) consider nonnegative integer counters with just one counter being zero-tested or some other restricted version of zero-tests
(4) consider one stack with possible other counters (nonnegative integer lub just integer), but without zero-tests

Of course one can come up with other models, but it turns out that the above models were widely considered.

Below we briefly discuss all the four options and also some other interesting models. Along the way I share with you my personal feelings what seems to be interesting in my opinion. I also list several challenges, which are easy to formulate, but I think require new understanding of the models we work with. Solving them might be very hard or maybe easy, but definitely would lead to new insights in the area.

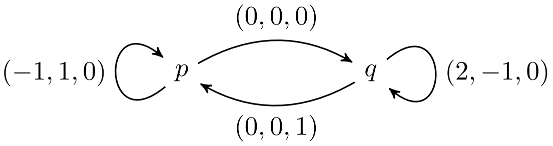
Vector addition systems

An automaton with d counters, which cannot be zero-tested on each transition just increments or decrements its counters by fixed values. It can be seen as an automaton adding vectors from $\mathbb{Z}^d$. Indeed, a configuration of such an automaton consists of a state and values of the d nonnegative integer counters, which can be seen as a vector in $\mathbb{N}^d$. A transition just adds a vector in $\mathbb{Z}^d$ to the current vector, it can be fired if the result of the addition is still a vector in $\mathbb{N}^d$. Such automata are called Vector Addition Systems with States, shortly VASS. If there is just one state they are called Vector Addition Systems, shortly VAS. It turns out that the reachability problems for VASS, for VAS and for Petri nets, a popular model of parallel computation are easily interreducible. The most robust model of out them is, in my opinion, the VASS model, so I will refer to the reachability problem in VASS. The reachability problem was shown decidable in 1981 [18], but only recently its complexity was settled to be Ackermann-complete. The upper bound was shown in 2019 [15], while the Ackermann-hardness in 2021 [6, 17]. The complexity class Ackermann is roughly speaking the class of problems, which can be solved in time around the Ackermann function. One can define easily the Ackermann function (some version of it, there are different versions, but they do not change the definition of the Ackermann complexity class as explained in [21]) using the so called hierarchy of fast growing functions F_k. We write $F_1(n) = 2n$ and $F_k(n) = \underbrace{F_{k-1} \circ \ldots \circ F_{k-1}}_{n}(1)$ for any $k \geq 2$. One can see that in particular $F_2(n) = 2^n$ and $F_3(n) = \text{Tower}(n)$. Then we define the Ackermann function $A(n) = F_n(n)$. Notice that $A(1) = 2$, $A(2) = 4$, $A(3) = 16$, while already $A(4) = \text{Tower}(\text{Tower}(\text{Tower}(\text{Tower}(1)))) = \text{Tower}(\text{Tower}(\text{Tower}(2))) = \text{Tower}(\text{Tower}(4)) = \text{Tower}(65536)$, which is an incredibly huge number. It is hard to imagine how fast the Ackermann function grows. Based on the F_k functions one can define the complexity classes $\mathcal{F}_k$, which contain roughly speaking problems solvable in time around F_k of the input size. Details can be found here [21].

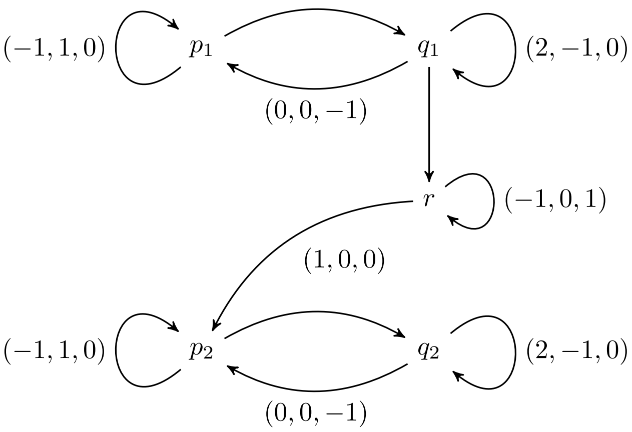
Despite the fact that the complexity of the reachability problem for VASS have been settled we still lack a lot of knowledge about the reachability properties in VASS. This can be easily seen if we look into VASS with d counters, which we also call d-dimensional VASS or shortly d-VASS. In general the complexity of the reachability problem for d-VASS is known to be in $\mathcal{F}_d$ [10]. The best currently known lower bound is $\mathcal{F}_d$-hardness in dimension $2d + 3$ [4]. The gap does not look big till we look at d-VASS for some small fixed d. Notice however, that the mentioned bounds give us Tower upper complexity bound for 3-VASS, but Tower-hardness only for 9-VASS, which is quite a big gap in understanding.

Let us summarise what is known about the reachability problem for d-VASS for small dimensions d. For small complexities it matters whether the numbers on transitions are represented in unary or in binary. Clearly binary representation is more concise, so possibly complexity of any problem for VASS with numbers represented in binary (shortly binary VASS) can be higher than the complexity for VASS with numbers represented in unary (shortly unary VASS). For dimension one it is rather easy to see that if there is a path in unary 1-VASS then there is also a path of at most cubic length, which easily implies NL algorithm and therefore also NL-completeness of the reachability problem. The complexity for binary 1-VASS is known to be NP-complete since 2009 [13]. A few years later came a big progress for 2-VASS. For binary 2-VASS it was shown in [2] that if there is a path between two configurations then there is also a path of at most exponential length. This easily implies existence of a PSpace algorithm, which just guesses this path step by step. Altogether the problem was shown in [2] to be PSpace-complete. One year later authors of [9] have proved that in unary 2-VASS if there is a path between two configurations then there is also a path of at most polynomial length, which implies algorithm in NL and thus NL-completeness.

It is a major challenge to go beyond dimension two. One particular reason for that is that reachability sets (and actually relations as well) are semilinear in 2-VASS, while it is not true for 3-VASS [14]. Recall that a set is linear of it is of a form $b + P^*$ for some vector $b \in \mathbb{N}^d$ and finite set $P \subseteq \mathbb{N}^d$ and a set is semilinear if it is a finite union of linear sets. The following example of a 3-VASS is probably the simplest way to see this and comes from the work of Hopcroft and Pansiot [14], thus we call it the HP-example. One can check that the set of configurations reachable from $p(1, 0, 0)$ in state p is of a form $\{p(x, y, z) \mid 1 \leq x + y \leq 2^z\}$, which is clearly not semilinear. Another challenge is that already in dimension three one can get VASS with finite reachability set, but of size k-fold exponential, so it is hard to obtain complexity bound below the size of the finite reachability set, so below Tower. To see that



such 3-VASS exist consider an example of a composition of a bit modified (transition from q to p now subtracts one, not adds one) two copies of the HP-example composed with itself. One can quite easily check that the reachability set from $p_1(1, 0, n)$ is doubly-exponential in n.



So see this notice that in particular any configuration of the form $p_2(x, y, 0)$ for $x + y = 2^{2^n}$ can be reached from $p_1(1, 0, n)$. Compositing k copies of the HP-example in the same fashion would result in getting $(k − 1)$-fold exponential size finite reachability set.

Unfortunately, not much upper or lower bounds are known for 3-VASS and for other d-VASS for low fixed $d \in \mathbb{N}$. I believe it is an important direction and I keep working in this area since a few years. We have some partial results with co-authors. In [5] we have shown that there exist a unary 3-VASS with two configurations such that the shortest path between them is of an exponential length. This distinguishes the situation from the unary 2-VASS, where the shortest path is always polynomial. Therefore the same way of proving membership in NL cannot work for unary 3-VASS. In fact recently together with Dmitry Chistikov, Filip Mazowiecki, Łukasz Orlikowski, Henry Sinclair-Banks and Karol Węgrzycki we have shown that the reachability problem for unary 3-VASS is NP-hard, it is a part of a paper just accepted to FOCS 2024.

On the other hand together with Ismaël Jecker, Sławomir Lasota and Łukasz Orlikowski we seem to have shown that in binary 3-VASS if there is a path between two configurations then there is one of at most doubly-exponential length. This would imply an ExpSpace membership for the reachability problem. However, the result is not yet written down, so there might be some mistake there. But, even if this result is true it might not be optimal. In every example of a 3-VASS we know, the shortest paths between two configurations are at most exponential, which would suggest PSpace algorithm. The lowest dimension in which we know that sometimes the shortest paths are of doubly-exponential length is four, in [5] we have shown an example of a 4-VASS with shortest path between some two configurations being doubly-exponential. Thus, I think that the following challenge is a very interesting one and could lead to getting a tight complexity bound for 3-VASS, which are still quite a simple and natural computation model.

▶ **Challenge 1.** *Find an example of a binary* 3*-VASS with doubly-exponential shortest path between two configurations.*

Another interesting challenge is to decide for which dimensions d reachability in d-VASS is elementary, so below Tower. The paper [4] gives us $\mathcal{F}_d$-hardness in dimension $2d + 3$, so Tower-hardness for 9-VASS. However, we have managed to show Tower-hardness already for 8-VASS in [7]. The question remains for d-VASS when $d \in \{4, 5, 6, 7\}$ (and also for $d = 3$ if our supposed ExpSpace membership for 3-VASS would turn wrong). Of course to get Tower-hardness for d-VASS we need to know examples of d-VASS with at least Tower-long shortest paths between two configurations. However, we even know no example below dimension eight with such a property. This leads to another interesting challenge.

▶ **Challenge 2.** *Find an example of a binary d-VASS with at least tower-long shortest path between two configurations for $d \leq 7$.*

Of course it might be that solving Challenges 1 and 2 is impossible, because there exist no such examples. This would however be probably hard to show, as designing faster algorithms is usually rather sophisticated. I list here challenges, which possibly might be resolved positively during one afternoon. That is why I have formulated the challenges in such a form.

Another interesting open problem for reachability in VASS is the question for fixed VASS. There is a conjecture that for every VASS V there is a constant C_V such that if there is a path from a configuration s to a configuration t then there is also one of length at most C_V times the sum of sizes of s and t [8]. Such a conjecture, if true, would imply a PSpace algorithm for the reachability problem for any fixed binary VASS V. The PSpace-hardness was already shown in [8]. I personally believe in this conjecture, but I might be wrong, so there is another hard challenge.

▶ **Challenge 3.** *Find an example of a binary VASS for which shortest paths in between its configurations are longer than linear.*

Integer VASS

Another way of relaxing infinite-state systems is to consider automata with integer counters, which cannot be zero-tested and moreover can drop below zero. Such systems are called $\mathbb{Z}$-VASS or integer VASS. Even though it is not entirely obvious, it is easy to reduce the reachability for $\mathbb{Z}$-VASS to the reachability for VASS. Every $\mathbb{Z}$-VASS counter can be simulated as a difference of two VASS counters. However, the reachability of $\mathbb{Z}$-VASS is much easier than the reachability for VASS, it is known to be NP-complete [12].

The reachability problem for automata with only $\mathbb{Z}$-counters have been well understood, as seen above. However, the power of adding $\mathbb{Z}$-counters to other systems is definitely not so fully understood. Actually, it is not clear whether an automaton with d VASS-counters and k integer-counters is closer in behaviour to a d-VASS or to $(d + k)$-VASS. One particularly interesting model is a 2-VASS with many $\mathbb{Z}$-counters. There are no known examples of these systems for which the shortest path is longer than exponential, so it is very possible that the reachability problem is in PSpace, thus PSpace-complete. On the other hand, to my best knowledge, there is no known upper complexity bound below Ackermann. Thus it is a natural challenge to investigate these systems.

▶ **Challenge 4.** *Find an example of a binary* 2*-VASS with additional $\mathbb{Z}$-counters with shortest path being longer than exponential.*

My personal conjecture is that there exists no system as mentioned above in Challenge 4, but I might be very wrong, it is always hard to imagine sophisticated examples. Notice that in the HP-example the third counter can be treated as a $\mathbb{Z}$-counter (as it never drops below zero), so reachability sets in 2-VASS with $\mathbb{Z}$-counters are not necessarily semilinear.

VASS with zero-tests

As mentioned at the beginning of this text, automata with two zero-tested counters already have undecidable reachability problem. But maybe we can allow for one zero-tested counter? If we have just an automaton with one zero-tested unary-encoded counter it is easy to show that if there is a path between two configurations then there is also a path of at most polynomial length. This means that the reachability problem is NL-complete for such automata. But, if we add more VASS-counters on top of the zero-tested counter then the situation is more sophisticated. However, in 2008 Klaus Reinhard have shown that the reachability problem for VASS with one zero-tested counter is decidable [20]. Actually, he has shown even more: decidability for VASS with hierarchical zero-tests. A d-VASS with hierarchical zero-tests (hierarchical VASS) have the following zero-tests: for each $k \leq d$ one can test whether all the first k counters are equal to zero at the same time. Interestingly, this model also has decidable reachability problem, even though it is so closed to the automaton with two zero-tested counters. Unfortunately, the proof of Reinhard was very complicated and hard to understand, so there were other tries to explain the nature of VASS with zero-tests. In 2011 Bonnet reproved the result using other techniques [3], but his work considered only VASS with one zero-test, not a hierarchical VASS. Very recently Guttenberg came up with another proof of decidability of the reachability problem for hierarchical VASS [11], this time hopefully more understandable.

Even though the decidability status of the reachability problem for hierarchical VASS is resolved its complexity is unclear. It cannot be better than Ackermann, as already the reachability problem for VASS is Ackermann-hard. However, it is interesting whether the problem is in Ackermann or maybe much higher. In particular, it would be interesting to know whether hierarchical zero-tests add anything to the power of VASS, namely whether hierarchical d-VASS are harder at some dimensions than the classical d-VASS. Interestingly, it was shown that in dimension two both models are very similar and the reachability problem for hierarchical 2-VASS is in PSpace [16]. Could this be true also in higher dimensions? It is an interesting challenge to find this out.

Pushdown VASS

Similarly as for zero-tests one can consider automata with just one stack. Reachability in a pushdown automaton is easy and can be performed in polynomial time, for example by the famous CYK algorithm. However, things become much harder if we add additional VASS-counters to this stack. An automaton with a stack and d VASS-counters is called a d-dimensional Pushdown VASS (d-PVASS). Similarly as pushdown automata are equivalent to context-free grammars the PVASS are equivalent to Grammar VAS (GVAS), which are just context-free grammars with terminals being vectors in $\mathbb{Z}^d$. A derivation of a GVAS is correct if starting in the initial vector in $\mathbb{N}^d$ and reading leaves in the prefix order one never drops below zero and finally reaches the target vector in $\mathbb{N}^d$.

Little is known about the reachability problem for PVASS. It is conjectured to be decidable, but there is no known algorithm even for 1-dimensional PVASS. An interesting result was shown by Atig and Ganty [1]. They have proved that for so called finite-index

GVAS the reachability problem is decidable. Concretely speaking they have reduced the reachability problem for finite-index d-GVAS with k nonterminals to the reachability problem for hierarchical dk-VASS, which is known to be decidable [20, 11].

Recently with Clotilde Bizière we have considered the reachability problem for 1-GVAS and we have an impression that we can show decidability by a reduction to the finite-index case. However, this result is also not confirmed and may contain a flaw. Even if it is correct the complexity of the problem for 1-GVAS remains open. We do not know any example of a 1-GVAS in which a path of length longer than exponential is needed, which would suggest low complexity. On the other hand even very simple 1-GVAS may have finite, but huge reachability sets. For example this 1-GVAS with just three nonterminals have a reachability set of Tower-size

- $X \longrightarrow 0$, $X \longrightarrow -1X2$; and
- $Y \longrightarrow 1$, $Y \longrightarrow -1YX$; and
- $Z \longrightarrow 1$; $Z \longrightarrow -1ZY$.

Indeed, with the starting nonterminal Z and starting value n we might reach up to Tower(n) values. To see this observe that nonterminal X with starting value n might output value up to $2n$. It turn the nonterminal Y with starting value n may produce up to n variables X and output value up to 2^n. Therefore nonterminal Z with starting value n may produce up to n variables Y and output value up to Tower(n). Similar grammars with d nonterminals may have finite reachability sets of size up to $F_d(n)$. So it looks interesting that we do not have any example of 1-GVAS with longer than exponential shortest path.

▶ **Challenge 5.** *Find an example of a 1-GVAS such that shortest derivations are longer than exponential.*

Other models

There are also other very interesting and natural models of infinite-state systems. One popular such system is Branching VAS (BVAS), for which the reachability problem is open in general, but was recently shown decidable in 2-dimensional BVAS. The results are not yet published, but they were referred here by Clotilde Bizière: `https://youtu.be/aKIxlgQAa2w?si=V7yWXsj8_JaoiBHv`.

Another interesting model, which illustrated how much we are in trouble with automata, which have both additive and multiplicative (or recursive) behaviour is a model about which I have heard from Anthony Lin. Consider a model, let us call it the Lin automaton, which is an automaton with one counter. A transition can increase or decrease this counter, as normally, but it can also multiply it or divide by two. Then it is unclear whether the reachability problem for such a simple automaton is decidable. Together with Moses Ganardi, Łukasz Orlikowski and Georg Zetzsche we think about this problem and we have some partial results. However, we still do not have any example of a Lin automaton, which has a path longer than exponential. Thus the following might be an interesting challenge.

▶ **Challenge 6.** *Find an example of a Lin-automaton such that shortest derivations are longer than exponential.*

To summarise, it seems that there are still many fundamental and natural challenges in the field of infinite-state systems, some of which are asking about deep properties of the computation. It might be that we are still at the beginning of our way to understand the computation even from this very high level perspective.

References

1 Mohamed Faouzi Atig and Pierre Ganty. Approximating Petri Net Reachability Along Context-free Traces. In *Proceedings of FSTTCS 2011*, volume 13 of *LIPIcs*, pages 152–163, 2011.

2 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *Proceedings of LICS 2015*, pages 32–43. IEEE Computer Society, 2015.

3 Rémi Bonnet. The Reachability Problem for Vector Addition System with One Zero-Test. In *Proceedings of MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157, 2011.

4 Wojciech Czerwinski, Ismaël Jecker, Slawomir Lasota, Jérôme Leroux, and Lukasz Orlikowski. New Lower Bounds for Reachability in Vector Addition Systems. In *Proceedings of FSTTCS 2023*, volume 284 of *LIPIcs*, pages 35:1–35:22, 2023.

5 Wojciech Czerwiński, Sławomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. Reachability in Fixed Dimension Vector Addition Systems with States. In *Proceedings of CONCUR 2020*, volume 171 of *LIPIcs*, pages 48:1–48:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

6 Wojciech Czerwiński and Łukasz Orlikowski. Reachability in Vector Addition Systems is Ackermann-complete. In *Proceedings of FOCS 2021*, pages 1229–1240. IEEE, 2021.

7 Wojciech Czerwiński and Łukasz Orlikowski. Lower Bounds for the Reachability Problem in Fixed Dimensional VASSes. In *Proceedings of LICS 2022*, pages 40:1–40:12. ACM, 2022.

8 Andrei Draghici, Christoph Haase, and Andrew Ryzhikov. Reachability in Fixed VASS: Expressiveness and Lower Bounds. In *Proceedings of FoSSaCS 2024*, volume 14575 of *Lecture Notes in Computer Science*, pages 185–205, 2024.

9 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *Proceedings of LICS 2016*, pages 477–484. ACM, 2016.

10 Yuxi Fu, Qizhe Yang, and Yangluo Zheng. Improved Algorithm for Reachability in d-VASS. In *Proceedings of ICALP 2024*, volume 297 of *LIPIcs*, pages 136:1–136:18, 2024.

11 Roland Guttenberg. Flattability of Priority Vector Addition Systems. In *Proceedings of ICALP 2024*, volume 297 of *LIPIcs*, pages 141:1–141:20, 2024.

12 Christoph Haase and Simon Halfon. Integer Vector Addition Systems with States. In *Reachability Problems – 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124, 2014.

13 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in Succinct and Parametric One-Counter Automata. In *Proceedings of CONCUR 2009*, pages 369–383. Springer Berlin Heidelberg, 2009.

14 John E. Hopcroft and Jean-Jacques Pansiot. On the Reachability Problem for 5-Dimensional Vector Addition Systems. *Theor. Comput. Sci.*, 8:135–159, 1979.

15 Jérôme Leroux and Sylvain Schmitz. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *Proceedings of LICS 2019*, pages 1–13. IEEE, 2019.

16 Jérôme Leroux and Grégoire Sutre. Reachability in Two-Dimensional Vector Addition Systems with States: One Test Is for Free. In *Proceedings of CONCUR 2020*, volume 171 of *LIPIcs*, pages 37:1–37:17, 2020.

17 Jérôme Leroux. The Reachability Problem for Petri Nets is Not Primitive Recursive. In *Proceedings of FOCS 2021*, pages 1241–1252, 2022.

18 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *Proceedings of STOC 1981*, pages 238–246, 1981.

19 Marvin L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., USA, 1967.

20 Klaus Reinhardt. Reachability in Petri Nets with Inhibitor Arcs. *Electron. Notes Theor. Comput. Sci.*, 223:239–264, 2008.

21 Sylvain Schmitz. Complexity Hierarchies beyond Elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016.

On Low Complexity Colorings of Grids

Jarkko Kari
Department of Mathematics and Statistics, University of Turku, Finland

Abstract

A d-dimensional configuration is a coloring of the infinite grid $\mathbb{Z}^d$ using a finite number of colors. For a finite subset $D \subseteq \mathbb{Z}^d$, the D-patterns of a configuration are the patterns of shape D that appear in the configuration. A configuration is said to be admitted by these patterns. The number of distinct D-patterns in a configuration is a natural measure of its complexity. We focus on low complexity configurations, where the number of distinct D-patterns is at most $|D|$, the size of the shape. This framework includes the notorious open Nivat's conjecture and the recently solved Periodic Tiling problem. We use algebraic tools to study the periodicity of low complexity configurations. In the two-dimensional case, if $D \subseteq \mathbb{Z}^2$ is a rectangle or any convex shape, we establish an algorithm to determine if a given collection of $|D|$ patterns admits any configuration. This is based on the fact that if the given patterns admit a configuration, then they admit a periodic configuration. We also demonstrate that a two-dimensional low complexity configuration must be periodic if it originates from the well-known Ledrappier subshift or from several other algebraically defined subshifts.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words; Theory of computation → Automata over infinite objects

Keywords and phrases symbolic dynamics, Nivat's conjecture, Periodic tiling problem, periodicity, low pattern complexity, annihilator

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.3

Category Invited Talk

Funding *Jarkko Kari*: Academy of Finland project 354965

1 Overview

In the *domino problem*, one is given a finite collection of allowed $m \times n$ arrays of colors, or patterns, and must determine whether it is possible to color the infinite grid $\mathbb{Z}^2$ such that only the allowed $m \times n$ patterns appear in the coloring. A classical result by Robert Berger states that the domino problem is undecidable: no algorithm can provide the correct answer to every instance of the problem. An underlying structural property is aperiodicity, where some choices of allowed patterns force the valid colorings to be non-periodic [1].

However, the situation is different in the low complexity setting where the number of allowed patterns is at most mn, the size of the array. In this case, there are no aperiodic systems, meaning that if a valid coloring of $\mathbb{Z}^2$ exists, then a periodic coloring also exists [5]. This fact leads to an algorithm to solve the domino problem in the low complexity setting. Similar results hold if, instead of $m \times n$ rectangles, one considers patterns of any convex shape $D \subseteq \mathbb{Z}^2$.

The low complexity setting also encompasses two interesting problems: Nivat's conjecture and the Periodic Tiling problem. These specific questions are of independent interest and have driven our research on this topic in recent years. Nivat's conjecture remains open, while the Periodic Tiling problem was recently solved positively by Siddhartha Bhattacharya [2] in the two-dimensional case and negatively in sufficiently high-dimensional grids by Rachel Greenfeld and Terrence Tao [3].

To address the low complexity setting, we initiated an algebraic approach in [6]. In this approach, we represent colors as integers and colorings of $\mathbb{Z}^d$ as formal Laurent power series with d variables. If the coloring has low complexity with respect to some finite shape $D \subseteq \mathbb{Z}^d$, a simple linear algebra argument shows that the corresponding power series has a non-trivial annihilator: a d-variate Laurent polynomial whose formal product with the power series vanishes. Moreover, periodicity can be simply formulated in terms of having a difference binomial annihilator. By studying the structure of the multivariate polynomial ideal of annihilators, we prove that a two-dimensional uniformly recurrent configuration with low complexity with respect to a convex shape must be periodic [5], implying the decidability of the domino problem in such a low complexity setting. We also show that low complexity elements of certain algebraic subshifts are all periodic [4].

References

1 R. Berger. *The Undecidability of the Domino Problem.* American Mathematical Society memoirs. American Mathematical Society, 1966.

2 S. Bhattacharya. Periodicity and decidability of tilings of $\mathbb{Z}^2$. *American Journal of Mathematics*, 142:255–266, 2016.

3 R. Greenfeld and T. Tao. *A counterexample to the periodic tiling conjecture.* arXiv preprint, 2022. `arXiv:2211.15847`.

4 J. Kari and E. Moutot. Nivat's conjecture and pattern complexity in algebraic subshifts. *Theoretical Computer Science*, 777:379–386, 2019.

5 J. Kari and E. Moutot. Decidability and periodicity of low complexity tilings. *Theory of Computing Systems*, 67(1):125–148, 2023.

6 J. Kari and M. Szabados. An algebraic geometric approach to Nivat's conjecture. In *Proceedings of ICALP 2015, part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 273–285, 2015.

From TCS to Learning Theory

Kasper Green Larsen
Aarhus University, Denmark

Abstract

While machine learning theory and theoretical computer science are both based on a solid mathematical foundation, the two research communities have a smaller overlap than what the proximity of the fields warrant. In this invited abstract, I will argue that traditional theoretical computer scientists have much to offer the learning theory community and vice versa. I will make this argument by telling a personal story of how I broadened my research focus to encompass learning theory, and how my TCS background has been extremely useful in doing so. It is my hope that this personal account may inspire more TCS researchers to tackle the many elegant and important theoretical questions that learning theory has to offer.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Theoretical Computer Science, Learning Theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.4

Category Invited Paper

Funding *Kasper Green Larsen*: Supported by a DFF Sapere Aude Research Leader Grant No. 9064-00068B.

1 A Personal Story

My goal with this invited abstract, is to inspire more theoretical computer science (TCS) researchers to engage in topics in machine learning theory. Being a TCS researcher myself, I hope that by telling my own personal story – of how I adopted the field of learning theory – may provide a unique and more compelling argument than if a core learning theorist was to attempt to convince you. Being a personal story, I will focus on some of the research results I obtained along the way, and not so much on the broader learning theory and TCS literature. The results I present have been chosen because they tell an interesting story and showcase how core TCS concepts have proven useful in learning theory. Along the way, I will tell some anecdotes on how these research projects started, and I will try to give you an idea of some of the thought processes I initially went through when trying to enter a new field. I have also tried to sprinkle in some advice for more junior researchers based on my own experiences. Finally, I will also formally define some of the learning theory questions I have studied, giving you an idea of what such questions may look like. I sincerely hope that you find this somewhat unusual abstract refreshing.

1.1 My TCS Background and Connections Between Fields

For those who do not know me, let me start by briefly giving you my background, while also trying to convince you that many fields of TCS are deeply connected, and that it pays off to have a broad interest in TCS.

I started my Ph.D. studies in 2008 under the supervision of Professor Lars Arge at Aarhus University, Denmark. My initial research was on data structures, in particular so-called *I/O-efficient* data structures [2, 5], that take the memory hierarchy of modern hardware into account. Ever since I started my studies, I have always been curious and eager to learn new areas by interacting with great colleagues in the TCS community. This openness to collaborations and sharing of ideas is one of the strongest and most rewarding traits of our community.

Initially, my interactions with other researchers were through fellow students at the department. I explicitly remember Allan Grønlund coming to my office with one of Mihai Pătraşcu's beautiful papers [33] on data structure lower bounds in the cell probe model [35], excitedly proclaiming that "*this is so cool, we need to do this too!*". At the time, I had never had a complexity course or seen a real lower bound proof, and I was immediately hooked. This led me on a long research journey trying to understand and develop the techniques for proving lower bounds on the efficiency (space, query time, update time) of data structures, see e.g. [23, 24, 29] for some highlights.

While cell probe lower bounds were my main focus for years, I always found research that bridges several fields the most exciting. The earliest such example in my own research started with a visit by Jeff M. Phillips in our research group. During his visit, Jeff introduced me to *discrepancy theory* while explaining some of his recent work. As discrepancy theory will be important later, let me give a rough definition of it here. At its core, discrepancy theory studies the following problem: Given a matrix $A \in \mathbb{R}^{m \times n}$, compute a *coloring* $x \in \{-1, 1\}^n$ minimizing the discrepancy $\|Ax\|_\infty$. Understanding the best achievable discrepancy for various types of matrices is an old and very well-studied topic in TCS [3, 34, 10, 7, 31]. In particular, I recall Jeff telling me about a result of Banaszczyk [6] showing that matrices with sparse columns always have low discrepancy colorings. While being intellectually intrigued by this result, I did not see any immediate applications of it in my own research. However, a couple of months later, Elad Verbin, a post doc in our group at the time, suggested to me that I should try to prove data structure lower bounds in the *group model.* Without going into details here, it turns out that Banaszczyk's discrepancy result could be used to directly translate decades of research on discrepancy lower bounds into group model data structure lower bounds [25]. The connection even improved some of the discrepancy upper bounds using data structure upper bounds as well.

This theme of connecting areas has proven very useful over the years. It for instance got me started on *streaming algorithms* when Jelani Nelson and Huy L. Nguyen asked whether we could use cell probe lower bound techniques to prove *time* lower bounds for streaming algorithms [27]. In later work with Jelani [26], we similarly proved optimality of the Johnson-Lindenstrauss (JL) lemma [21] in *dimensionality reduction* using an encoding-based argument. Such arguments were typically used in cell probe data structure lower bounds. As it will be important later, let us recall that the JL lemma says that any set of n points in $\mathbb{R}^d$ can be embedded into $O(\varepsilon^{-2} \log n)$ dimensions while preserving all pairwise distances to within a factor $1 \pm \varepsilon$.

Another interesting example was initiated during a visit at MIT in 2017 where Vinod Vaikuntanathan was telling me how "*data structures and crypto are a match made in heaven*". This claim got me curious, and after returning home, I read a bit on data structure in cryptography and stumbled on Oblivious RAMs (ORAMs) [17]. An ORAM is basically a primitive for obfuscating the memory access pattern of an algorithm, such that the memory accesses reveal nothing about the data stored. Checking the references of the papers I was reading, it turned out that a crypto Professor sitting just down stairs, Jesper Buus Nielsen, had made multiple contributions to ORAMs. After some brief initial discussions, we quickly realized that cell probe lower bound techniques could be tweaked to prove strong lower bounds for ORAMs [28]. This connection started a whole line of research into lower bounds for ORAMs and related primitives in crypto.

Finally, let me mention one last example outside learning theory. Back in 2017, I was attending a communication complexity workshop in Banff, where Mark Braverman was giving a talk about some work of his [11] on a conjecture in information theory/network coding [30]. The conjecture relates to communication networks, where a network is modeled by a graph with capacities on the edges, designating how many bits may be sent across. The conjecture then says that for undirected graphs (messages may be sent in both directions), if we are given k source-sink pairs that each need to send an r bit message from source to sink, then the largest number of bits r we can handle without violating capacity constraints, is equal to the multi-commodity flow rate. Intuitively, this means that the best you can do is to simply forward your messages as indivisible bits. Mark's talk was excellent, but at the time, I had no immediate applications of his result. However, while spending a semester at the Simons Institute in 2018, a coincidental conversation with MohammadTaghi Hajiaghayi led us [15] to proving conditional lower bounds for I/O-efficient algorithms using the information theory conjecture that Mark presented. Needless to say, it was quite satisfactory to come full circle and address the topic that started my Ph.D. studies. In addition to the lower bounds for I/O-efficient algorithms, the conjecture also gave a clean conditional $\Omega(n \log n)$ lower bound for constant degree boolean circuits for multiplication [1]. I find it quite fascinating how two such seemingly different problems of multiplication and communication in graphs may prove to be so intimately connected.

If there is one message to take away from my own experiences, in particular for junior researchers, it must be that many things are deeply connected and you never know when results in one branch of TCS may prove useful in another. That is one of its beauties. I would thus strongly recommend that you attend talks, seminars, and generally engage in discussions with the many brilliant TCS researchers, also in fields outside your own immediate interest. Build your expertise in one area, but always be curious and look for connections and inspiration from others.

1.2 Entering Learning Theory

Now let me get to the promised topic of entering learning theory from a TCS background, and how this background proved extremely useful. For me, this journey started around 2019. As you all know, machine learning was the big hype at the time, and probably still is. Like many others, I initially found deep neural networks and what they could do quite fascinating. This led me to consider whether I should get into machine learning and work on this extremely hot topic. However, after getting a slightly better understanding of the field, I quickly found that training a deep neural net and running some experiments on a benchmark data set, to be much more engineering and craftsmanship than deep stimulating thoughts. And with all the obligations that come with a faculty position, I just did not have the time to learn the practical skills that it takes to make efficient and competitive implementations. And probably was not that interested in it after all. I almost abandoned machine learning, had it not been for a student of mine, Alexander Mathiasen. Alexander was very independent and eager to get into machine learning. He had realized that I was more interested in theory questions and had on his own discovered the great source of open problems in learning theory that are published with the COLT proceedings. The open problem Alexander had found was related to a technique called *boosting*. Let me start by introducing the general idea in boosting and the setup for binary classification in supervised learning.

Supervised Learning and Boosting

In supervised learning, a binary classification problem is specified by an input domain $\mathcal{X}$, an unknown target function $f : \mathcal{X} \to \{-1, 1\}$ and an unknown data distribution $\mathcal{D}$ over $\mathcal{X}$. A learning algorithm receives as input a training data set of n i.i.d. samples $(x_i, f(x_i))$ with each $x_i \sim \mathcal{D}$. The goal of the learning algorithm is to output a classifier $h : \mathcal{X} \to \{-1, 1\}$ minimizing the probability of misprediction the label of a new sample from $\mathcal{D}$, i.e. where $\mathrm{er}_{\mathcal{D}}(h) := \Pr_{x \sim \mathcal{D}}[h(x) \neq f(x)]$ is as small as possible. To have an example in mind, think of $\mathcal{X}$ as the set of all images of a particular size and f as the hard-to-specify function that maps every image of a mammal to -1 and every other image to 1. The learning problem is thus to train a classifier that can detect whether an image contains a mammal or not.

Boosting is then a powerful and elegant idea that allows one to combine multiple inaccurate classifiers into a highly accurate *voting classifier*. Boosting algorithms such as AdaBoost [16] work by iteratively running a base learning algorithm on reweighed versions of the training data to produce a sequence of classifiers $h_1, \dots, h_t$. After obtaining h_i, the weighting of the training data is updated to put larger weights on samples misclassified by h_i, and smaller weights on samples classified correctly. This effectively forces the next training iteration to focus on points with which the previous classifiers struggle. After sufficiently many rounds, the classifiers $h_1, \dots, h_t$ are finally combined by taking a (weighted) majority vote among their predictions. This idea also goes under the name of *multiplicative weight updates* and has many applications in TCS.

Boosting works exceptionally well in practice and much theoretical work has gone into explaining its huge success. One important line of work in this direction is based on the notion of *margins* [8]. A voting classifier g may be written as $g(x) = \mathrm{sign}(\sum_{i=1}^{t} \alpha_i h_(x))$ with $\alpha_i \in \mathbb{R}$. If we normalize the weights so that $\sum_i |\alpha_i| = 1$, then $\sum_{i=1}^{t} \alpha_i h_i(x) \in [-1, 1]$. The margin of g on a sample $(x, y) \in \mathcal{X} \times \{-1, 1\}$ is then defined as $y \sum_{i=1}^{t} \alpha_i h_i(x)$. The margin is thus a number in $[-1, 1]$ and can be thought of as a confidence in the prediction. If the margin is 1, then all classifiers combined by g agree and are correct. If the margin is 0, then we have a (weighted) 50-50 split between the two possible predictions $-1/1$, and so on. It has then been proved that voting classifiers g with large margins on all training samples (x_i, y_i) have a small $\mathrm{er}_{\mathcal{D}}(g)$ [8].

Boosting and Discrepancy Theory

Now let me return to the open problem Alexander approached me with. Since voting classifiers with large margins have a small $\mathrm{er}_{\mathcal{D}}(g)$, it seems natural to develop boosting algorithms that obtain large margins by combining few base classifiers, i.e. with a small t. This leads to faster predictions and possibly also faster training. The question now is as follows: If the best possible margin on all training samples is $\gamma^\star$ when one is allowed to combine arbitrarily many base classifiers h_i, then how large margins can we obtain by combining t base classifiers? The best known algorithm obtained margins of $\gamma^\star - O(\sqrt{\log(n)/t})$ with n training samples, and the best known lower bound showed that this is tight whenever $t \leq n^{1/2}$. The conjecture stated that this remains tight for $t \leq n \log n$.

In my first learning theory paper [32], we observed an interesting connection between this question and discrepancy theory. In more detail, we considered the matrix A obtained by forming a row for each training sample (x_i, y_i) and a column for each h_i in a voting classifier $g(x) = \mathrm{sign}(\sum_{j=1}^{t} \alpha_j h_j(x))$. The entry corresponding to (x_i, y_i) and h_j stores the value $y_i \alpha_j h_j(x)$. Notice that the sum of the entries in the i'th row is precisely the margin of g on (x_i, y_i). Next, if we can find a coloring $x \in \{-1, 1\}^t$ such that $\|Ax\|_\infty \approx 0$, then this means that for every single row of A, the sum over the columns j where $x_j = 1$ is almost exactly the same as the sum over columns where $x_j = -1$. Thus if we replace g by a new voting classifier where we set all α_j to 0 when x_j equals the majority of $-1/1$'s in x, and to 2 when x_j is the minority, then the new g' has almost exactly the same margins as before but uses only half as many base classifiers. Recursively finding a low discrepancy coloring x using the celebrated *Spencer's six standard deviations suffice* result [34], we were able to refute the conjecture on the tradeoff between t and the achievable margin while establishing yet an interesting connection between discrepancy theory and other areas.

Support Vector Machines, Sketching and Teaching

Around the same time as our first learning theory results, I had voluntarily started teaching the Machine Learning undergraduate course at Aarhus University. Never having followed such a course myself, this was a great way of forcing myself to learn the basics. But beyond learning the basics, teaching also led to new research questions. After having completed the first project on boosting, we naturally started reading up on the references proving that large margins lead to small $\mathrm{er}_{\mathcal{D}}(g)$. Such results are referred to as *generalization bounds* and are a cornerstone of learning theory. Studying the classic proofs of generalization for large margin voting classifiers [8], there seemed to be an underlying idea of *sketching* or *compressing* the voting classifier g while exploiting that it has large margins. Intuitively, if there is a small-bit representation of g, then there are only few choices for g. A union bound over all these choices shows that it is unlikely that there is even a single g that performs great on the training data (has large margins) and poor on new data (has large $\mathrm{er}_{\mathcal{D}}(g)$).

Around the same time, I had just taught Support Vector Machines (SVMs) [14]. SVMs are another type of learning algorithm where the input domain $\mathcal{X}$ is $\mathbb{R}^d$. In the simplest setup, when given a training set of n samples $(x_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$ with $\|x_i\| \leq 1$ for all i, SVM searches for the separating hyperplane with the largest *margin* to the data. In the context of SVMs, if we assume for simplicity that a hyperplane passes through the origin and has unit length normal vector $w \in \mathbb{R}^d$, then we predict the label of a new point $x \in \mathbb{R}^d$ by returning $\mathrm{sign}(w^T x)$. The margin of the hyperplane on a labeled point (x, y) is then defined as $yx^T w$. The margin is positive if the hyperplane correctly predicts the label of (x, y) and the absolute value of the margin measures how far the point is from the hyperplane itself.

Similarly to the case for boosting, there were known generalization bounds [9] stating that if a hyperplane h has large margins, then $\mathrm{er}_{\mathcal{D}}(h)$ is small. However, these bounds were proved in a completely different way than the boosting bounds and seemed to be sub-optimal. In our work [18], we improved these generalization bounds by observing a connection to the Johnson-Lindenstrauss (JL) lemma [21] that I previously worked on [26]. Shortly after the lower bound for JL by Jelani and I, Noga Alon and Bo'az Klartag [4] presented an alternative proof that at its core gives a randomized sketch for representing a set of n unit length vectors in $\mathbb{R}^d$ using $O(\varepsilon^{-2} \log n)$ bits for each vector, while preserving their pairwise inner products up to additive ε. Our idea was then to apply this sketch to w and the training data. Since $w^T x$ changes by only additive ε, the sketched hyperplane $\tilde{w}$ remains correct (i.e. $\mathrm{sign}(\tilde{w}^T x) = \mathrm{sign}(w^T x) = y$) if the margin was larger than ε before sketching. In this way, we get a compression whose size depends on the margin. Combining this with the union bound idea mentioned earlier led to improved generalization bounds for SVMs.

Replicable Learning and Sketching

Let me give another example where the JL lemma proved useful in learning theory. In a very exciting line of work, starting with [20], core TCS researchers introduced the notion of *replicable learning* as an attempt at addressing issues with reproducibility of results in machine learning. Here the basic setup is that you want to develop learning algorithms $\mathcal{A}$, such that if $\mathcal{A}$ is run on two sets S and S' of n i.i.d. samples from the same distribution $\mathcal{D}$, then we should get the same output with high probability. The intuition is thus, if you rerun someones replicable algorithm on your own data set, you will get the same results as they did on their data set, provided that the data set is sampled from the same distribution. To help you in this seemingly difficult task, the two executions of $\mathcal{A}$ share a random seed. This may be justified in practice by publishing the seed of a pseudorandom generator along with your machine learning paper and experimental results. There is naturally a ton of problems to revisit in replicable learning and some strong connections with *differential privacy* have already been established [12].

In a recent result [22], we considered SVMs in the replicable setting. Here we are promised that there exists a hyperplane with all margins at least γ on the support of the data distribution $\mathcal{D}$, and must replicably compute a hyperplane h with small $\mathrm{er}_{\mathcal{D}}(h)$. The key idea in our work is quite simple: partition the training data into t chunks and run SVM on each to obtain hyperplanes with normal vectors $w_1, \dots, w_t$. Average these normal vectors $w = t^{-1} \sum_i w_i$ and apply the sketching technique by Alon and Klartag [4] to w. The first main observation is that computing the average w reduces variance compared to each w_i. Thus for two independent but identically distributed training data sets S and S', the resulting w and w' will be close. The key property of the sketching technique of Alon and Klartag is that such close vectors are very likely to be "rounded" to the same hyperplane/sketch if we share a carefully chosen random seed.

Multi-Distribution Learning, Discrepancy Theory and NP-Hardness

Let me conclude with one last example of TCS techniques playing a key role in learning theory results. A recent line of work in learning theory has studied learning in a setup with multiple data distributions. Formally, we have k distributions $\mathcal{D}_1, \dots, \mathcal{D}_k$ over $\mathcal{X} \times \{-1, 1\}$ and the goal is to train a classifier that performs well on all k distributions simultaneously. For this purpose, we are given a hypothesis set $\mathcal{H}$ containing hypotheses $h : \mathcal{X} \to \{-1, 1\}$. As an example, think of $\mathcal{H}$ as all hyperplanes in $\mathbb{R}^d$. If we let $\tau = \min_{h \in \mathcal{H}} \max_i \mathrm{er}_{\mathcal{D}_i}(h)$ denote the best achievable max-error of any $h \in \mathcal{H}$, then the goal is produce a classifier $f : \mathcal{X} \to \{-1, 1\}$ with $\max_i \mathrm{er}_{\mathcal{D}_i}(f) \leq \tau + \varepsilon$ using as few training samples as possible.

It is known that for a single distribution $\mathcal{D}$, this task requires $\Theta(d/\varepsilon^2)$ samples and it is straight forward to generalize the algorithm to $O(dk/\varepsilon^2)$ samples for k distributions. Very surprisingly, it turns out that this can be improved to essentially $O((d+k)/\varepsilon^2)$ samples using boosting ideas [19]. However, the upper bounds achieving this number of samples are cheating a little. Concretely, they do not output a single hypothesis f with $\max_i \mathrm{er}_{\mathcal{D}_i}(f) \leq \tau + \varepsilon$. Instead, they output a distribution $\mathcal{D}_f$ over hypotheses such that $\max_i \mathbb{E}_{f \sim \mathcal{D}_f}[\mathrm{er}_{\mathcal{D}_i}(f)] \leq \tau + \varepsilon$. Attempting to *derandomize* this does not seem to work as we only have Markov's inequality to argue that with constant probability, a random $f \sim \mathcal{D}_f$ has $\max_i \mathrm{er}_{\mathcal{D}_i}(f) = O(k(\tau + \varepsilon))$. In a current manuscript, we show that there is a good reason why this is the case. Concretely, we prove via a reduction from discrepancy minimization that it is NP-hard to compute an $f : \mathcal{X} \to \{-1, 1\}$ with $\max_i \mathrm{er}_{\mathcal{D}_i}(f) \leq \tau + \varepsilon$. Here we use a result of Charikar et al. [13] stating that it is NP-hard to distinguish whether, for a given a 0/1 matrix A, there exists an $x \in \{-1,1\}^n$ with $Ax = 0$, or whether all $x \in \{-1,1\}^n$ have $\|Ax\|_\infty = \Omega(\sqrt{n})$. In our reduction, we think of the columns as the input domain $\mathcal{X}$ and we form a distribution for each row of A. The key idea is to show that if a learning algorithm computes an $f : \mathcal{X} \to \{-1,1\}$ with $\max_i \mathrm{er}_{\mathcal{D}_i}(f) \le \tau + \varepsilon$, then the predictions made by f on the columns must give a low discrepancy coloring that can distinguish the two cases.

While showing NP-hardness is by now completely standard in TCS, let me add that my two coauthors are both from Statistics. They had heard NP-hardness mentioned before, but never used it, and found quite some joy in getting to apply it on a relevant problem. I guess the experience shows that what may be common tools in one discipline may be highly novel in another.

1.3 Conclusion and Thoughts

To summarize my research journey into learning theory, I hope that I convinced you that many techniques and tricks from TCS are extremely useful. Not only do they help in addressing learning theory questions, but they also bring an interesting new direction to the field, such as e.g. arguing computational hardness of training a classifier. Furthermore, this transfer of techniques is not isolated to learning theory, but has been a guiding principle in all my research since the early days of my Ph.D. studies. Keep your eyes open for surprising connections between seemingly disjoint topics - they are often hiding just below the surface.

To me, the questions studied in learning theory are as clean, elegant, beautiful and well-defined as any TCS topic, and are very well fit for anyone with a TCS background. What has worked well for me, has been to start in some corner (coincidentally boosting for me) and get to know the literature and related questions. Then as you have a better understanding and overview, you can start approaching neighboring questions and growing your focus. I suppose this is not much different from starting your Ph.D. studies by working on a narrow topic and broadening out as you mature. Finally, do not underestimate the value of talking to peers and attending talks for inspiration. As a newcomer, it is sometimes hard to ask the right question, and calibrating with an expert is very useful. Also, the COLT open problems have been a good source of inspiration as you at least know that some experts in the field find the question interesting.

I strongly hope that this abstract may inspire some of you to work on learning theory, or give you the courage to enter a new field that you find intriguing, knowing that your TCS skills are useful in many surprising places.

References

1 Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen. Lower bounds for multiplication via network coding. In *ICALP*, volume 132 of *LIPIcs*, pages 10:1–10:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

2 Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, September 1988. doi:10.1145/48529.48535.

3 R. Alexander. Geometric methods in the study of irregularities of distribution. *Combinatorica*, 10(2):115–136, 1990.

4 Noga Alon and Bo'az Klartag. Optimal compression of approximate inner products and dimension reduction. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 639–650, 2017. doi:10.1109/FOCS.2017.65.

5 Lars Arge. The buffer tree: A new technique for optimal i/o-algorithms. In Selim G. Akl, Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures*, pages 334–345, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

6 Wojciech Banaszczyk. Balancing vectors and gaussian measures of n-dimensional convex bodies. *Random Structures & Algorithms*, 12:351–360, July 1998.

7 Nikhil Bansal. Constructive algorithms for discrepancy minimization. In *Proc. 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 3–10, 2010.

8 Peter Bartlett, Yoav Freund, Wee Sun Lee, and Robert E. Schapire. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

9 Peter Bartlett and John Shawe-Taylor. *Generalization performance of support vector machines and other pattern classifiers*, pages 43–54. MIT Press, Cambridge, MA, USA, 1999.

10 J. Beck and T. Fiala. Integer-making theorems. *Discrete Applied Mathematics*, 3:1–8, February 1981.

11 Mark Braverman, Sumegha Garg, and Ariel Schvartzman. Coding in undirected graphs is either very helpful or not helpful at all. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 18:1–18:18, 2017.

12 Mark Bun, Marco Gaboardi, Max Hopkins, Russell Impagliazzo, Rex Lei, Toniann Pitassi, Satchit Sivakumar, and Jessica Sorrell. Stability is stable: Connections between replicability, privacy, and adaptive generalization. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 520–527, 2023.

13 Moses Charikar, Alantha Newman, and Aleksandar Nikolov. Tight hardness results for minimizing discrepancy. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1607–1614. SIAM, 2011. doi:10.1137/1.9781611973082.124.

14 Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

15 Alireza Farhadi, MohammadTaghi Hajiaghayi, Kasper Green Larsen, and Elaine Shi. Lower bounds for external memory integer sorting via network coding. In *STOC*, pages 997–1008. ACM, 2019.

16 Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

17 Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 1996.

18 Allan Grønlund, Lior Kamma, and Kasper Green Larsen. Near-tight margin-based generalization bounds for support vector machines. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 3779–3788. PMLR, 2020.

19 Nika Haghtalab, Michael I. Jordan, and Eric Zhao. On-demand sampling: Learning optimally from multiple distributions. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/02917acec264a52a729b99d9bc857909-Abstract-Conference.html.

20 Russell Impagliazzo, Rex Lei, Toniann Pitassi, and Jessica Sorrell. Reproducibility in learning. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 818–831. ACM, 2022. doi:10.1145/3519935.3519973.

21 William Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, January 1984. doi:10.1090/conm/026/737400.

22 Alkis Kalavasis, Amin Karbasi, Kasper Green Larsen, Grigoris Velegkas, and Felix Zhou. Replicable learning of large-margin halfspaces. In *ICML*, Proceedings of Machine Learning Research. PMLR, 2024. To appear.

23 Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *STOC*, pages 85–94. ACM, 2012.

24 Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *FOCS*, pages 293–301. IEEE Computer Society, 2012.

25 Kasper Green Larsen. On range searching in the group model and combinatorial discrepancy. *SIAM J. Comput.*, 43(2):673–686, 2014.

26 Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *FOCS*, pages 633–638. IEEE Computer Society, 2017.

27 Kasper Green Larsen, Jelani Nelson, and Huy L. Nguyên. Time lower bounds for nonadaptive turnstile streaming algorithms. In *STOC*, pages 803–812. ACM, 2015.

28 Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In *CRYPTO (2)*, volume 10992 of *Lecture Notes in Computer Science*, pages 523–542. Springer, 2018.

29 Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. In *STOC*, pages 978–989. ACM, 2018.

30 Zongpeng Li and Baochun Li. Network coding: the case of multiple unicast sessions. In *Proceedings of the 42nd Allerton Annual Conference on Communication, Control and Computing*, Allerton'04, 2004.

31 Shachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. *SIAM Journal on Computing*, 44(5):1573–1582, 2015.

32 Alexander Mathiasen, Kasper Green Larsen, and Allan Grønlund. Optimal minimal margin maximization with boosting. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 4392–4401. PMLR, 2019.

33 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. doi:10.1137/09075336X.

34 Joel Spencer. Six standard deviations suffice. *Trans. Amer. Math. Soc.*, 289:679–706, 1985.

35 Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, July 1981. doi:10.1145/322261.322274.

Fine-Grained Complexity of Program Analysis

Rupak Majumdar
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

There is a well-known "cubic bottleneck" in program analysis and language theory: many program analysis problems can be solved in time cubic in the size of the input but, despite years of effort, there are no known sub-cubic algorithms. For example, context-free reachability (whether there is a path in a labeled graph that is labeled with a word from a context-free language), the emptiness problem for pushdown automata, and the recognition problem for two-way nondeterministic pushdown automata all belong to the cubic class. We survey the status of these problems through the lens of fine-grained complexity.

We study the related *certification* task: given an instance of any of these problems, are there small and efficiently checkable certificates for the existence and for the non-existence of a path? We show that, in both scenarios, there exist succinct certificates ($O(n^2)$ in the size of the problem) and these certificates can be checked in subcubic (matrix multiplication) time. Thus, all these problems lie in nondeterministic and co-nondeterministic *subcubic* time.

We also study a hierarchy of program analysis problems above the cubic bottleneck. A representative problem here is the recognition problem for two-way nondeterministic pushdown automata with k heads. We show fine-grained hardness results for this hierarchy.

We also discuss purely language-theoretic consequences of these results: for example, we obtain hardest languages accepted by two-way nondeterministic multihead pushdown automata, as well as separations between language classes.

(Joint work with A. R. Balasubramanian, Dmitry Chistikov, and Philipp Schepper.)

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Theory of computation → Problems, reductions and completeness; Software and its engineering → Compilers

Keywords and phrases Fine-grained complexity, CFL reachability, 2NPDA recognition, PDA emptiness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.5

Category Invited Talk

Funding This research was partially funded by the DFG project 389792660 TRR 248–CPEC.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 5; pp. 5:1–5:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Monotonicity of the Cops and Robber Game for Bounded Depth Treewidth

Isolde Adler
University of Bamberg, Germany

Eva Fluck
RWTH Aachen University, Germany

Abstract

We study a variation of the cops and robber game characterising treewidth, where in each round at most one cop may be placed and in each play at most q rounds are played, where q is a parameter of the game. We prove that if k cops have a winning strategy in this game, then k cops have a monotone winning strategy. As a corollary we obtain a new characterisation of bounded depth treewidth, and we give a positive answer to an open question by Fluck, Seppelt and Spitzer (2024), thus showing that graph classes of bounded depth treewidth are homomorphism distinguishing closed.

Our proof of monotonicity substantially reorganises a winning strategy by first transforming it into a pre-tree decomposition, which is inspired by decompositions of matroids, and then applying an intricate breadth-first "cleaning up" procedure along the pre-tree decomposition (which may temporarily lose the property of representing a strategy), in order to achieve monotonicity while controlling the number of rounds simultaneously across all branches of the decomposition via a vertex exchange argument. We believe this can be useful in future research.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases tree decompositions, treewidth, treedepth, cops-and-robber game, monotonicity, homomorphism distinguishing closure

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.6

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2402.09139` [4]

1 Introduction

Search games were introduced by Parsons and Petrov in [36, 37, 38] and since then gained much interest in many (applied and theoretical) areas of computer science and in discrete mathematics [6, 10, 9, 28, 17, 35, 25, 15, 24, 21]. In search games on graphs, a fugitive and a set of searchers move on a graph, according to given rules. The searchers' goal is to capture the fugitive, and the fugitive tries to escape. Here the interest lies in minimising the resources needed to guarantee capture. Typically this means minimising the number of searchers, but we also seek to bound the number of rounds of the game, if the searchers can only move one by one. Search games have proven very useful for providing a deep understanding of structural and algorithmic properties of width parameters of graphs, such as treewidth [8, 43], pathwidth [9], cutwidth [30], directed treewidth [26], treedepth [33], and b-branched treewidth [14, 32].

The crux in relating a given variant of a search game to a width parameter often lies in the question of whether the game is *monotone*, i. e. whether the searchers always have a winning strategy in which a previously cleared area never needs to be searched again – without needing additional resources. Furthermore, monotonicity of a search game provides a polynomial space certificate for proving that determining the winner is in NP.

In their classic paper [43], Seymour and Thomas proved monotonicity of the cops and robber game characterising treewidth. They use a very elegant inductive argument via the dual concept of *brambles*. In this paper we study a variation of this game, where k cops try

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 6; pp. 6:1–6:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to capture a robber, but they are limited to placing at most one cop per round and playing at most q rounds, for a fixed number $q \in \mathbb{N}$. It is an open question from [13], whether this game is monotone. We give a positive answer to this question.

The notion of *treedepth* was first introduced by Nešetřil and Ossona de Mendes [33]. They exhibit a number of equivalent parameters, and a characterisation by a monotone game is implicitely given. This was subsequently made more explicit in [19]. In [18], a characterisation by a different game called *lifo game* is given for which monotonicity is proven. The game we study can be seen as generalising the monotone game implicit in [33]. However, it also captures treewidth and it is not monotone by definition.

Recently, width parameters received a renewed interest in the context of counting homomorphisms and the expressive power of logics [12, 20, 11, 41, 13]. In this context a non-monotone search game characterisation of the width parameter is useful to ensure that there are no graphs of higher width that can be added to the graph class without changing the expressive power of the logic [34, 13]. The main obstacle then is to find such a non-monotone characterisation, as the natural characterisation as a search-game of many graph parameters is inherently monotone. *Bounded depth treewidth* and the game studied in this paper were first defined in [13]. An equivalent characterisation of these graph classes by so-called k-pebble forest covers of depth q, which is bounded width treedepth, was already given in [1].

Homomorphism Counts. Homomorphism counts are an emerging tool to study equivalence relations between graphs. Many equivalence relations between graphs can be characterized as homomorphism indistinguishability relations, these include graph isomorphism [29], graph isomorphism relaxations [31, 22, 40], cospectrality (folklore) and equivalence with respect to first-order logic with counting quantifiers [12, 20, 11, 13]. In order to study the expressiveness of such equivalence relations, it is crucial to know under which circumstances distinct graph classes yield distinct equivalence relations. Towards this question one considers the closure of a graph class under homomorphism indistinguishability. Let $\mathcal{F}$ be a graph class. Two graphs G, H are *homomorphism indistinguishable over* $\mathcal{F}$, if for all $F \in \mathcal{F}$ the number of homomorphisms from F to H equals the number of homomorphisms from F to G. The graph class $\mathcal{F}$ is *homomorphism distinguishing closed*, if for every graph $F \notin \mathcal{F}$ there exists two graphs G, H, that are homomorphism indistinguishable over $\mathcal{F}$ but that do not have the same number of homomorphisms from F. It has been conjectured by Roberson [39], that all graph classes that are closed under taking minors and disjoint unions are homomorphism distinguishing closed. So far the list of graph classes for which the conjecture is confirmed is short: the class of all planar graphs [39], graph classes that are essentially finite [42], the classes of all graphs of treewidth at most $k-1$ [34] and the classes of all graphs of treedepth at most q [13]. The latter two results rely on characterisations of the graph classes in terms of non-monotone cops-and-robber games. We study *bounded depth treewidth*, which bounds both the width and the depth simultaneously. We give a game characterisation that does not rely on monotonicity, and as a consequence we obtain that the classes of all graphs of bounded depth treewidth are also homomorphism distinguishing closed.

Our Contribution. We show the following (cf. Theorem 27).

Fix integers $k, q \geq 1$. For every graph G the following are equivalent.

- *G has a tree decomposition of width at most $k-1$ and depth at most q.*
- *k cops have a monotone winning strategy in the cops and robber game on G with at most q placements.*
- *k cops have a winning strategy in the cops and robber game on G with at most q placements.*

The equivalence between the last two statements gives a positive answer to an open question from [13]. Our proof of monotonicity gives both a proof of monotonicity for the classical cops and robber game characterising treewidth as well as for the game characterising treedepth as special cases, by removing the bound on the number of rounds or respectively the number of cops. As a corollary, we obtain the following (cf. Theorem 18).

Let $k, q \geq 0$ be integers. The class of all graphs having a tree decomposition of width at most $k-1$ and depth at most q is homomorphism distinguishing closed.

Proof Techniques. In contrast to the proof of monotonicity of the classic cops and robber game [43], our proof does not use a dual concept such as brambles. Instead, we modify a (possibly non-monotone) winning strategy, turning it first into what we call a *pre-tree decomposition*, and then cleaning it up while keeping track of width and depth, thus finally transforming the pre-tree decomposition into a monotone winning strategy. Our concept of pre-tree decomposition is inspired by decompositions of matroids and it is based on ideas from [3, 7]. Our cleaning-up technique is similar to the proof of monotonicity of the game for b-branching treewidth [32]. However, the cleaning-up technique in [32] loses track of the number of cop movements, as local modifications may have non-local effects that are not controlled. We need to keep track in order to control the depth.

This poses a major challenge which we resolve in our proof by a fine grained cleaning-up technique in our pre-tree decomposition based on a careful decision of which vertices to "push up and through the tree" and which to "push down". The vertices "pushed up" may have an effect on the part of the pre-tree decomposition that was processed in previous steps, which we manage to control by a vertex exchange argument. Additionally we keep track of how the first modification at some node in the pre-tree decomposition relates back to the original strategy. We believe that our techniques will also help in future research.

Our proof provides an independent proof of monotonicity of the classic game characterising treewidth as a special case, namely when q is greater than or equal to the number of vertices of the graph. Our proof strategy is entirely different of the original proof of [43], as it does not use an equivalence via a dual object such as brambles. Instead, we provide a more direct transformation of a (possibly non-monotone) winning strategy.

Further Related Research. Search games are used to model a variety of real-world problems such as searching a lost person in a system of caves [36], clearing contaminated tunnels [28], searching environments in robotics [24], and modelling bugs in distributed environments [17], cf. [16] for a survey.

There is a fine line between games that are monotone and those that are not. For example, the marshalls and robber game played on a hypergraph is a natural generalisation of the cops and robber game, it is related to hypertreewidth, but it is not monotone [2]. However, the monotone and the non-monotone variants are strongly related [5] to each other. In a directed graph setting the games are also not monotone [27].

Structure of the Paper. In Section 2 we fix our notation and we define tree decompositions of bounded depth and width. Section 3 introduces pre-tree decomposition, relevant properties, and establishes a relation to tree decompositions. The game is introduced in Section 4, and in Section 5 we give the main construction, showing how to make a strategy tree exact while maintaining the bounds on width and depth. The insights given by our answer to the open question in the area of homomorphism counts are briefly discussed in Section 6. Due to space restrictions all proofs are deferred to the appendix.

2 Preliminaries

Sets and Partitions. Let A be a finite set. We write 2^A to denote the power-set of A and, for $k \in \mathbb{N}$, $\binom{A}{\leq k}$ to denote all subsets of A of size $\leq k$. $\mathrm{Part}(A)$ is the set of all *ordered partitions* of A, where we allow partitions to contain multiple (but finitely many) copies of the empty set. Let $\pi = \{X_1, \dots, X_d\} \in \mathrm{Part}(A)$ and $F \subseteq A$. For $i \in [d]$, the partition

$$i_{X_i \to F} \coloneqq \{X_1 \setminus F, \dots, X_{i-1} \setminus F, X_i \cup F, X_{i+1} \setminus F, \dots, X_d \setminus F\},$$

is called the *F-extension in X_i of π*. A function $w\colon \mathrm{Part}(A) \to \mathbb{N}$ is *submodular* if, for all $\pi, \pi' \in \mathrm{Part}(A)$, for all sets $X \in \pi$ and $Y \in \pi'$ with $X \cup Y \neq A$, it holds that

$$w(\pi) + w(\pi') \geq w(\pi_{X \to \overline{Y}}) + w(\pi'_{Y \to \overline{X}}).$$

Let $f\colon A \to B$ be a function and $C \subseteq A$. By $f|_C$ we denote the restriction of f to C, i. e. $f|_C\colon C \to B$ and $f|_C(c) = f(c)$, for all $c \in C$.

Graphs. A graph G is a tuple $(V(G), E(G))$, where $V(G)$ is a finite set of vertices and $E(G) \subseteq \binom{V(G)}{\leq 2}$ is the set of edges. We usually write uv or vu to denote the edge $\{u, v\} \in E(G)$. We write $\overrightarrow{E(G)}$, when we orient the edges of G, that is $\overrightarrow{E(G)} \coloneqq \{(u,v),(v,u) \mid uv \in E(G)\}$, and call $(u,v) \in \overrightarrow{E(G)}$ an arc. If G is clear from the context we write V, E instead of $V(G), E(G)$. By G° we denote the graph obtained from G by adding all self-loops that are not present in G, that is $V(G^\circ) \coloneqq V(G)$ and $E(G^\circ) \coloneqq E(G) \cup \{vv \mid v \in V(G)\}$. For $U \subseteq V$ we write $G[U]$ to denote the subgraph of G induced by U. For $v \in V$ we write $E_G(v) \coloneqq \{uv \mid uv \in E(G)\}$ for the edges incident to v and $N_G(v) \coloneqq \{u \mid uv \in E(G)\}$ for its neighbours.

A *tree* is a graph where any two vertices are connected by exactly one path. A *rooted tree* (T, r) is a tree T together with some designated vertex $r \in V(T)$, the *root* of T. At times, the following alternative definition is more convenient. We can view a rooted tree (T, r) as a pair $(V(T), \preceq)$, where $\preceq$ is a partial order on $V(T)$ and for every $v \in V(T)$ the elements of the set $\{u \in V(T) \mid u \preceq v\}$ are pairwise comparable: The minimal element of $\preceq$ is precisely the root of T, and we let $v \preceq w$ if v is on the unique path from r to w. Let $t, t' \in V(T)$, we call $t^* \in V(T)$ the *greatest common ancestor* if $t^* \preceq t, t'$ but for all $t'' \in V(T)$ with $t^* \prec t''$ either $t'' \not\preceq t$ or $t'' \not\preceq t'$. By $L(T)$ we denote the set of all *leaves* of T, that is $L(T) \coloneqq \{\ell \in V(T) \mid \text{ there is no } s \in V(T) \text{ such that } \ell \prec s\}$ is the set of all maximal elements of $\preceq$. All vertices that are not leafs are called *inner vertices.*

▶ **Definition 1.** *Let G be a graph, let (T, r) be a rooted tree and let $\beta\colon V(T) \to 2^{V(G)}$ be a function from the nodes of T to sets of vertices of G. We call (T, r, β) a* tree decomposition *of G, if*

(T1) *$\bigcup_{t \in V(T)} G[\beta(t)] = G$, and*

(T2) *for every vertex $v \in G$, the graph $T_v \coloneqq T[\{t \in V(T) \mid v \in \beta(t)\}]$ is connected.*

The sets $\beta(t)$ are called the bags *of this tree decomposition.*

The *width* of a tree decomposition (T, r, β) is $\mathrm{wd}(T, r, \beta) \coloneqq \max_{t \in V(T)} |\beta(t)| - 1$, the *depth* is $\mathrm{dp}(T, r, \beta) \coloneqq \max_{\ell \in L(T)} |\bigcup_{t \preceq \ell} \beta(t)|$. The *treewidth* of a graph G is the minimum width of any tree decomposition of G, the *treedepth* of a graph G is the minimum depth of any tree decomposition (see [13]). For $k, q \geq 1$ we define the class $\mathcal{T}_q^k$ to be all graphs that have a tree decomposition (T, r, β) with $\mathrm{wd}(T, r, \beta) \leq k - 1$ and $\mathrm{dp}(T, r, \beta) \leq q$. The following lemma is a well known consequence from (T2).

▶ **Lemma 2.** *Let G be a graph and $U \subseteq V(G)$ connected in G. Let (T, r, β) be a tree decomposition of G, then $T_U \coloneqq T[\{t \in V(T) \mid U \cap \beta(t) \neq \emptyset\}]$ is connected.*

3 Pre-Tree Decomposition, Exactness and Submodularity

Here we consider a definition of tree decompositions that is inspired by matroid tree decompositions [23]. We relax this definition into what we call a pre-tree decomposition.

▶ **Definition 3.** *Let $G = (V(G), E(G))$ be a graph. Let $X \subseteq E(G)$. We define the* boundary *of a set of edges $\delta(X) \coloneqq \{v \in V(G) \mid \exists e \in X, e' \in E(G) \setminus X, v \in e \cap e'\}$. Let π be a partition of $E(G)$. We define the* boundary *of a partition*

$$\delta(\pi) \coloneqq \bigcup_{X \in \pi} \delta(X).$$

A tuple (T, r, β, γ), where (T, r) is a (rooted) tree, $\beta \colon V(T) \to 2^{V(G)}$ and $\gamma \colon \overrightarrow{E(T)} \to 2^{E(G)}$, is a (rooted) pre-tree decomposition *if:*

(PT1) *$\beta(r) = \emptyset$ and for every connected component C of G, there is a child c of the root with $\gamma(r, c) = E(C)$.*

(PT2) *For every leaf $\ell \in L(T)$ with neighbour t, it holds that $|\gamma(t, \ell)| \leq 1$.*

(PT3) *For every $t \in V(T)$, the tuple π_t is a partition of $E(G)$ and $\delta(\pi_t) \subseteq \beta(t)$.*
For every internal node $t \in V(T) \setminus L(T)$, we define $\pi_t \coloneqq (\gamma(t, t_1), \ldots, \gamma(t, t_d))$, where $N(t) = \{t_1, \ldots, t_d\}$ is an arbitrary enumeration of the neighbours of t. For a leaf $\ell \in L(T)$ with parent p we define $\pi_\ell \coloneqq (\gamma(\ell, p), \overline{\gamma(\ell, p)})$.

(PT4) *For every edge $st \in E(T)$, it holds that $\gamma(s, t) \cap \gamma(t, s) = \emptyset$.*

We call an edge $st \in E(T)$ exact *if $\gamma(s, t) \cup \gamma(t, s) = E(G)$, we call (T, r, γ, β)* exact*, if every edge is exact and $\beta(t) = \delta(\pi_t)$, for all $t \in V(T)$. We call $\beta(t)$ the* bag *at node t and $\gamma(s, t)$ the* cone *at arc (s, t).*

The reader may note that the boundary of a set of edges is symmetric, that is for all $X \subseteq E(G)$ it holds that $\delta(X) = \delta(E(G) \setminus X)$. Furthermore it holds that $v \in \delta(X)$ if and only if $\emptyset \neq E_G(v) \cap X \neq E_G(v)$, for all $v \in V(G)$. The function γ describes a partition of the edges of the graph at every inner node, whereas the function β gives a vertex separator for this partition. This separator may contain more vertices than necessary at a certain node, which is needed to define the depth of a pre-tree decomposition, as seen below. For some edge $st \in E(T)$ with $s \prec t$, we can view $\gamma(s, t)$ as the set of edges that need to be decomposed in the subtree below and $\gamma(t, s)$ as the set of edges that is for sure decomposed somewhere else within the tree. With this point of view the axioms correspond to the following ideas:

(PT1) We start by separating the different connected components of the graph and assign one distinct subtree to decompose each component. This way we also ensure that all of the graph is decomposed in some subtree.

(PT2) We want to decompose the graph into single edges. We do allow empty leafs and even empty subtrees, to ease our cleaning up procedure in the following sections. The reader may recall that the root of a rooted tree is never a leaf, by definition.

(PT3) At every node of the tree we make sure that all edges of the original graph are accounted for and that β is indeed a separator for the given partition.

(PT4) If a parent node assigns an edge to the set that still has to be decomposed, the child node can not assign this edge to the set that is already decomposed somewhere else. But the other direction is possible, if the parent node assigns an edge to the set that is decomposed somewhere else, the child can still assign it to one of its subtrees. If the latter is also not the case the edge is exact.

Similar to the definition of width and depth for tree decompositions we define the width and depth of a pre-tree decomposition. We slightly adapt the definition of depth as (T2) does not hold in pre-tree decompositions.

▶ **Definition 4.** *The* width *of a partition π of the edges of a graph is*

$$\mathrm{wd}(\pi) := |\delta(\pi)|.$$

The width *of a pre-tree decomposition is*

$$\mathrm{wd}(T, r, \beta, \gamma) := \max_{t \in V(T)} |\beta(t)| - 1.$$

The depth *of a rooted pre-tree decomposition is*

$$\mathrm{dp}(T, r, \beta, \gamma) := \max_{t \in V(T)} \sum_{r \prec s \preceq t} |\beta(s) \setminus \beta(p_s)|,$$

where p_s is the parent of s.

The reader may note that the width of a pre-tree decomposition only gets smaller if one sets $\beta(t) := \delta(\pi_t)$, for all nodes $t \in V(T)$, but the depth can get larger. We show that the width of a partition of the edges as defined above is submodular. We need this property to show that our main construction does not enlarge the width of the pre-tree decomposition.

▶ **Lemma 5** ([7]). *For every graph G,* wd *is submodular.*

We continue this section with some lemmas, that help us to get comfortable with the definition of a pre-tree decomposition and are useful to prove that our cleaning up procedure in the following sections is correct. We start with a lemma about the cones along a path of exact edges. It is a direct consequence of exactness and the fact that the cones incident to a node form a partition of the edges.

▶ **Lemma 6.** *Let (T, r, β, γ) be a pre-tree decomposition of a graph G. Let $P = t_1, \ldots, t_\ell$ be a path in T, such that every edge $t_i t_{i+1}$, for $i \in [\ell - 1]$, is exact. Then it holds that $\gamma(t_1, t_2) \supseteq \gamma(t_2, t_3) \supseteq \ldots \supseteq \gamma(t_{\ell-1}, t_\ell)$.*

The following lemma shows, that (PT3) spreads over exact edges, that is any subtree of T that only contains exact edges induces a partition of the edges of the original graph. It is again a direct consequence of exactness and the partitions at the nodes together with the previous lemma.

▶ **Lemma 7.** *Let (T, r, β, γ) be a pre-tree decomposition of a graph G and let (T', r') be a subtree of (T, r), where r' is the minimal node of T' with respect to $\preceq$, such that all edges of T' are exact. We pick arbitrary enumerations of $N_T(V(T')) := \{t_1, \ldots, t_a\}$ and of $L(T) \cap L(T') := \{\ell_1, \ldots, \ell_b\}$. We define $U := \{t_1, \ldots, t_a, \ell_1, \ldots, \ell_b\}$ and define $s \colon U \to V(T')$ to be the natural mapping to the corresponding neighbour in $V(T')$. Then $(\gamma(s(t_1), t_1), \ldots \gamma(s(t_a), t_a), \gamma(s(\ell_1), \ell_1), \ldots, \gamma(s(\ell_1), \ell_1)))$ is an ordered partition of $E(G)$.*

The next lemma is needed to prove how one can translate exact pre-tree decompositions into tree-decompositions. Furthermore it will help us bound the depth within our cleaning up procedure in the following sections. The Lemma follows from the combination of Lemma 6 with the fact that the boundary of the partition is the union of the boundaries of the cones.

▶ **Lemma 8.** *Let (T, r, β, γ) be a pre-tree decomposition of a graph G and let (T', r') be a subtree of (T, r), where r' is the minimal node of T' with respect to $\preceq$, such that all edges of T' are exact. It holds that the induced subgraph $T'_v := T[t \in V(T') \mid v \in \delta(\pi_t)]$, for every vertex $v \in V(G)$, is connected. In particular, if $r = r'$, for every $t \in V(T')$ it holds that*

$$\sum_{\substack{s \preceq t \\ s \neq r}} |\delta(\pi_s) \setminus \delta(\pi_{p_s})| = |\bigcup_{s \preceq t} \delta(\pi_s)|,$$

where p_s is the parent of s.

We conclude this section with a lemma that shows that a pre-tree decomposition of a graph G is indeed a relaxation of a tree-decomposition of G. If every edge is exact and all bags are exactly the boundary of the partition then we can construct a tree decomposition. We need to start with a pre-tree decomposition of the graph G° with all self-loops added to ensure that every non-isolated vertex does appear in some bag and that the components corresponding to isolated vertices are covered by the pre-tree decomposition. If we drop the cones from the tuple we get a tree decomposition by Lemmas 7 and 8. On the other hand we can transform a tree-decomposition into a pre-tree decomposition, by copying the tree-decomposition of each connected component of G and adding leaves that correspond to the edges of G°.

▶ **Lemma 9.** *Let $k, q \geq 1$. Let $G = (V, E)$ be a graph. Any tree-decomposition of G of width $\leq k-1$ and depth $\leq q$ gives rise to an exact pre-tree decomposition of G° of width $\leq k-1$ and depth $\leq q$ and vice versa.*

Proof. Let (T, r, β, γ) be an exact pre-tree decomposition of G° of width $\leq k-1$ and depth $\leq q$. We define $\beta' \colon V(T) \to 2^{V(G)}$ as follows

$$\beta'(t) := \begin{cases} \{v\} & \text{if } t \in L(T) \text{ and } r \text{ is parent of } t \text{ and } \gamma(r, t) = \{vv\}, \\ \beta(t) & \text{otherwise.} \end{cases}$$

▷ Claim 10. (T, β') is a tree-decomposition of width $\leq k-1$ and depth $\leq q$.

Proof. From (PT1), (PT2) and Lemma 7 applied to the complete tree (T, r) we get that for every edge $uv \in E(G^\circ)$ there is some leaf ℓ with parent p and $\gamma(p, \ell) = \{uv\}$. Thus if $u = v$, then $\beta'(\ell) = \{v\}$ and thus $vv \in E(G[\beta'(\ell)])$. Otherwise it holds that $uu, vv \in E(G^\circ) \setminus \{uv\}$ and thus $u, v \in \beta'(\ell)$ and $uv \in E(G[\beta'(\ell)])$. All in all we get that (T1) holds.

By Lemma 8 applied to the complete tree (T, r) we know that all T_v are connected. Therefore (T2) also holds and (T, β') is a tree-decomposition.

The width and depth are obvious as $k, q \geq 1$. ◁

Now let (T, r, β) be a tree-decomposition of G of width $\leq k-1$ and depth $\leq q$. W.l.o.g. β is *tight*, that is for all $t \in V(T)$ and $v \in \beta(t)$, that (T, r, β'), where $\beta'(t) := \beta(t) \setminus \{v\}$ and $\beta'(s) = \beta(s)$, for all $s \in V(T) \setminus \{t\}$, is not a tree-decomposition of G. We construct a new tree T' with root r' and functions $\beta' \colon V(T') \to 2^{V(G)}$, $\gamma \colon \overrightarrow{E(T')} \to 2^{E(G^\circ)}$ and $f \colon V(T') \setminus (L(T') \cup \{r'\}) \to V(T)$ as follows. Let C be a connected component of G and let $V_C := \{t \in V(T) \mid V(C) \cap \beta(t) \neq \emptyset\}$. By Lemma 2 V_C is connected. If C contains only an isolated vertex v, then $V_C = \{t\}$, for some $t \in V(T)$. We add a new node t_v to T' and connect it to the root. We set $\beta'(t_v) = \emptyset$, $\gamma(r', t_v) = \{vv\}$ and $\gamma(t_v, r') = E(G^\circ) \setminus \{vv\}$. Otherwise let T_C be a copy of the subtree induced by V_C with root r_C and vertices V_C^* and $f|_{V_C^*} \colon V_C^* \to V_C$ the natural bijection between the copies and their originals. We attach r_C to the root r'. For every $v \in V(C)$, there is some $t_v \in V_C$ such that $v \in \beta(t_v)$, as C is not an isolated vertex. We add a new leaf t'_v that we attach to $f|_{V(T_C)}^{-1}(t_v)$ and set $\beta'(t'_v) = \{v\}$, $\gamma(f|_{V_C^*}^{-1}(t_v), t'_v) = \{vv\}$ and $\gamma(t'_v, f|_{V_C^*}^{-1}(t_v)) = E(G^\circ) \setminus \{vv\}$. For every $e \in E_G(C)$ there is some $t_e \in V_C$ such that $e \subseteq \beta(t_e)$. We add a new leaf t'_e that we attach to $f|_{V_C^*}^{-1}(t_e)$ and set $\beta'(t'_e) = e$, $\gamma(f|_{V_C^*}^{-1}(t_e), t'_e) = \{e\}$ and $\gamma(t'_e, f|_{V_C^*}^{-1}(t_e)) = E(G^\circ) \setminus \{e\}$. For every node $t \in V_C^*$ with parent p we add all edges $e \in E_G(C)$, where t'_e is a descendant of t, and all self-loops $vv \in E_{G^\circ}(C)$, where t'_v is a descendant of t, to $\gamma(p, t)$. Furthermore we set $\gamma(t, p) := E(G^\circ) \setminus \gamma(p, t)$ and $\beta'(t) := \delta(\pi_t) \subseteq \beta(f(t)) \cap V(C)$. By tightness of β there is some

$v \in \beta(f(\ell))$ such that $T_v = \{f(\ell)\}$, for every $\ell \in L(T_C)$, thus no leaf of T_C is a leaf in T', thus (T', r', β', γ) satisfies (PT2). (PT1), (PT3) and (PT4) hold by construction. Furthermore every edge is exact by construction. Thus we get that (T', r', β', γ) is an exact pre-tree decomposition of G°.

The width is obvious as every bag in β' is a subset of some bag in β. To see that the depth bound also holds we observe two things. For every leaf $\ell \in L(T')$ with parent p we get that $\beta'(\ell) \setminus \beta'(p) = \emptyset$. For every inner node $t \in V(T') \setminus L(T')$ with parent p we get that $\beta'(t) \setminus \beta'(p) \subseteq \beta(f(t))$ and, if $p \neq r'$, $\beta'(t) \setminus \beta'(p) \subseteq \beta(f(t)) \setminus \beta(f(p))$, by the tightness of β. ◀

4 The Game

In the cops and robber game on a graph G, the cops occupy sets X of at most k vertices of G, and the robber moves on edges of G. In order to make the rules precise, we need *edge components* of G that arise when the cops are blocking a set X.

▶ **Definition 11.** *Let $G = (V, E)$ be a graph and $X \subseteq V$. We let* the edge component graph *of G with respect to X be the graph G^X obtained as the disjoint union of the following graphs. (In order to make all graphs disjoint we introduce copies of vertices where needed.)*

- *For every $uv \in E(G[X])$, the graph $G_{uv} := (\{u, v\}, \{uv\})$, and*
- *for every connected component C of $G \setminus X$, the graph G_C, with $V(G_C) := V(C) \cup N_G(V(C))$ and $E(G_C) := E(C) \cup E(V(C), X)$, where $E(V(C), X)$ is the set of edges of G incident to both a vertex of C and a vertex in X.*

The reader may note that G^X may contain multiple copies of the vertices in X, but exactly one copy of each edge in G.

▶ **Observation 12.** *There is a natural bijection $\Psi \colon E(G^X) \to E(G)$ between the edges of G^X and the edges of G.*

▶ **Definition 13** (q-rounds k-cops-and-robber game)**.** *Let G be a graph and let $k, q \geq 1$. The* q-rounds k-cops-and-robber game $\mathrm{CR}_q^k(G)$ *is defined as follows:*

If G does not contain any edges the cop player wins immediately.

- *The cop positions are sets $X \in V(G)^{\leq k}$.*
- *The robber position is an edge $uv \in E(G)$.*
- *The initial position (X_0, u_0v_0) of the game is $X_0 = \emptyset$ and $u_0v_0 \in E(G)$, thus the game starts with no cops positioned on G and the robber on an arbitrary edge in a connected component of G of his choice.*
- *For $X \subseteq V(G)$ and $uv \in E(G)$, we write $\mathrm{esc}(X, uv) := \Psi(E(C))$ for the component C of the graph G^X, such that $uv \in E(C)$. We call this component* the robber escape space*. Thus if the cops are at positions X and robber at an edge uv we write $(X, \mathrm{esc}(X, uv))$ for the position of the game.*
- *In round i the cop player can move from the set X_{i-1} to a set X_i, if $|X_i \setminus X_{i-1}| \leq 1$, that is the cop player can add at most one new vertex to his position.*
- *In round i the robber player can move along a path with no internal vertex in $X_{i-1} \cap X_i$. Thus the robber player can move to some edge u_iv_i, such that the edge $\Psi^{-1}(u_iv_i)$ is in a connected component of G^{X_i} that is contained in $\Psi^{-1}(\mathrm{esc}(u_{i-1}v_{i-1}, X_{i-1} \cap X_i))$ via a path $p = w_1, \dots, w_\ell$ where $\{w_1, w_2\} = \{u_{i-1}, v_{i-1}\}$ and $\{w_{\ell-1}, w_\ell\} = \{u_i, v_i\}$ and $\{w_2, \dots, w_{\ell-1}\} \cap X_i \cap X_{i-1} = \emptyset$.*
- *The cop-player wins in round i, if $\{u_i, v_i\} \subseteq X_i$, and we say the cop player* captures *the robber in round i. The robber-player wins if the cop player has not won in round q.*

We call the game monotone q-round k-cops-and-robber game*, if we further restrict the movement of the cop player such that always* $\operatorname{esc}(X_{i-1}, u_{i-1}v_{i-1}) \supseteq \operatorname{esc}(X_{i-1} \cap X_i, u_{i-1}v_{i-1})$ *and write* $\text{mon-CR}_q^k(G)$*.*

This notion of monotone is also known as robber-monotone. One can also define the notion of cop-monotone, that is during a single play, the cop player may never revisit a vertex they have previously left. This is the stronger notion of monotone as a cop-monotone play is also robber-monotone. Our construction goes through unchanged with the notion of cop-monotone as the strategy that is derived from the final decomposition is cop-monotone. The game played on the graph G° corresponds to a game on G, where the robber can hide both inside a vertex or an edge. It is easy to see that this does not benefit the robber player, that is he wins the game $\text{CR}_q^k(G)$ if and only if he wins the game $\text{CR}_q^k(G^\circ)$, as the components that are reachable by the robber player are essentially the same. In [13], the authors introduce a cops-and-robber game, where the robber player can only hide in the vertices. Again this does not pose a restriction for the robber player with the same argument as above. There is a tight connection between the cops and robber game defined above and tree decompositions of graphs.

▶ **Lemma 14** ([13]). *Let G be a graph and $k, q \in \mathbb{N}$. The cop player wins* $\text{mon-CR}_q^k(G)$ *if and only if $G \in \mathcal{T}_q^k$.*

Towards strengthening the above connection to also include the non-monotone game we first introduce how to construct a pre-tree decomposition from a winning strategy of the cop player.

▶ **Definition 15** (strategy tree). *Let G be a graph without isolated vertices and let $k, q \in \mathbb{N}$. Let $\sigma \colon V(G)^{\leq k} \times E(G) \to V(G)^{\leq k}$ a cop strategy such that that for all $X \in V(G)^{\leq k}$, for all $uv \in E(G)$ and for all $u'v' \in \operatorname{esc}(X, uv)$ we have that $\sigma(X, uv) = \sigma(X, u'v')$. We write $\sigma(X, \operatorname{esc}(X, uv))$ instead of $\sigma(X, uv)$.*

The strategy tree *of σ is a pre-tree decomposition (T, r, β, γ), inductively defined as follows:*

- *$\beta(r) = \emptyset$,*
- *for every connected component C of G, there is a child c of the root r and $\gamma(r, c) = E(C)$,*
- *for every node $t \in V(T) \setminus \{r\}$ with parent $s \in V(T)$,*
  - *if the robber player is caught, we set $\beta(t) = e$, where $\gamma(s, t) = \{e\}$,*
  - *else $\beta(t) = \sigma(\beta(s), \gamma(s, t))$ and*
  - *for every connected component C of $G^{\beta(t)}$, that has a non-empty intersection with $\Psi^{-1}(\gamma(s, t))$, there is a child c of t and $\gamma(t, c) = \Psi(E(C))$,*
  - *$\gamma(t, s) := E(G) \setminus \bigcup_{c \text{ child of } t} \gamma(t, c)$, if $t \notin L(T)$, and*
  - *$\gamma(t, s) := E(G) \setminus \gamma(s, t)$, if $t \in L(T)$.*

We call $t \in V(T)$ a branching node *if the cop player placed a new cop incident to the robber escape space.*

Observe that if $t \in V(T)$ is a leaf, then the robber is captured and the depth of (T, r, β, γ) is $\leq q$ if and only if σ is a winning strategy in $\text{CR}_q^k(G)$.

Note that w.l.o.g. every child of the root is a branching node, as the cop player w.l.o.g. only plays positions that are inside the component the robber player chose in the first round. If the game is played on G°, then every branching node that does not correspond to the placement of a cop onto an isolated vertex has more than one child. We observe that the monotone moves of the cop player correspond to the exact edges in the strategy tree by construction.

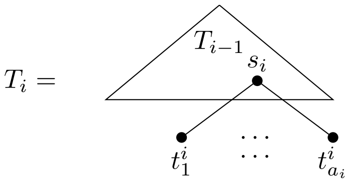


Figure 1 The subtree T_i appearing in the construction.

▶ **Lemma 16.** *For edge $st \in E(T)$, where $s \prec t$ it holds that the move $\sigma(\beta(s), \gamma(s,t))$ is monotone if and only if st is exact.*

The following lemma about the self-loops of the graph G° is key to prove the construction in the next section does not enlarge the depth of the pre-tree decomposition. To prove this one finds the node where the vertex incident to the self-loop was introduced into the bag. At this node the self-loop either was not in the robber escape space in the first place or gets removed from the robber escape space. Then as long as the cops occupy the vertex incident to the self-loop, the self-loop is not reachable by the robber.

▶ **Lemma 17.** *Considering the game on G° and some $s \in V(T) \setminus L(T)$. For all self-loops vv incident to $\beta(s)$ it holds that either $vv \in \gamma(s, p_s)$ or there is a child c of s such that $\gamma(s,c) = \{vv\}$. Furthermore s has a child c with $\gamma(s,c) = \{vv\}$, for some non-isolated vertex v if and only if s is a branching node and $v \in \beta(s) \setminus \beta(p_s)$.*

5 Making a Strategy Tree Exact

Our goal is to prove the following theorem.

▶ **Theorem 18.** *Let $G = (V(G), E(G))$ be a graph, let $k, q \geq 1$ and let (T, r, β, γ) be a strategy tree for some cop strategy $\sigma\colon V(G^\circ)^k \times E(G^\circ) \to V(G^\circ)^k$. If σ is a winning strategy in $\mathrm{CR}_q^k(G^\circ)$, then there is a tree decomposition of G with width $\leq k-1$ and depth $\leq q$.*

To prove this we construct an exact pre-tree decomposition of G° from the strategy tree, starting at the root r and traversing the tree nodes in a breadth-first-search. We then use Lemma 9 to get the desired tree decomposition. When we consider a node we change the pre-tree decomposition so that all incident edges are exact afterwards. Note that by the choice of the traversal we only need to consider outgoing edges.

The Construction. Let (T, r, β, γ) be the pre-tree decomposition of G° from a winning strategy as in Theorem 18. Let $s_1, \dots, s_{n_T}$ be an order of the nodes of T in bfs where $s_1 = r$. Let $\beta_0 := \beta$ and $\gamma_0 := \gamma$. We construct a sequence $(T, r, \beta_0, \gamma_0), \dots, (T, r, \beta_{n_T}, \gamma_{n_T})$ of pre-tree decompositions, such that $(T, r, \beta_{n_T}, \gamma_{n_T})$ is exact. We say s_i is *considered in step i*. Let

$$T_i := T[\{s_1, \dots, s_i\} \cup N_T(\{s_1, \dots, s_i\})] = T[V(T_{i-1}) \cup N_T(s_i)].$$

See Figure 1 for an illustration of T_i. (It will become clear that this is the subtree of all nodes where the pre-tree decomposition is modified in or before step i. We also point out that edges from T_i to $T \setminus T_i$ may become non-exact during our modification process.)

If s_i is a leaf, there are no outgoing edges that are not exact, and we set $\beta_i := \beta_{i-1}$ and $\gamma_i := \gamma_{i-1}$. Otherwise let $t_1^i, \dots, t_{a_i}^i \in N_T(s_i)$ be all children of s_i.

- We pick pairwise disjoint $F^i_1, \dots, F^i_{a_i} \subseteq E(G^\circ)$, with

  $$F^i_j \subseteq \overline{\gamma_{i-1}(t^i_j, s_i)} \cap \overline{\gamma_{i-1}(s_i, t^i_j)},$$

  such that the partition π^* that results from taking the F^i_j-extensions in $\gamma_{i-1}(s_i, t^i_j)$ (in arbitrary order) has the minimum size boundary. If there are multiple optimal choices for $F^i_1, \dots, F^i_{a_i}$ we select the one that minimizes the size of $\bigcup_{j\in[a_i]} F^i_j$, if there are still several options we break ties arbitrarily.
- Let $F^i \coloneqq \bigcup_{j\in[a_i]} F^i_j$ and $R^i_j \coloneqq \left(\left(\overline{\gamma_{i-1}(t^i_j, s_i)} \cap \overline{\gamma_{i-1}(s_i, t^i_j)}\right) \cup F^i\right) \setminus F^i_j$. For every vertex $p \in V(T_i)$ with child c we set

  $$\gamma_i(p,c) \coloneqq \begin{cases} \left(\gamma_{i-1}(s_i, t^i_j) \setminus F^i\right) \cup F^i_j & \text{if } (p,c) = (s_i, t^i_j), \text{ for some } j \in [a_i], \\ \gamma_{i-1}(p,c) \setminus R^i_j & \text{if } p = t^i_j, \text{ for some } j \in [a_i], \\ \gamma_{i-1}(p,c) \cup F^i & \text{if } p \prec s_i, \\ \gamma_{i-1}(p,c) \setminus F^i & \text{otherwise,} \end{cases}$$

  and

  $$\gamma_i(c,p) \coloneqq \begin{cases} \gamma_{i-1}(c,p) \cup R^i_j & \text{if } (p,c) = (s_i, t^i_j), \text{ for some } j \in [a_i], \\ \gamma_{i-1}(c,p) & \text{if } p = t^i_j, \text{ for some } j \in [a_i], \\ \gamma_{i-1}(c,p) \setminus F^i & \text{if } p \prec s_i, \\ \gamma_{i-1}(c,p) \cup F^i & \text{otherwise,} \end{cases}$$

  and all other $uv \in \overrightarrow{E(T)}$ we set $\gamma_i(u,v) \coloneqq \gamma_{i-1}(u,v)$. Furthermore we set

  $$\beta_i(t) \coloneqq \begin{cases} \delta(\pi^i_t) & \text{if } t \in V(T_i), \\ \beta_{i-1}(t) & \text{otherwise.} \end{cases}$$

Intuitively in the construction above we pick F^i_j, that is the set of edges that we add to the cone at the arc pointing towards t^i_j, from the set of edges that are not covered by the cones along the arcs in such a way that the boundary at s_i is minimized. R^i_j then corresponds to the edges that we need to remove from all cones at arcs that point away from s_i at the child t^i_j and in turn add to the cone at the arc pointing towards s_i to make the edge exact. Then we push the change at s_i through T_{i-1}, that is for all edges in T_{i-1} we add F^i to the arc that points towards s_i and remove F^i from the arcs in the other direction. This push can also be interpreted in terms of extensions.

▶ **Lemma 19.** *Let $t \in V(T_i) \setminus \{s_i\}$ and t' is next node on the path from t to s_i in T. Then*

$$\pi^i_t = \left(\pi^{i-1}_t\right)_{\gamma_{i-1}(t,t') \to \overline{\gamma_i(t',t)}}$$

and if additionally $t \in V(T_{i-1})$ also

$$\pi^i_t = \left(\pi^{i-1}_t\right)_{\gamma_{i-1}(t,t') \to F^i}.$$

Observe furthermore that if $\beta_i(s_i) = \beta_{i-1}(s_i)$, then there are no changes to the bags at other nodes than the t^i_j by minimality of $|F^i|$, and if $\beta_i(s_i) \neq \beta_{i-1}(s_i)$, we have $|\beta_i(s_i)| < |\beta_{i-1}(s_i)|$ again by the minimality of the choice. For every $i \in [n_T]$, before step i we only remove edges from the cones pointing downwards from s_i, thus we obtain the following observation.

▶ **Lemma 20.** *Let $i, j \in [n_T]$ such that s_i is the parent of s_j. Then $\gamma_\alpha(s_i, s_j) \subseteq \gamma(s_i, s_j)$, for all $\alpha < i$.*

The Proof Idea. We prove Theorem 18 in three steps. First we prove that the construction indeed yields an exact pre-tree decomposition. Next we show that the width can be bounded as desired and lastly we prove that the construction yields the desired depth.

In step i, every edge incident to s_i is made exact and for every other edge in T_i we remove from one arc exactly what we add to the other arc. We get that our construction indeed yields an exact pre-tree decomposition.

▶ **Lemma 21.** *For all $i \in [n_T]$, $(T, r, \beta_i, \gamma_i)$ is a pre-tree decomposition. Furthermore all edges in $E(T_i)$ are exact.*

Hence, for $i = n_T$, we get that $(T, r, \beta_{n_T}, \gamma_{n_T})$ is an exact pre-tree decomposition. Note that it is possible that $\gamma_{n_T}(s, t)$ is empty for an arc $(s, t) \in \overrightarrow{E(T)}$. By Lemma 9 we obtain a tree decomposition, from this pre-tree decomposition. We show below that the width and depth are as stated in the theorem.

Our construction does not change the width of the decomposition. To prove this we observe that in step i the bound in s_i is minimal. We then push the change through the subtree T_i and find that if a change would increase the width, we could push this change back to the node s_i and find an even smaller bound there, which contradicts the minimality of our choice. This argument yields the following lemma.

▶ **Lemma 22.** $\mathrm{wd}(T, r, \beta_i, \gamma_i) \leq \mathrm{wd}(T, r, \beta, \gamma)$*, for all $i \in [n_T]$.*

Proof. $\overline{\gamma_i(t', t)}$ We prove the statement for all $0 \leq i \leq n_T$ by induction. As $(T, r, \beta_0, \gamma_0) = (T, r, \beta, \gamma)$, the statement clearly holds for $i = 0$. Next we show that $\mathrm{wd}(T, r, \beta_i, \gamma_i) \leq \mathrm{wd}(T, r, \beta_{i-1}, \gamma_{i-1})$, for all $i \in [n_T]$. Obviously $|\beta_i(t)| = |\beta_{i-1}(t)|$, for all $t \notin T_i$. Furthermore by construction $|\beta_i(s_i)| \leq |\beta_{i-1}(s_i)|$. Let $j \in [a_i]$, let $X := \gamma_i(s_i, t^i_j)$ and let $Y := \gamma_{i-1}(t^i_j, s_i)$. By Lemma 19 it holds that

$$|\beta_i(t^i_j)| = \mathrm{wd}\left(\left(\pi^{i-1}_{t^i_j}\right)_{Y \to \overline{X}}\right) \leq \mathrm{wd}(\pi^{i-1}_{t^i_j}) \leq |\beta_{i-1}(t^i_j)|$$

as otherwise by submodularity for the partitions $\pi^{i-1}_{t^i_j}$ and $\pi^i_{s_i}$, we get that

$$\mathrm{wd}\left(\left(\pi^i_{s_i}\right)_{X \to \overline{Y}}\right) < \mathrm{wd}(\pi^i_{s_i}),$$

which contradicts the minimality of the bound for $F^i_1, \ldots, F^i_{a_i}$.

Lastly assume there is a node t in $V(T_i) \setminus \{s_i, t^i_1, \ldots, t^i_{a_i}\}$ such that $|\beta_i(t)| > |\beta_{i-1}(t)|$. We assume t is of minimal distance to s_i with this property. Let $x_0 = t, x_1, \ldots, x_b = s_i$ be the path from t to s_i. By minimality of the distance we know that $|\beta_i(x_1)| \leq |\beta_{i-1}(x_1)|$. Additionally we know that all edges on the path from s_i to x_1 are exact in γ_i, as well as the edge $x_1 t$ in γ_{i-1}. Now let $Y := \gamma_{i-1}(t, x_1)$, $X_\alpha := \gamma_i(x_{\alpha+1}, x_\alpha)$ and $Z_\alpha := \gamma_i(x_\alpha, x_{\alpha+1})$, for all $0 \leq \alpha < b$. From Lemma 19 we get $\left(\pi^{i-1}_t\right)_{Y \to F^i} = \left(\pi^{i-1}_t\right)_{Y \to \overline{X_0}}$. Thus assuming that $\mathrm{wd}\left(\left(\pi^{i-1}_t\right)_{Y \to F^i}\right) = |\beta_i(t)| > |\beta_{i-1}(t)| = \mathrm{wd}(\pi^{i-1}_t)$ using submodularity we get that $\mathrm{wd}(\pi^i_{x_1}) > \mathrm{wd}\left(\left(\pi^i_{x_1}\right)_{X_0 \to \overline{Y}}\right)$. As the edge $x_1 t$ was exact at step $i - 1$, we know that

$$F' := \overline{Y} \setminus X_0 = F^i \setminus Y \subseteq F^i.$$

We now push this change back to s_i along the path $x_1, \ldots, x_b$ and we again find a contradiction to the minimality of the bound of $F^i_1, \ldots, F^i_{a_i}$. For this, let us assume we have pushed the change to x_α, that is we changed $\pi^i_{x_\alpha}$ to $\pi^*_{x_\alpha} = \left(\pi^i_{x_\alpha}\right)_{X_{\alpha-1} \to F'}$ and we know that $\mathrm{wd}(\pi^*_{x_\alpha}) < \mathrm{wd}(\pi^i_{x_\alpha})$. As $x_\alpha x_{\alpha+1}$ is exact in γ_i, we get that $\left(\pi^*_{x_\alpha}\right)_{(Z_\alpha \setminus F') \to \overline{X_\alpha}} = \pi^i_{x_\alpha}$. Let

$$\pi^*_{x_{\alpha+1}} := \left(\pi^i_{x_{\alpha+1}}\right)_{X_\alpha \to \overline{(Z_\alpha \setminus F')}} = \left(\pi^i_{x_{\alpha+1}}\right)_{X_\alpha \to F'},$$

then by submodularity $\mathrm{wd}(\pi^*_{x_{\alpha+1}}) < \mathrm{wd}(\pi^i_{x_{\alpha+1}})$. When we have pushed the change to $\alpha = b$, we find the desired contradiction. ◀

To prove that our construction does not increase the depth we show that in every step i the depth up to the nodes in T_i is bounded by the depth up to these nodes in the original tree. We prove this by induction on the number of steps. We recall that $V(T_i) = V(T_{i-1}) \cup \{t_1^i, \dots, t_{a_i}^i\}$ and that $s_i \in L(T_{i-1})$. For the nodes $t \in V(T_{i-1})$ we can directly build upon the induction hypothesis. But the nodes t_j^i, with $j \in [a_i]$, are added into the subtree. Here we need to compare directly to the original bags, as we can no longer use that in step $i-1$ the depth at these nodes is bounded by the depth in the original strategy tree. We can prove for these nodes that every vertex newly placed at one of these nodes in step i is also newly placed in the original strategy. Then we can show that the difference between depth at these nodes and their parent in step i can be bounded by the difference in the original strategy tree.

▶ **Lemma 23.** *Every $j \in [a_i]$ satisfies $\beta_i(t_j^i) \setminus \beta_i(s_i) \subseteq \beta(t_j^i) \setminus \beta(s_i)$.*

Proof. Let $v \in \beta_i(t_j^i) \setminus \beta_i(s_i)$. Since $v \in \beta_i(t_j^i)$ it holds that $E_{G^\circ}(v) \not\subseteq \gamma_i(t_j^i, s_i)$. As $v \notin \beta_i(s_i)$ we get that $v \notin \delta(\gamma_i(t_j^i, s_i))$ and thus $E_{G^\circ}(v) \cap \gamma_i(t_j^i, s_i) = \emptyset$. By construction we have that $\gamma_i(t_j^i, s_i) \supseteq \gamma_{i-1}(t_j^i, s_i) = \gamma(t_j^i, s_i)$, and thus $vv \notin \gamma(t_j^i, s_i)$. As $v \in \beta_i(t_j^i) = \delta(\pi_{t_j^i})$, there are two distinct children c_1, c_2 of t_j^i such that $v \in \delta(\gamma_i(t_j^i, c_\ell))$ and thus $E_{G^\circ}(v) \cap \gamma_i(t_j^i, c_\ell) \neq \emptyset$, for $\ell = 1, 2$. By construction we have $\gamma_i(t_j^i, c_\ell) \subseteq \gamma_{i-1}(t_j^i, c_\ell) = \gamma(t_j^i, c_\ell)$, for $\ell = 1, 2$. And thus $v \in \delta(\pi_{t_j^i}) \subseteq \beta(t_j^i)$. By Lemma 17 there thus is a child c of t_j^i such that $\gamma(t_j^i, c) = \{vv\}$ and, by Lemma 17, $v \in \beta(t_j^i) \setminus \beta(s_i)$. ◀

For the nodes $t \in V(T_{i-1})$ we show that the depth is not only bounded by the depth within the original pre-tree decomposition, but within the previous step. We do this by a vertex exchange argument, that is we track all vertices added or removed from any bag within T_{i-1}. For the vertices that are added to any bag in $V(T_{i-1})$ we get the following lemma.

▶ **Lemma 24.** *Let $i \in [n_T]$ and let $t \in V(T_{i-1})$. If $v \in \beta_i(t) \setminus \beta_{i-1}(t)$, then $v \in \beta_i(t^*)$, for all t^* on the path from t to s_i.*

Proof. Let $t^* \neq t$. Let t' be the next node on the path from t to s_i. Per definition it holds that $\gamma_i(t, t') = \gamma_{i-1}(t, t') \cup F^i$. As $\gamma_i(t, t')$ is the only set incident to t where edges are added in step i, we get that $v \in \delta(\gamma_i(t, t'))$. Combined with $v \notin \delta(\gamma_{i-1}(t, t'))$ we get that $E_{G^\circ}(v) \cap \gamma_{i-1}(t, t') = \emptyset$ and thus $\emptyset \neq E_{G^\circ}(v) \cap F^i \neq E_{G^\circ}(v)$. Now suppose that $v \notin \beta_i(t^*)$, and thus also $v \notin \delta(\gamma_i(t^*, p))$, where p is the next node on the path from t^* to t. As $\emptyset \neq E_{G^\circ}(v) \cap F^i \neq E_{G^\circ}(v)$ this implies that $E_{G^\circ}(v) \cap \gamma_i(t^*, p) = E_{G^\circ}(v) \cap (\gamma_{i-1}(t^*, p) \setminus F_i) = \emptyset$. We know from Lemma 21 that all edges in T_i are exact and thus that $\gamma_i(t', t) \subseteq \gamma_i(t^*, p)$ by Lemma 6. Thus we get that $E_{G^\circ}(v) \cap \gamma_i(t', t) = \emptyset$. This is a contradiction to the assumption that $v \in \delta(\gamma_i(t, t')) = \delta(\gamma_i(t', t))$ and thus $v \in \beta_i(t^*)$. ◀

The next lemma is used to show that a vertex that disappears from a Observe that if $\beta_i(s_i) = \beta_{i-1}(s_i)$, then there are no changes to the bags at other nodes than the t_j^i by minimality of $|F^i|$, and if $\beta_i(s_i) \neq \beta_{i-1}(s_i)$, we have $|\beta_i(s_i)| < |\beta_{i-1}(s_i)|$ again by the minimality of the choice.bag in $V(T_{i-1})$ at step i also disappears from the union of bags that determine the depth at that bag, especially if a vertex disappears from the bag at s_i, then it disappears from every bag in $V(T_i)$.

▶ **Lemma 25.** *Let $i \in [n_T]$ and let $t \in V(T_{i-1})$. If $v \in \beta_{i-1}(t) \setminus \beta_i(t)$, then $v \notin \beta_i(t^*)$, for all $t^* \in V(T_{i-1})$ such that t is contained in the path from t^* to s_i.*

Proof. We have $E_{G^\circ}(v) \cap F^i \neq \emptyset$.

Let $t = s_i$. Since $v \in \beta_{i-1}(s_i) = \delta(\pi^{i-1}_{s_i})$ it holds that $E_{G^\circ}(v) \not\subseteq \gamma_{i-1}(s_i, p_{s_i})$. As $v \notin \beta_i(s_i)$ we get that $v \notin \delta(\gamma_i(s_i, p_{s_i}))$ and thus $E_{G^\circ}(v) \cap \gamma_i(s_i, p_{s_i}) = E_{G^\circ}(v) \cap \gamma_{i-1}(s_i, p_{s_i}) \cap \overline{F^i} = \emptyset$. Now let $t^* \in V(T_{i-1})$ and t' be the next node on the path from t^* to s_i. Then by Lemma 21 we get that $\gamma_i(t^*, t') \supseteq \gamma_i(p_{s_i}, s_i) \supseteq E_{G^\circ}(v)$ and thus $v \notin \beta_i(t^*)$.

Otherwise let $t \neq s_i$ Let t' be the next node on the path from t to s_i. Since $v \in \delta(\pi^{i-1}_t)$, we get that $E_{G^\circ}(v) \not\subseteq \gamma_{i-1}(t, t')$. As $v \notin \delta(\gamma_i(t, t'))$ and $E_{G^\circ}(v) \cap F^i \neq \emptyset$ it follows that $E_{G^\circ}(v) \subseteq \gamma_i(t, t') = \gamma_{i-1}(t, t') \cup F^i$ and that $E_{G^\circ}(v) \cap \gamma_{i-1}(t', t) \subseteq E_{G^\circ}(v) \cap F^i$. Assume there is some $t^* \in V(T_{i-1})$ such that $v \in \beta_i(t^*)$. We observe that due to Lemma 21 and because all edges incident to v are contained in $\gamma_i(t, t')$, we get that t is not contained in the path from t^* to s_i. ◀

This induction then yields the following lemma.

▶ **Lemma 26.** *For all $i \in [n_T]$ and all $t \in V(T_i)$, it holds that*

$$\sum_{r \prec s \preceq t} |\beta_i(s) \setminus \beta_i(p_s)| \leq \sum_{r \prec s \preceq t} |\beta(s) \setminus \beta(p_s)|.$$

Proof. Let $\ell \in [n_T]$. As by construction $\beta_\ell(t) = \delta(\pi^\ell_t)$, for all $t \in V(T_\ell)$, we get from Lemma 8 and Lemma 21 that $|\bigcup_{s \preceq t} \beta_\ell(s)| = \sum_{r \prec s \preceq t} |\beta_\ell(s) \setminus \beta_\ell(p_s)|$. Thus it suffices to show that $|\bigcup_{s \preceq t} \beta_i(s)| \leq \sum_{r \prec s \preceq t} |\beta(s) \setminus \beta(p_s)|$.

We prove the statement by induction on the steps i. Recall that $(T, r, \beta_0, \gamma_0) = (T, r, \beta, \gamma)$, thus the statement holds for $i = 0$. Now assume the statement holds for $i - 1$, thus for all $t \in V(T_{i-1})$ it holds that $|\bigcup_{s \preceq t} \beta_{i-1}(s)| \leq \sum_{r \prec s \preceq t} |\beta(s) \setminus \beta(p_s)|$. We recall that $V(T_i) = V(T_{i-1}) \cup \{t^i_1, \ldots, t^i_{a_i}\}$ and that $s_i \in L(T_{i-1})$. We consider all vertices that appear at a bag at any node in T_i due to the changes in step i.

Let $t \in V(T_{i-1})$. Let $U \coloneqq \left(\bigcup_{s \preceq t} \beta_i(s)\right) \setminus \left(\bigcup_{s \preceq t} \beta_{i-1}(s)\right)$. Let t^* be the greatest common ancestor of t and s_i. As t^* is on every path from some node $s \preceq t$ to s_i, from Lemma 24 we know that $u \in \beta_i(t^*) \setminus \beta_{i-1}(t^*)$, for all $u \in U$. Let $W \coloneqq \beta_{i-1}(t^*) \setminus \beta_i(t^*)$. As by Lemma 22 $|\beta_i(t^*)| \leq |\beta_{i-1}(t^*)|$, we know that $|U| \leq |W|$. Applying Lemma 25 we get that $W \subseteq \left(\bigcup_{s \preceq t} \beta_{i-1}(s)\right) \setminus \left(\bigcup_{s \preceq t} \beta_i(s)\right)$ and using this *vertex exchange* we conclude that $\left|\bigcup_{s \preceq t} \beta_i(s)\right| \leq \left|\bigcup_{s \preceq t} \beta_{i-1}(s)\right|$.

Otherwise it holds that $t = t^i_j$ for some $j \in [a_i]$. By construction we can conclude that $\bigcup_{s \preceq t} \beta_i(s) = \bigcup_{s \preceq s_i} \beta_i(s) \cup \beta_i(t) \setminus \beta_i(s_i)$. We know that $|\bigcup_{s \preceq s_i} \beta_i(s)| \leq \sum_{r \prec s \preceq s_i} |\beta(s) \setminus \beta(p_s)|$ and by Lemma 23 we have $\beta_i(t) \setminus \beta_i(s_i) \subseteq \beta(t) \setminus \beta(s_i)$. Thus we can bound the union $|\bigcup_{s \preceq t} \beta_i(s)| \leq \sum_{r \prec s \preceq t} |\beta(s) \setminus \beta(p_s)|$. ◀

Summarising all results we get the following equivalences.

▶ **Theorem 27.** *Let $k, q \geq 1$ and G be a graph. The following are equivalent:*

(1) *G admits a tree decomposition of width at most $k - 1$ and depth at most q.*

(2) *G° admits a tree decomposition of width at most $k - 1$ and depth at most q.*

(3) *G° admits an exact pre-tree decomposition of width at most $k - 1$ and depth at most q.*

(4) *The cop player wins* mon-$\mathrm{CR}^k_q(G^\circ)$.

(5) *The cop player wins* $\mathrm{CR}^k_q(G^\circ)$.

(6) *The cop player wins* mon-$\mathrm{CR}^k_q(G)$.

(7) *The cop player wins* $\mathrm{CR}^k_q(G)$.

6 Excursion on Counting Homomorphisms

In this section we give an overview over the field of counting homomorphisms and the equivalence relations on graphs, that can be derived from these counts. We focus ourselves to the results and open questions regarding the homomorphism counts from graphs in the class $\mathcal{T}_q^k$, for fixed $k, q \geq 0$.

We recall the definition of homomorphism distinguishing closed from the introduction. In [13] the authors have reduced the question whether the class $\mathcal{T}_q^k$ is homomorphism distinguishing closed down to the question if monotonicity is a restriction for the cop player. The following lemma is thus implied in [13].

▶ **Lemma 28** ([13])**.** *Let $k, q \geq 1$. The graph class $\mathcal{C} := \{G \mid$ cop player wins $\mathrm{CR}_q^k(G)\}$ is homomorphism distinguishing closed.*

In this paper we show that the cop player wins $\mathrm{CR}_q^k(G)$ if and only if $G \in \mathcal{T}_q^k$, thus the we get the following.

▶ **Theorem 29.** *Let $k, q \geq 0$. The class $\mathcal{T}_q^k$ is homomorphism distinguishing closed.*

7 Conclusion

We gave a new characterisation of bounded depth treewidth by the cops and robber game with both a bound on the number of cops and on the number of rounds, where the cop player is allowed to make non-monotone moves. As a corollary we gave a positive answer to an open question on homomorphism counts. The core of our contribution is a proof of monotonicity of this game. For this proof we substantially reorganise a winning strategy. First we transform it into a pre-tree decomposition. Then we apply a breadth-first "cleaning up" procedure along the pre-tree decomposition (which may temporarily lose the property of representing a strategy), in order to achieve monotonicity while controlling the number of cop rounds simultaneously across all branches of the decomposition via a vertex exchange argument. As an interesting observation we obtain that cop moves onto some vertex not incident to the robber escape space, i. e. to positions that are not part of the boundary, can be ignored and the depth of the exact pre-tree decomposition is the number of cops placed into the robber escape space. To see that consider the proof of Lemma 23 where we compute how much larger the depth at some node t_j^i at step i is than at the considered node s_i. The depth increases only if the node t_j^i is branching by Lemma 17 as t_j^i has a child where the cone contains only a self-loop and hence this is a move into the robber escape space.

▶ **Corollary 30.** $\mathrm{dp}(T, r, \beta_{n_T}, \gamma_{n_T}) \leq \max_{\ell \in L(T)} |\{t \in V(T) \mid t \preceq \ell$ *and t is branching*$\}|$.

We note that we use a slightly different notion of branching node as [32], as we use a different game characterisation. On a graph without isolated vertices our branching nodes are branching nodes in the sense of [32], but not the other way around, as in the non-deterministic cops and robber game, the cop player can chose if he whats to branch in the strategy tree, that is if he wants to know the position of the robber or not.

In the future, it would be interesting to know if it is possible to give a proof that entirely argues with game strategies (not requiring pre-tree decompositions), and we leave this open. We also leave open whether a dual object similar to brambles can be defined for bounded depth treewidth. Finally, given a winning strategy for k cops with q rounds, it would be interesting to know if it is possible to bound the number of cops necessary for winning with only $q - 1$ rounds in terms of k and q, given that the cop player still can win.

References

1 Samson Abramsky, Anuj Dawar, and Pengming Wang. The pebbling comonad in finite model theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005129.

2 Isolde Adler. Marshals, monotone marshals, and hypertree-width. *J. Graph Theory*, 47(4):275–296, 2004. doi:10.1002/JGT.20025.

3 Isolde Adler. Games for width parameters and monotonicity. *CoRR*, abs/0906.3857, 2009. arXiv:0906.3857.

4 Isolde Adler and Eva Fluck. Monotonicity of the cops and robber game for bounded depth treewidth. *CoRR*, abs/2402.09139, 2024. doi:10.48550/arXiv.2402.09139.

5 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007. doi:10.1016/J.EJC.2007.04.013.

6 Martin Aigner and M. Fromme. A game of cops and robbers. *Discret. Appl. Math.*, 8(1):1–12, 1984. doi:10.1016/0166-218X(84)90073-8.

7 Omid Amini, Frédéric Mazoit, Nicolas Nisse, and Stéphan Thomassé. Submodular partition functions. *Discret. Math.*, 309(20):6000–6008, 2009. doi:10.1016/J.DISC.2009.04.033.

8 Daniel Bienstock. Graph searching, path-width, tree-width and related problems (A survey). In Fred Roberts, Frank Hwang, and Clyde L. Monma, editors, *Reliability Of Computer And Communication Networks, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, December 2-4, 1989*, volume 5 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 33–50. DIMACS/AMS, 1989. doi:10.1090/DIMACS/005/02.

9 Daniel Bienstock and Paul D. Seymour. Monotonicity in graph searching. *J. Algorithms*, 12(2):239–245, 1991. doi:10.1016/0196-6774(91)90003-H.

10 Hans L. Bodlaender and Dimitrios M. Thilikos. Computing small search numbers in linear time. 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 37–48. Springer, 2004. doi:10.1007/978-3-540-28639-4_4.

11 Anuj Dawar, Tomáš Jakl, and Luca Reggio. Lovász-Type Theorems and Game Comonads. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, June 2021. doi:10.1109/LICS52264.2021.9470609.

12 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, August 2010. doi:10.1002/jgt.20461.

13 Eva Fluck, Tim Seppelt, and Gian Luca Spitzer. Going Deep and Going Wide: Counting Logic and Homomorphism Indistinguishability over Graphs of Bounded Treedepth and Treewidth. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.27.

14 Fedor V. Fomin, Pierre Fraigniaud, and Nicolas Nisse. Nondeterministic graph searching: From pathwidth to treewidth. *Algorithmica*, 53(3):358–373, 2009. doi:10.1007/S00453-007-9041-6.

15 Fedor V. Fomin, Petr A. Golovach, and Jan Kratochvíl. On tractability of cops and robbers game. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *Fifth IFIP International Conference On Theoretical Computer Science - TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, September 7-10, 2008, Milano, Italy*, volume 273 of *IFIP*, pages 171–185. Springer, 2008. doi:10.1007/978-0-387-09680-3_12.

16 Fedor V. Fomin and Dimitrios M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008. doi:10.1016/J.TCS.2008.02.040.

17 Matthew K. Franklin, Zvi Galil, and Moti Yung. Eavesdropping games: a graph-theoretic approach to privacy in distributed systems. *J. ACM*, 47(2):225–243, 2000. doi:10.1145/333979.333980.

18 Archontia C. Giannopoulou, Paul Hunter, and Dimitrios M. Thilikos. Lifo-search: A min-max theorem and a searching game for cycle-rank and tree-depth. *Discret. Appl. Math.*, 160(15):2089–2097, 2012. doi:10.1016/J.DAM.2012.03.015.

19 Archontia C. Giannopoulou and Dimitrios M. Thilikos. A min-max theorem for lifo-search. *Electron. Notes Discret. Math.*, 38:395–400, 2011. doi:10.1016/J.ENDM.2011.09.064.

20 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 507–520, New York, NY, USA, 2020. Association for Computing Machinery. event-place: Saarbrücken, Germany. doi:10.1145/3373718.3394739.

21 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. doi:10.1145/2636918.

22 Martin Grohe, Gaurav Rattan, and Tim Seppelt. Homomorphism Tensors and Linear Equations. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 70:1–70:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.70.

23 Petr Hlinený and Geoff Whittle. Matroid tree-width. *Eur. J. Comb.*, 27(7):1117–1128, 2006. doi:10.1016/J.EJC.2006.06.005.

24 Geoffrey A. Hollinger, Athanasios Kehagias, and Sanjiv Singh. GSST: anytime guaranteed search. *Auton. Robots*, 29(1):99–118, 2010. doi:10.1007/S10514-010-9189-9.

25 Paul Hunter and Stephan Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3):206–219, 2008. doi:10.1016/J.TCS.2008.02.038.

26 Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001. doi:10.1006/JCTB.2000.2031.

27 Stephan Kreutzer and Sebastian Ordyniak. Digraph decompositions and monotonicity in digraph searching. *Theor. Comput. Sci.*, 412(35):4688–4703, 2011. doi:10.1016/j.tcs.2011.05.003.

28 Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993. doi:10.1145/151261.151263.

29 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3):321–328, September 1967. doi:10.1007/BF02280291.

30 Fillia Makedon and Ivan Hal Sudborough. On minimizing width in linear layouts. *Discret. Appl. Math.*, 23(3):243–265, 1989. doi:10.1016/0166-218X(89)90016-4.

31 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672, 2020. doi:10.1109/FOCS46700.2020.00067.

32 Frédéric Mazoit and Nicolas Nisse. Monotonicity of non-deterministic graph searching. *Theor. Comput. Sci.*, 399(3):169–178, 2008. doi:10.1016/J.TCS.2008.02.036.

33 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/J.EJC.2005.01.010.

34 Daniel Neuen. Homomorphism-Distinguishing Closedness for Graphs of Bounded Tree-Width, April 2023. doi:10.48550/arXiv.2304.07011.

35 Jan Obdrzálek. Dag-width: connectivity measure for directed graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 814–821. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109647.

36 T. D. Parsons. Pursuit-evasion in a graph. In Yousef Alavi and Don R. Lick, editors, *Theory and Applications of Graphs*, pages 426–441, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

37 Torrence D Parsons. The search number of a connected graph. In *Proc. 9th South-Eastern Conf. on Combinatorics, Graph Theory, and Computing*, pages 549–554, 1978.

38 Nikolai N. Petrov. A problem of pursuit in the absence of information on the pursued. *Differentsial'nye Uravneniya*, 18(1):345–1352, 1982.

39 David E. Roberson. Oddomorphisms and homomorphism indistinguishability over graphs of bounded degree, June 2022. doi:10.48550/arXiv.2206.10321.

40 David E. Roberson and Tim Seppelt. Lasserre Hierarchy for Graph Isomorphism and Homomorphism Indistinguishability. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 101:1–101:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.101.

41 Benjamin Scheidt and Nicole Schweikardt. Counting homomorphisms from hypergraphs of bounded generalised hypertree width: A logical characterisation. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 79:1–79:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.MFCS.2023.79.

42 Tim Seppelt. Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 82:1–82:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2023.82.

43 Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993. doi:10.1006/JCTB.1993.1027.

Quantum Polynomial Hierarchies: Karp-Lipton, Error Reduction, and Lower Bounds

Avantika Agarwal ✉
David R. Cheriton School of Computer Science and Institute for Quantum Computing,
University of Waterloo, Canada

Sevag Gharibian ✉
Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS),
Paderborn University, Germany

Venkata Koppula ✉
Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India

Dorian Rudolph ✉
Department of Computer Science and Institute for Photonic Quantum Systems (PhoQS),
Paderborn University, Germany

Abstract

The Polynomial-Time Hierarchy (PH) is a staple of classical complexity theory, with applications spanning randomized computation to circuit lower bounds to "quantum advantage" analyses for near-term quantum computers. Quantumly, however, despite the fact that at least *four* definitions of quantum PH exist, it has been challenging to prove analogues for these of even basic facts from PH. This work studies three quantum-verifier based generalizations of PH, two of which are from [Gharibian, Santha, Sikora, Sundaram, Yirka, 2022] and use classical strings (QCPH) and quantum mixed states (QPH) as proofs, and one of which is new to this work, utilizing quantum pure states (pureQPH) as proofs. We first resolve several open problems from [GSSSY22], including a collapse theorem and a Karp-Lipton theorem for QCPH. Then, for our new class pureQPH, we show one-sided error reduction pureQPH, as well as the first bounds relating these quantum variants of PH, namely $\mathsf{QCPH} \subseteq \mathsf{pureQPH} \subseteq \mathsf{EXP}^{\mathsf{PP}}$.

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

Keywords and phrases Quantum complexity, polynomial hierarchy

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.7

Related Version *Full Version*: `https://arxiv.org/abs/2401.01633` [5]

Funding *Sevag Gharibian*: Supported by the DFG under grant numbers 450041824 and 432788384, the BMBF within the funding program "Quantum Technologies - from Basic Research to Market" via project PhoQuant (grant number 13N16103), and the project "PhoQC" from the programme "Profilbildung 2020", an initiative of the Ministry of Culture and Science of the State of Northrhine Westphalia.
Venkata Koppula: Supported by the Pankaj Gupta Fellowship at IIT Delhi.

Acknowledgements We thank Chirag Falor, Shu Ge, Anand Natarajan, Sabee Grewal, and Justin Yirka for the pleasure of productive discussions during the concurrent development of our works. This work was completed in part while Avantika Agarwal was a student at Indian Institute of Technology Delhi and in part while visiting Paderborn University.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 7; pp. 7:1–7:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Introduced by Stockmeyer in 1976 [28], the Polynomial-Time Hierarchy (PH) is one of the foundation stones of classical complexity theory. Intuitively, the levels of PH, denoted Σ_i^p (respectively, Π_i^p) for $i \geq 1$, yield progressively harder, yet natural, "steps up" from NP (respectively, coNP). Specifically, a Σ_i^p verifier is a deterministic poly-time Turing Machine M which, given input $x \in \{0,1\}^n$, takes in i proofs $y_i \in \{0,1\}^{\text{poly}(n)}$, and satisfies:

$$\text{if } x \text{ is a YES input:} \qquad \exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i \text{ s.t. } M(x, y_1, \ldots, y_i) = 1 \tag{1}$$

$$\text{if } x \text{ is a NO input:} \qquad \forall y_1 \exists y_2 \forall y_3 \cdots \overline{Q_i} y_i \text{ s.t. } M(x, y_1, \ldots, y_i) = 0. \tag{2}$$

Above, Q_i is $\forall$ ($\exists$) if i is even (odd). PH has played a prominent (and often surprising!) role in capturing the complexity of various computing setups, including the power of BPP [26, 23], low-depth classical circuits [11], counting classes [29], and even near-term quantum computers [9, 1, 8].

In contrast, the role of *quantum* analogues of PH in quantum complexity theory remains embarrassingly unknown. So, where is the bottleneck? *Defining* "quantum PH" is not the problem – indeed, Yamakami [31], Lockhart and González-Guillén [18], and Gharibian, Santha, Sikora, Sundaram and Yirka [13] all gave different definitions of quantum PH. Instead, the difficulty lies in proving even basic properties of quantum PH, which often runs up against difficult phenomena lurking about open problems such as $\exists \cdot \mathsf{BPP} \stackrel{?}{=} \mathsf{MA}$ and $\mathsf{QMA} \stackrel{?}{=} \mathsf{QMA}(2)$.

In this work, we resolve open questions regarding some fundamental properties of quantum PH. We focus on three definitions of quantum PH, chosen because they naturally generalize[1] QCMA and QMA. The first two definitions are from [13] (formal definitions in Section 2), and the third is new to this work. The definitions all use a poly-time uniformly generated quantum verifier V, and are given as follows (for brevity, here we only state the YES case definitions):

- QCPH: $\exists y_1 \forall y_2 \exists y_3 \cdots Q_i y_i$ s.t. $V(x, y_1, \ldots, y_i)$ outputs 1 with probability $\geq 2/3$.
- QPH: $\exists \rho_1 \forall \rho_2 \exists \rho_3 \cdots Q_i \rho_i$ s.t. $V(x, \rho_1, \ldots, \rho_i)$ outputs 1 with probability $\geq 2/3$.
- pureQPH: $\exists |\psi_1\rangle \forall |\psi_2\rangle \exists |\psi_3\rangle \cdots Q_i |\psi_i\rangle$ s.t. $V(x, |\psi_1\rangle, \ldots, |\psi_i\rangle)$ outputs 1 with probability $\geq 2/3$.

In words, QCPH, QPH, and pureQPH utilize poly-size quantum verifiers taking in classical, mixed quantum, and pure quantum proofs, respectively. It is immediate from the definitions that $\mathsf{QCMA} \subseteq \mathsf{QCPH}$, $\mathsf{QMA} \subseteq \mathsf{QPH}$, and $\mathsf{QMA} \subseteq \mathsf{pureQPH}$. Beyond this, not much is clear. For example, a standard use of PH is via its collapse theorem – if for any i, $\Sigma_i^p = \Pi_i^p$, then $\mathsf{PH} = \Sigma_i^p$. Do any of QCPH, QPH, or pureQPH satisfy such a collapse theorem? Does error reduction hold for QPH or pureQPH? What is the relationship between QCPH, QPH, and pureQPH? Note that standard convexity arguments (as used for e.g. QMA) cannot be used to argue $\mathsf{QPH} = \mathsf{pureQPH}$, due to the presence of alternating quantifiers (which make the verification non-convex in the proofs). Can one recover celebrated results for these hierarchies analogous to the Karp-Lipton [20] Theorem for PH?

Our results. *1. Collapse Theorem for* QCPH. We first resolve an open question of [13] by giving a collapse theorem for QCPH.

[1] QCMA and QMA are quantum generalizations of Merlin-Arthur (MA), with a classical proof and quantum verifier and a quantum proof and quantum verifier, respectively.

▶ **Theorem 1.** *If for any $k \geq 1$, $\mathsf{QC\Sigma}_k = \mathsf{QC\Pi}_k$, then $\mathsf{QCPH} = \mathsf{QC\Sigma}_k$.*

This is in contrast to QPH, for which a collapse theorem is believed difficult to show, as it would imply a subsequent collapse[2] $\mathsf{QMA}(2) \subseteq \mathsf{PSPACE}$.

2. Quantum-Classical Karp-Lipton Theorem for QCPH*.* The celebrated Karp-Lipton theorem [20] states that if SAT can be solved by polynomial-size circuits, then PH collapses to Σ_2^p. We next leverage Theorem 1 and other techniques to obtain a Karp-Lipton Theorem for QCPH:

▶ **Theorem 2** (Karp-Lipton for QCPH)**.** *If $\mathsf{QCMA} \subseteq \mathsf{BQP}_{/\mathrm{mpoly}}$, then $\mathsf{QCPH} = \mathsf{QC\Sigma}_2 = \mathsf{QC\Pi}_2$.*

Here, $\mathsf{BQP}_{/\mathrm{mpoly}}$ is BQP with poly-size classical advice (Definition 11). In words, Theorem 2 says QCMA cannot be solved by (even non-uniformly generated) poly-size quantum circuits, unless QCPH collapses to its second level. This resolves a second open question of [13].

3. Error reduction for pureQPH*.* While error reduction for QCPH follows from parallel repetition (due to its classical proofs), achieving it for pureQPH is non-trivial for the same reason it is non-trivial for QMA(2) – the tensor product structure between proofs is not necessarily preserved when postselecting on measurements across proof copies in the NO case. Here, we show *one-sided* error reduction for pureQPH (e.g. *exponentially* small soundness):

▶ **Theorem 3.** *For all $i > 0$ and $c - s \geq 1/p(n)$ for some polynomial p,*

1. *For even $i > 0$:*
   a. $\mathsf{pureQ\Sigma}_i(c, s) \subseteq \mathsf{pureQ\Sigma}_i^{\mathsf{SEP}}(1/np(n)^2, 1/e^n)$
   b. $\mathsf{pureQ\Pi}_i(c, s) \subseteq \mathsf{pureQ\Pi}_i^{\mathsf{SEP}}(1 - 1/e^n, 1 - 1/np(n)^2)$
2. *For odd $i > 0$:*
   a. $\mathsf{pureQ\Sigma}_i(c, s) \subseteq \mathsf{pureQ\Sigma}_i^{\mathsf{SEP}}(1 - 1/e^n, 1 - 1/np(n)^2)$
   b. $\mathsf{pureQ\Pi}_i(c, s) \subseteq \mathsf{pureQ\Pi}_i^{\mathsf{SEP}}(1/np(n)^2, 1/e^n)$

Above, $\mathsf{pureQ\Sigma}_i^{\mathsf{SEP}}$ and $\mathsf{pureQ\Pi}_i^{\mathsf{SEP}}$ have the promise that in the YES case, the verifier's acceptance measurement is separable (see Section 4.2). We remark the proof of this uses a new *asymmetric* version of the Harrow-Montanaro [16] Product Test, which may be of independent interest (see Lemma 13). The reason we are unable to recover exponentially small error simultaneously for both completeness and soundness is because our approach requires the final quantifier to be $\exists$.

4. Upper and lower bounds on pureQPH*.* Having introduced pureQPH in this work, we next give bounds on its power.

▶ **Theorem 4.** $\mathsf{QCPH} \subseteq \mathsf{pureQPH} \subseteq \mathsf{EXP}^{\mathsf{PP}}$*.*

While the upper bound above is not difficult to show (Theorem 17; this may be viewed as an "exponential analogue" of Toda's theorem), the lower bound is surprisingly subtle. The naive strategy of replacing each proof y_i of QCPH with pure state proof $|\psi_i\rangle$, which is then measured in the standard basis, does not work, as the measurement gives rise to mixed states. Mixed states, in turn, are difficult to handle in QPH, as the latter is not a convex optimization due to alternating quantifiers. We remark that while $\mathsf{QPH} \subseteq \mathsf{pureQPH}$ follows easily via purification of proofs, we do *not* know how to show the analogous lower bound $\mathsf{QCPH} \subseteq \mathsf{QPH}$ (our approach uses our asymmetric product test, which requires pure states).

[2] QMA(2) is QMA with two proofs in tensor product. Since its introduction in 2001 by Kobayashi, Matsumoto, and Yamakami [21, 22], its complexity remains stubbornly open. The current best bounds are $\mathsf{QMA} \subseteq \mathsf{QMA}(2) \subseteq \mathsf{Q\Sigma}_3 \subseteq \mathsf{NEXP}$, where the second and third containments are from[13].

Related Work. Yamakami [31] gave the first definition of a quantum PH, which takes quantum inputs (in contrast, we use classical inputs). The same paper [31] also discusses a variant of $\mathsf{pureQPH}$ (which they call QPH). However, their QPH is very powerful and its first level already captures QMA(2) (which is contained in the third level of $\mathsf{pureQPH}$), and error reduction is trivial for this complexity class. Gharibian and Kempe [12] defined and obtained hardness of approximation results for $\mathsf{QC\Sigma}_2$, obtaining the first hardness of approximation results for a quantum complexity class. (See [7] for a recent extension to QCMA-hardness of approximation results.) Lockhart and González-Guillén [18] defined a class QCPH' similar to QCPH, except using existential and universal *operators.* Thus, in [18] $\mathsf{QC\Sigma}_1' = \exists \cdot \mathsf{BQP}$, which is not known to equal QCMA (for the same reason $\exists \cdot \mathsf{BPP} \overset{?}{=} \mathsf{MA}$ remains open). In exchange for not capturing QCMA, however, the benefit of QCPH' is that its properties are easier to prove than QCPH. Gharibian, Santha, Sikora, Sundaram and Yirka [13] defined QCPH and QPH, and showed weaker variants of the Karp-Lipton theorem (Precise-$\mathsf{QCMA} \subseteq \mathsf{BQP}_{/\mathrm{mpoly}}$ implies $\mathsf{QC\Sigma}_2 = \mathsf{QC\Pi}_2$) and Toda's theorem [29] ($\mathsf{QCPH} \subseteq \mathsf{P}^{\mathsf{PP}^{\mathsf{PP}}}$). They also showed $\mathsf{QMA(2)} \subseteq \mathsf{Q\Sigma}_3 \subseteq \mathsf{NEXP}$, giving the first class sitting between $\mathsf{QMA(2)}$ and NEXP, and observed that $\mathsf{Q\Sigma}_2 = \mathsf{Q\Pi}_2 = \mathrm{QRG}(1) \subseteq \mathsf{PSPACE}$ (due to work of Jain and Watrous [19]). Finally, Aaronson, Ingram and Kretschmer [4] showed that relative to a random oracle, PP is not in the "QMA hierarchy", i.e. in $\mathsf{QMA}^{\mathsf{QMA}^{\cdot^{\cdot^{\cdot}}}}$. The relationship between this QMA hierarchy and any of QCPH, QPH, or $\mathsf{pureQPH}$ remains open.

The Karp-Lipton theorem has been studied in the setting of quantum advice. Prior works by Aaronson and Drucker [3], and Aaronson, Cojocaru, Gheorghiu, and Kashefi [2] studied the consequences of solving NP-complete problems using polynomial-sized quantum circuits with polynomial quantum advice. Nishimura and Yamakami [25] define the *language* class BQP with classical advice and compare it to BQP with quantum advice. In this paper, however, we study promise problems in BQP with classical advice (called $\mathsf{BQP}_{/\mathrm{mpoly}}$).

Finally, the Product Test was first introduced by Mintert, Kuŝ and Buchleitner [24], and rigorously analyzed and strikingly leveraged by Harrow and Montanaro[16] to show $\mathsf{QMA}(k) = \mathsf{QMA}(2)$ for polynomial k, as well as error reduction for $\mathsf{QMA}(2)$. (See also Soleimanifar and Wright [27].)

Concurrent Work. We mention two concurrent works on quantum variants of the polynomial hierarchy. First, our collapse theorem for QCPH (Theorem 1) was proven concurrently by Falor, Ge, and Natarajan [10]. Second, we upper bound QCPH by showing that for all k, $\mathsf{QC\Pi}_k \subseteq \mathsf{pureQ\Sigma}_k \subseteq \mathsf{pureQ\Sigma}_i \subseteq \mathsf{NEXP}^{\mathsf{NP}^{i-1}}$, implying $\mathsf{QCPH} \subseteq \mathsf{pureQPH} \subseteq \mathsf{EXP}^{\mathsf{PP}}$ (Theorem 4 and Theorem 17). Grewal and Yirka [15] show the stronger bound $\mathsf{QCPH} \subseteq \mathsf{QPH}$, at the expense of the minor caveat that their proof does not obtain level-wise containment for all k, but rather requires constant factor blowup in level. Beyond this, our papers diverge. We show a Quantum-Classical Karp-Lipton Theorem for QCPH and error reduction for $\mathsf{pureQPH}$. Grewal and Yirka [15] define a new quantum polynomial hierarchy called the *entangled quantum polynomial hierarchy* (QEPH), which allows entanglement across alternatively quantified quantum proofs. They show that QEPH collapses to its second level (even with polynomially many proofs), and is equal to $\mathsf{QRG}(1)$, the class of one round quantum-refereed games. They also define a generalization of QCPH, denoted DistributionQCPH, in which proofs are not strings but *distributions* over strings. They show QCPH=DistributionQCPH.

Techniques. We sketch our approach for each result mentioned above.

1. Collapse Theorem for QCPH*.* Collapse theorems for PH are shown via inductive argument – by fixing an arbitrary proof for the first quantifier of Σ_i^p, one obtains an instance of Π_{i-1}^p. Reference [13] noted this approach does not obviously work for QCPH, as fixing the first proof of $\mathsf{QC\Sigma}_i$ does *not* necessarily yield a valid $\mathsf{QC\Pi}_{i-1}$ instance (i.e. the latter might not satisfy the desired promise gap). We bypass this obstacle by observing that even if most choices for existentially quantified proofs are problematic, there always exists *at least one* "good" choice, for which the recursion works. Formally, we are implicitly using a *promise* version of NP which is robustly[3] defined relative to any promise oracle.

2. Quantum-Classical Karp-Lipton Theorem. The classical Karp-Lipton theorem crucially uses the search-to-decision reduction for SAT. Given a (non-uniform) circuit family that can decide a SAT instance, we can use the circuit family to find a witness for a SAT instance. However, this search-to-decision reduction does not work in the quantum-classical setting since we are working with promise problems instead of languages. As a result, we cannot replicate the classical proof in the quantum-classical setting. Instead, we first convert the QCMA problem (obtained by fixing the universally quantified proof) to a UniqueQCMA (UQCMA) problem. For this, we use the quantum-classical analogue of Valiant-Vazirani's isolation lemma [30] given by Aharonov, Ben-Or, Brandão, and Sattath [6]. We then use a single-query (quantum) search-to-decision reduction for UQCMA that was presented in a recent work by Irani, Natarajan, Nirkhe, Rao and Yuen [17].

3. Error reduction for $\mathsf{pureQPH}$*.* As with $\mathsf{QMA}(2)$, the challenge with error reduction via parallel repetition for $\mathsf{pureQPH}$ is the following: Given proof $|\psi\rangle_{A_1,B_1}\otimes|\phi\rangle_{A_2,B_2}$, postselecting on a joint measurement outcome on registers $\{A_1, B_1\}$ may entangle registers $\{A_2, B_2\}$. To overcome this, we give an asymmetric version of the Product Test [16], denoted APT. The APT takes in an n-system state $|\psi\rangle$ in register A, and (ideally) m copies of $|\psi\rangle$ in register B. It picks a random subsystem i of A, as well as a random copy j of i in B, and applies the SWAP test (Figure 1) between them. We prove (Lemma 13) that if this test passes with high probability, then $|\psi\rangle_A \approx |\psi_1\rangle\otimes\cdots\otimes|\psi_n\rangle$ for some $\{|\psi_i\rangle\}_{i=1}^n$, i.e. $|\psi\rangle_A$ was of tensor product form.

With the APT in hand, we can show error reduction for (e.g.) $\mathsf{pureQ\Pi}_i$. Here, the aim of an honest ith (existentially quantified) prover is to send many copies of proofs 1 through $i-1$. With probability 1/2, the verifier runs the APT with register A being proofs 1 to $i-1$ and register B being all their copies bundled with the ith proof, and with probability 1/2, the verifier runs parallel repetition on all copies of proofs bundled with the ith proof. This crucially leverages the fact that the ith proof is existential in the YES case, and thus can be assumed to be of this ideal form. In the NO case, however, the ith proof is universally quantified – thus, we cannot assume anything about its structure, which is why we do not get error reduction for the soundness parameter (in this case).

4. Upper and lower bounds on $\mathsf{pureQPH}$*.* We focus on the more difficult direction, $\mathsf{QCPH}\subseteq\mathsf{pureQPH}$, for which we actually show that for all even $k\geq 2$, $\mathsf{QC\Pi}_k\subseteq\mathsf{pureQPH}_k$ (Lemma 16). When simulating QCPH, the challenge is again how to deal with universally quantified proofs, denoted by index set $U\subseteq[k]$. Unlike existentially quantified proofs, which can be assumed to be set honestly to some optimal string y_i, for any index $j\in U$ the proof $|\psi_j\rangle$ can be any pure quantum state. This causes two problems: (1) Measuring $|\psi_j\rangle$ in the standard basis yields a *distribution* D_j over strings, and due to non-convexity of $\mathsf{pureQPH}$, it is not clear if

[3] Formally, let M be a deterministic machine with access to a promise oracle O. We say M is robust [14] if, regardless of how any invalid queries to O (i.e. queries violating the promise gap of O) are answered, M returns the same answer. One can define "PromiseNP" for non-deterministic M with access to O similarly: In the YES case, M has at least one robust accepting branch, and in the NO case, all branches of M are robust and rejecting. For clarity, we do not formally define and use PromiseNP in this work, but the viewpoint sketched here is equivalent to our approach for Theorem 1.

this can help a cheating prover succeed with higher probability that having sent a string. (2) Conditioned on distribution D_j, what should the next, existentially quantified, prover $j+1$ set its optimal proof/string to? We overcome these obstacles as follows. The initial setup is similar to Theorem 3 – prover k (which is existentially quantified in the YES case) send copies of all previous proofs 1 to $k-1$, and with probability $1/2$, we run the APT. However, now with probability $1/2$, we measure *all* proofs in the standard basis. The key step is to immediately *accept* if for any universally quantified proof index $i \in U$, measuring proof $|\psi_i\rangle$ does *not* match all of its copies bundled in proof k. In contrast, for existentially quantified proofs, we *reject* if a mismatch occurs. Finally, assuming no mismatches occur, we simply run the original QCPH verifier on the corresponding strings obtained via measurement. Showing correctness is subtle, and requires a careful analysis for both YES and NO cases, since recall the location of universally quantified proofs changes between cases.

Discussion and open questions. Many questions remain open for QCPH, QPH, and pureQPH. Perhaps the most frustrating for QCPH is a lack of a genuine Toda's theorem – [13] shows QCPH $\subseteq$ $\mathsf{P}^{\mathsf{PP}^{\mathsf{PP}}}$, but what one really wants is containment in P^{PP}. Is this possible? And if not, can one show an oracle separation between QCPH and P^{PP}? Moving to QPH, its role in this mess remains rather murky. Is pureQPH $\subseteq$ QPH (recall the converse direction follows via purification)? For this, our proof technique for QCPH $\subseteq$ pureQPH appears not to apply, as it requires pure states for the SWAP test. QPH *does* have one advantage over pureQPH, however – while the third level of both contains QMA(2), only $\mathsf{Q}\Sigma_3$ is known to be in NEXP [13], providing a class "between" QMA(2) and NEXP. Reference [13]'s proof breaks down for pureQPH, as its semidefinite-programming approach requires *mixed* state proofs.[4] Finally, for pureQPH, can one show two-sided error reduction? Is there a collapse theorem for pureQPH? Can one improve our bound pureQPH $\subseteq$ $\mathsf{EXP}^{\mathsf{PP}}$? As a first step, is pure$\mathsf{Q}\Sigma_3$ $\subseteq$ NEXP? If not, this would suggest the combination of "unentanglement" across proofs and alternating quantifiers yields a surprisingly powerful proof system, as it trivially holds that QMA(2) $\subseteq$ NEXP.

Organization. Section 2 begins with notation and definitions. Section 3.1 and Section 3.2 give our collapse theorem and Karp-Lipton theorem for QCPH, respectively. Section 4 shows error reduction for pureQPH. Section 5 gives upper and lower bounds for pureQPH.

2 Preliminaries

Notation. Let conv(S) denote the convex hull of set S. Then, the set of separable operators acting on $\mathbb{C}^{d_1} \otimes \cdots \otimes \mathbb{C}^{d_i}$ is

$$\operatorname{conv}\left(|\psi_1\rangle\langle\psi_1| \otimes \cdots \otimes |\psi_i\rangle\langle\psi_i| \mid \forall j \in [i],\ |\psi_j\rangle \in \mathbb{C}^{d_j} \text{ is a unit vector}\right). \tag{3}$$

Throughout this paper, we study *promise problems*. A promise problem is a pair $A = (A_{\mathrm{yes}}, A_{\mathrm{no}})$ such that $A_{\mathrm{yes}}, A_{\mathrm{no}} \subseteq \{0,1\}^*$ and $A_{\mathrm{yes}} \cap A_{\mathrm{no}} = \emptyset$, but $A_{\mathrm{yes}} \cup A_{\mathrm{no}} = \{0,1\}^*$ does not necessarily hold.

[4] Briefly, in NEXP one can guess the first existentially quantified proof of $\mathsf{Q}\Sigma_3$, leaving a $\mathsf{Q}\Pi_2$ computation. Since we are using mixed states, via duality theory one can rephrase this via an exponential-side SDP, which can be solved in exponential time. Note this "convexification" does not seem to apply for larger values of k.

2.1 Quantum-Classical Polynomial Hierarchy (QCPH)

We first recall the quantum analogue of PH that generalizes QCMA, i.e. has classical proofs [13].

▶ **Definition 5** ($\mathsf{QC}\Sigma_i$)**.** *Let $A = (A_{\text{yes}}, A_{\text{no}})$ be a promise problem. We say that A is in $\mathsf{QC}\Sigma_i(c,s)$ for poly-time computable functions $c, s : \mathbb{N} \mapsto [0,1]$ if there exists a poly-bounded function $p : \mathbb{N} \mapsto \mathbb{N}$ and a poly-time uniform family of quantum circuits $\{V_n\}_{n\in\mathbb{N}}$ such that for every n-bit input x, V_n takes in classical proofs $y_1 \in \{0,1\}^{p(n)}, \dots, y_i \in \{0,1\}^{p(n)}$ and outputs a single qubit, such that:*

- *Completeness: $x \in A_{\text{yes}} \Rightarrow \exists y_1 \forall y_2 \dots Q_i y_i$ s.t. $\text{Prob}[V_n$ accepts $(y_1, \dots, y_i)] \geq c$.*
- *Soundness: $x \in A_{\text{no}} \Rightarrow \forall y_1 \exists y_2 \dots \overline{Q}_i y_i$ s.t. $\text{Prob}[V_n$ accepts $(y_1, \dots, y_i)] \leq s$.*

Here, Q_i equals $\exists$ when m is odd and equals $\forall$ otherwise and $\overline{Q}_i$ is the complementary quantifier to Q_i. Finally, define

$$\mathsf{QC}\Sigma_i := \bigcup_{c-s\in\Omega(1/\operatorname{poly}(n))} \mathsf{QC}\Sigma_i(c,s). \tag{4}$$

Comments: Note that the first level of this hierarchy corresponds to QCMA. The complement of the i^{th} level of the hierarchy, $\mathsf{QC}\Sigma_i$, is the class $\mathsf{QC}\Pi_i$ defined next.

▶ **Definition 6** ($\mathsf{QC}\Pi_i$)**.** *Let $A = (A_{\text{yes}}, A_{\text{no}})$ be a promise problem. We say that $A \in \mathsf{QC}\Pi_i(c,s)$ for poly-time computable functions $c, s : \mathbb{N} \mapsto [0,1]$ if there exists a polynomially bounded function $p : \mathbb{N} \mapsto \mathbb{N}$ and a poly-time uniform family of quantum circuits $\{V_n\}_{n\in\mathbb{N}}$ such that for every n-bit input x, V_n takes in classical proofs $y_1 \in \{0,1\}^{p(n)}, \dots, y_i \in \{0,1\}^{p(n)}$ and outputs a single qubit, such that:*

- *Completeness: $x \in A_{\text{yes}} \Rightarrow \forall y_1 \exists y_2 \dots Q_i y_i$ s.t. $\text{Prob}[V_n$ accepts $(y_1, \dots, y_i)] \geq c$.*
- *Soundness: $x \in A_{\text{no}} \Rightarrow \exists y_1 \forall y_2 \dots \overline{Q}_i y_i$ s.t. s.t. $\text{Prob}[V_n$ accepts $(y_1, \dots, y_i)] \leq s$.*

Here, Q_i equals $\forall$ when m is odd and equals $\exists$ otherwise, and $\overline{Q}_i$ is the complementary quantifier to Q_i. Finally, define

$$\mathsf{QC}\Pi_i := \bigcup_{c-s\in\Omega(1/\operatorname{poly}(n))} \mathsf{QC}\Pi_i(c,s). \tag{5}$$

Now the corresponding quantum-classical polynomial hierarchy is defined as follows.

▶ **Definition 7** (Quantum-Classical Polynomial Hierarchy (QCPH))**.**

$$\mathsf{QCPH} = \bigcup_{m\in\mathbb{N}} \mathsf{QC}\Sigma_i = \bigcup_{m\in\mathbb{N}} \mathsf{QC}\Pi_i. \tag{6}$$

2.2 (Pure-State) Quantum Polynomial Hierarchy (pureQPH)

Next, we introduce Quantum PH with pure-state proofs (for clarity, the definition below is new to this work). Prior work [13] defined QPH using mixed-state quantum proofs. Unlike QMA or QMA(2) (where one may argue due to convexity that pure states suffice) it is not clear how to use convexity arguments in the presence of alternating quantifiers.

▶ **Definition 8** ($\mathsf{pureQ}\Sigma_i$)**.** *A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in $\mathsf{pureQ}\Sigma_i(c,s)$ for poly-time computable functions $c, s : \mathbb{N} \mapsto [0,1]$ if there exists a polynomially bounded function $p : \mathbb{N} \mapsto \mathbb{N}$ and a poly-time uniform family of quantum circuits $\{V_n\}_{n\in\mathbb{N}}$ such that for every n-bit input x, V_n takes $p(n)$-qubit states $|\psi_1\rangle, \dots, |\psi_i\rangle$ as quantum proofs and outputs a single qubit, then:*

- *Completeness: If $x \in A_{\text{yes}}$, then $\exists|\psi_1\rangle\forall|\psi_2\rangle \dots Q_i|\psi_i\rangle$: V_n accepts $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_i\rangle$ with probability $\geq c$.*
- *Soundness: If $x \in A_{\text{no}}$, then $\forall|\psi_1\rangle\exists|\psi_2\rangle \dots \overline{Q}_i|\psi_i\rangle$: V_n accepts $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_i\rangle$ with probability $\leq s$.*

Here, Q_i equals $\forall$ when m is even and equals $\exists$ otherwise, and $\overline{Q}_i$ is the complementary quantifier to Q_i. Define

$$\mathsf{pureQ\Sigma}_i = \bigcup_{c-s\in\Omega(1/\operatorname{poly}(n))} \mathsf{pureQ\Sigma}_i(c,s). \tag{7}$$

▶ **Definition 9** (pureQΠ$_i$). *A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is in $\mathsf{pureQ\Pi}_i(c,s)$ for poly-time computable functions $c, s : \mathbb{N} \mapsto [0,1]$ if there exists a polynomially bounded function $p : \mathbb{N} \mapsto \mathbb{N}$ and a poly-time uniform family of quantum circuits $\{V_n\}_{n\in\mathbb{N}}$ such that for every n-bit input x, V_n takes $p(n)$-qubit states $|\psi_1\rangle, \dots, |\psi_i\rangle$ as quantum proofs and outputs a single qubit, then:*

- *Completeness: If $x \in A_{\text{yes}}$, then $\forall|\psi_1\rangle\exists|\psi_2\rangle \dots Q_i|\psi_i\rangle$: V_n accepts $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_i\rangle$ with probability $\geq c$.*
- *Soundness: If $x \in A_{\text{no}}$, then $\exists|\psi_1\rangle\forall|\psi_2\rangle \dots \overline{Q}_i|\psi_i\rangle$: V_n accepts $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_i\rangle$ with probability $\leq s$.*

Here, Q_i equals $\exists$ when m is even and equals $\forall$ otherwise, and $\overline{Q}_i$ is the complementary quantifier to Q_i. Define

$$\mathsf{pureQ\Pi}_i = \bigcup_{c-s\in\Omega(1/\operatorname{poly}(n))} \mathsf{pureQ\Pi}_i(c,s). \tag{8}$$

The Quantum Polynomial Hierarchy with pure-state proofs can now be defined as follows.

▶ **Definition 10** (Pure Quantum Poly-Hierarchy (pureQPH)).

$$\mathsf{pureQPH} = \bigcup_{m\in\mathbb{N}} \mathsf{pureQ\Sigma}_i = \bigcup_{m\in\mathbb{N}} \mathsf{pureQ\Pi}_i.$$

2.3 Other complexity classes

▶ **Definition 11** (BQP$_{/\text{mpoly}}$). *A promise problem $\Pi = (A_{\text{yes}}, A_{\text{no}})$ is in $\mathsf{BQP}_{/\text{mpoly}}$ if there exists a poly-sized family of quantum circuits $\{C_n\}_{n\in\mathbb{N}}$ and a collection of binary advice strings $\{a_n\}_{n\in\mathbb{N}}$ with $|a_n| = \operatorname{poly}(n)$, such that for all $n \in \mathbb{N}$ and all strings $x \in \{0,1\}^n$,*

$$\Pr[C_n(|x\rangle, |a_n\rangle) = 1] \geq 2/3 \text{ if } x \in A_{\text{yes}} \quad \text{and} \quad \Pr[C_n(|x\rangle, |a_n\rangle) = 1] \leq 1/3 \text{ if } x \in A_{\text{no}}.$$

3 Collapse theorems and Quantum Karp-Lipton

3.1 Collapse Theorem for QCPH

For the quantum-classical hierarchy, QCPH, we now show a quantum analogue of the standard collapse theorem for classical PH, i.e. $\mathsf{\Sigma}_2^p = \mathsf{\Pi}_2^p$ implies $\mathsf{PH} = \mathsf{\Sigma}_2^p$, resolving an open question of [13].

▶ **Lemma 12.** *If for any $k \geq 1$, $\mathsf{QC\Sigma}_k = \mathsf{QC\Pi}_k$, then for all $i \geq k$, $\mathsf{QC\Sigma}_i = \mathsf{QC\Pi}_i = \mathsf{QC\Sigma}_k$.*

Proof. We proceed by induction. For $j \geq k$, define $P(j) := \mathsf{QC\Sigma}_j = \mathsf{QC\Pi}_j = \mathsf{QC\Sigma}_k$. The base case $P(k)$ holds by the assumption of the lemma. For the inductive case, assume $P(j)$ holds for all $k \leq j \leq i-1$. We show $P(j)$ holds for $j = i$. Consider arbitrary promise problem $L = (L_{\text{yes}}, L_{\text{no}}, L_{\text{inv}}) \in \mathsf{QC\Sigma}_i$ and let $\{V_n\}$ be the verifier circuits for the promise problem. Define new promise problem $L' = (L'_{\text{yes}}, L'_{\text{no}}, L'_{\text{inv}})$:

$$L'_{\text{yes}} = \left\{ (x, y_1) \mid \forall y_2 \exists y_3 \ldots Q_i y_i \ \Pr[V(x, y_1, y_2, \ldots, y_i) = 1] \geq \frac{2}{3} \right\} \tag{9}$$

$$L'_{\text{no}} = \left\{ (x, y_1) \mid \exists y_2 \forall y_3 \ldots \overline{Q}_i y_i \ \Pr[V(x, y_1, y_2, \ldots, y_i) = 1] \leq \frac{1}{3} \right\} \tag{10}$$

$$L'_{\text{inv}} = \{0,1\}^* \setminus (L'_{\text{yes}} \cup L'_{\text{no}}). \tag{11}$$

Clearly, $L'_{\text{yes}} \cap L'_{\text{no}} = \emptyset$, and so $(L'_{\text{yes}}, L'_{\text{no}}, L'_{\text{inv}}) \in \mathsf{QC\Pi}_{i-1}$. By the induction hypothesis, there exists promise problem $L'' = (L''_{\text{yes}}, L''_{\text{no}}, L''_{\text{inv}}) \in \mathsf{QC\Sigma}_{i-1}$ such that $L'_{\text{no}} \subseteq L''_{\text{no}}$ and $L'_{\text{yes}} \subseteq L''_{\text{yes}}$. Letting $\{V''_n\}$ denote the verification circuits for L'', we have

$$\begin{aligned} (x, y_1) \in L'_{\text{yes}} \ &\Rightarrow \ (x, y_1) \in L''_{\text{yes}} \ \Rightarrow \ \exists y_2 \forall y_3 \ldots Q_i y_i \colon \ \Pr[V''(x, y_1, \ldots, y_i) = 1] \geq \tfrac{2}{3}, \\ (x, y_1) \in L'_{\text{no}} \ &\Rightarrow \ (x, y_1) \in L''_{\text{no}} \ \Rightarrow \ \forall y_2 \exists y_3 \ldots \overline{Q}_i y_i \colon \ \Pr[V''(x, y_1, \ldots, y_i) = 1] \leq \tfrac{1}{3}. \end{aligned} \tag{12}$$

Now considering again $L = (L_{yes}, L_{\text{no}}, L_{\text{inv}})$, we have

$$\begin{aligned} x \in L_{\text{yes}} \ &\Rightarrow \exists y_1 \colon (x, y_1) \in L'_{\text{yes}} \ \Rightarrow \exists y_1 \exists y_2 \forall y_3 \ldots Q_{i-1} y_i \Pr[V'(x, y_1, \ldots, y_i) = 1] \geq \tfrac{2}{3} \\ x \in L_{\text{no}} \ &\Rightarrow \forall y_1 \colon (x, y_1) \in L'_{\text{no}} \ \Rightarrow \forall y_1 \forall y_2 \exists y_3 \ldots \overline{Q}_{i-1} y_i \Pr[V'(x, y_1, \ldots, y_i) = 1] \leq \tfrac{1}{3}. \end{aligned} \tag{13}$$

We conclude $L \in \mathsf{QC\Sigma}_{i-1} = \mathsf{QC\Sigma}_i$. So $\mathsf{QC\Sigma}_j = \mathsf{QC\Sigma}_k$. Similarly, $\mathsf{QC\Pi}_i \subseteq \mathsf{QC\Pi}_{i-1} = \mathsf{QC\Sigma}_k$. Thus, $P(i)$ holds, as claimed. ◀

Theorem 1 follows from Lemma 12.

3.2 Quantum-Classical Karp-Lipton Theorem

We will show that if there exists a polynomial size circuit family $\{C_n\}_{n \in \mathbb{N}}$ that can decide a QCMA complete problem, then $\mathsf{QC\Pi}_2 \subseteq \mathsf{QC\Sigma}_2$. Using Theorem 1, it follows that if $\mathsf{QCMA} \subseteq \mathsf{BQP}_{/\text{mpoly}}$, then QCPH collapses to the second level.

▶ **Theorem 2** (Karp-Lipton for QCPH). *If* $\mathsf{QCMA} \subseteq \mathsf{BQP}_{/\text{mpoly}}$*, then* $\mathsf{QCPH} = \mathsf{QC\Sigma}_2 = \mathsf{QC\Pi}_2$.

The formal proof is presented in the full version of the paper, together with some immediate applications of the quantum-classical Karp-Lipton theorem. Here, we present an overview of the proof.

Proof-sketch. Let $L = (L_{\text{yes}}, L_{\text{no}}, L_{\text{inv}}) \in \mathsf{QC\Pi}_2$, and let V be the corresponding (quantum) verifier. Since the proofs are classical, we assume V has small error. We show that $L \in \mathsf{QC\Sigma}_2$ by using the following (quantum) verifier V': it takes as input an instance x, a quantum circuit C, a string y_1. The verifier V' runs the circuit C on input (x, y_1) and receives a string y_2. It then accepts x if V accepts (x, y_1, y_2). If $x \in L_{\text{no}}$, then there exists a y_1 such that for all y_2, $V(x, y_1, y_2)$ accepts with negligible probability. Therefore, for any circuit C, there exists a y_1 such that $V'(x, C, y_1) = V(x, y_1, C(x, y_1))$ is 1 with very low probability.

For any $x \in L_{\text{yes}}$, we have that for all y_1, the pair (x, y_1) is a YES-instance of a QCMA problem. Using the [6] isolation procedure, we obtain an instance $\phi_{(x,y_1)}$ such that with non-negligible probability $\phi_{(x,y_1)}$ has a unique witness. Next, using the assumption that $\mathsf{QCMA} \subseteq \mathsf{BQP}_{/\text{mpoly}}$, we get that there exists a circuit $\tilde{C}$ that can decide $\phi_{(x,y_1)}$. Finally, using the UQCMA search-to-decision procedure of [17], we can use C to find the unique

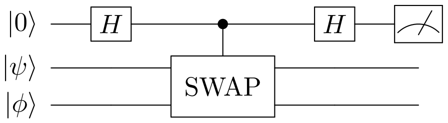


Figure 1 The SWAP test, whose output is the measurement result on the first wire.

witness for $\phi_{(x,y_1)}$. Let C be the circuit that, on input, (x, y_1), first performs the witness isolation from [6], followed by the UQCMA search-to-decision reduction from [17]. Putting these together, we get that there exists a circuit C that, for any y_1, finds a y_2 with non-negligible probability such that $V(x, y_1, y_2) = 1$. Therefore, there exists C such that for any y_1, $V'(x, C, y_1) = 1$ with non-negligible probability. ◀

4 Error reduction for pureQPH

We next study (weak) error reduction for pureQPH (pure proofs). For this, we first require an asymmetric generalization of the Product Test [24, 16], given in Section 4.1. We then give one-sided error reduction results in Section 4.2.

4.1 Asymmetric product test

We first give a generalization of the Product Test [24, 16], which we denote the Asymmetric Product Test (APT), stated as follows:

1. The input is $|\psi\rangle \in \mathbb{C}^{d_1} \otimes \cdots \otimes \mathbb{C}^{d_n}$ in register A, and $|\phi\rangle \in \mathbb{C}^{d^m}$ in register B, where for brevity $d := d_1 \cdots d_n$. We think of B as encoding m copies of A.
2. Choose $(i, j) \in [n] \times [m]$ uniformly at random.
3. Run the SWAP Test (Figure 1) between the ith register of A, and in B, the ith register of the jth copy of A.
4. Accept if the SWAP Test outputs 0, reject otherwise.

In fact, above, one can also assume that $|\phi\rangle$ in the APT is potentially entangled across two registers B and C, as we do for the main lemma of this section, given below.

▶ **Lemma 13.** *[Asymmetric Product Test (APT)] Define $d = d_1 \cdots d_n$. Consider $|\phi\rangle_{BC} \in \mathbb{C}^{d^m} \otimes \mathbb{C}^{d'}$ for some $d' > 0$. Suppose*

$$\max_{|\psi\rangle := |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle \in \mathbb{C}^d} \langle \phi|_{BC}[(|\psi\rangle\langle\psi|^{\otimes m})_B \otimes I_C]|\phi\rangle_{BC} = 1 - \varepsilon \tag{14}$$

for $\varepsilon \geq 0$. Then, given the state $|\eta\rangle_{ABC} := |\psi\rangle_A \otimes |\phi\rangle_{BC}$, the APT accepts with probability at most $1 - \varepsilon/2mn$.

Proof. Let $\{|\alpha_{ik}\rangle\}_{k\in[d_i]}$ be an orthonormal basis of $\mathbb{C}^{d_i}$ with $|\alpha_{i1}\rangle := |\psi_i\rangle$, and define index set

$$X = \{(x_{ij})_{i\in[n],j\in[m]} \mid \forall ij : x_{ij} \in [d_i]\}. \tag{15}$$

Then, rewrite $|\phi\rangle_{BC}$ in the $\{|\alpha_{ik}\rangle\}_k$ bases to obtain:

$$|\phi\rangle = \sum_{x\in X} a_x \left(\bigotimes_{ij} |\alpha_{i,x_{ij}}\rangle \right)_B |\gamma_x\rangle_C \tag{16}$$

for some states $|\gamma_x\rangle_C$. Without loss of generality, consider the swap test between the first register of A, $\mathbb{C}^{d_1}$ (which contains $|\psi_1\rangle$), and the first copy of $\mathbb{C}^{d_1}$ in B. Just prior to the final measurement in the test, we have the state $|\eta'\rangle_{SABC}$ given by (where S encodes the control qubit for SWAP in the SWAP test)

$$\begin{aligned}&\sum_{x\in X} a_x\Big(\frac{1}{2}|0\rangle_S\big(|\psi_1\rangle|\alpha_{1,x_{11}}\rangle+|\alpha_{1,x_{11}}\rangle|\psi_1\rangle\big)+\frac{1}{2}|1\rangle_S\big(|\psi_1\rangle|\alpha_{1,x_{11}}\rangle-|\alpha_{1,x_{11}}\rangle|\psi_1\rangle\big)\Big)\\&\qquad\otimes\bigotimes_{i\neq 1\text{ or }j\neq 1}|\alpha_{i,x_{ij}}\rangle|\gamma_x\rangle\;=:\;\sum_{x\in X}a_x|\eta'_x\rangle_{SABC}.\end{aligned}\tag{17}$$

We now show that the cross terms of $\langle\eta'|\eta'\rangle$ vanish, as $\langle\eta'_x|\eta'_y\rangle=0$ with $x\neq y$, where $x,y\in X$: (1) If $x_{ij}\neq y_{ij}$ for $(i,j)\neq(1,1)$, this follows immediately from orthonormality of the basis sets $\{|\alpha_{ik}\rangle\}_{k\in[d_i]}$. (2) The only remaining case is $x_{11}\neq y_{11}$. Without loss of generality, $y_{11}\neq 1$. Then $|\alpha_{1,y_{11}}\rangle$ is orthogonal to $|\psi_1\rangle$, again by choice of our basis set. Hence, $\langle\psi_1,\alpha_{1,x_{11}}|\alpha_{1,y_{11}},\psi_1\rangle=0$. Since trivially $\langle\psi_1,\alpha_{1,x_{11}}|\psi_1,\alpha_{1,y_{11}}\rangle=0$, we again have $\langle\eta'_x|\eta'_y\rangle=0$.

As for the non-cross-terms, we first have again from our basis choice that for any $x\in X$,

$$\langle\eta'_x|(|0\rangle\langle 0|_S)|\eta'_x\rangle=\begin{cases}1, & \text{if } x_{11}=1\\ \frac{1}{2}, & \text{if } x_{11}\neq 1\end{cases}.\tag{18}$$

Recall now that the APT selects $i\in[n]$ and $j\in[m]$ uniformly at random and does a SWAP test between $|\psi_i\rangle$ (the ith register of A) and the jth copy of the i register of B. Thus, conditioned on the APT randomly choosing $(i,j)=(1,1)$, we may bound its acceptance probability as

$$\Pr[\mathrm{APT}(|\eta\rangle)=1\mid i=1,j=1]=\langle\eta'|(|0\rangle\langle 0|_S)|\eta'\rangle=\sum_{\substack{x\in X\\ \text{s.t. } x_{11}=1}}|a_x|^2+\frac{1}{2}\sum_{\substack{x\in X\\ \text{s.t. } x_{11}\neq 1}}|a_x|^2.\tag{19}$$

By symmetry, an identical argument holds for any pair (i,j), and so

$$\begin{aligned}\Pr[\mathrm{APT}(|\eta\rangle)=1]&=\frac{1}{mn}\sum_{ij}\left(\sum_{\substack{x\in X\\ \text{s.t. } x_{ij}=1}}|a_x|^2+\frac{1}{2}\sum_{\substack{x\in X\\ \text{s.t. } x_{ij}\neq 1}}|a_x|^2\right) &(20)\\ &\leq|a_{1^{mn}}|^2+\frac{2mn-1}{2mn}\sum_{x\in X\setminus\{1^{mn}\}}|a_x|^2=1-\frac{\varepsilon}{2mn}. &(21)\end{aligned}$$

◀

4.2 One-sided error reduction

With Lemma 13 (APT) in hand, we now show one-sided error reduction for $\mathsf{pureQ\Pi}_i$, which suffices to obtain statements for all desired classes subsequently in Theorem 3. For this, we define classes $\mathsf{pureQ\Sigma}_i^{\mathsf{SEP}}$ and $\mathsf{pureQ\Pi}_i^{\mathsf{SEP}}$ as identical to $\mathsf{pureQ\Sigma}$ and $\mathsf{pureQ\Pi}$, respectively, except the measurement POVM of the verifier in the YES case must additionally be a separable operator relative to the cuts between each of the i proofs $|\psi_1\rangle$ to $|\psi_i\rangle$. (This is analogous to $\mathsf{QMA}(2)$ versus $\mathsf{QMA}^{\mathsf{SEP}}(2)$ [16].)

▶ **Lemma 14.** *[One-sided* $\mathsf{pureQ\Pi}_i$ *amplification] If* i *is even, then*

$$\mathsf{pureQ\Pi}_i(c,s)\subseteq\mathsf{pureQ\Pi}_i^{\mathsf{SEP}}\left(1-\frac{1}{e^n},1-\frac{1}{np(n)^2}\right),\tag{22}$$

for all functions c *and* s *such that* $c-s\geq 1/p(n)$ *for some polynomial* p.

Proof. Let L $= (L_{yes}, L_{\text{no}}, L_{\text{inv}}) \in \mathsf{pureQ\Pi}_i(c, s)$, with verifier V taking in i proofs denoted $|\psi_1\rangle, \ldots |\psi_i\rangle$. Since i is even, the last proof, $|\psi_i\rangle$, is existentially quantified. We define a new verifier V' to decide L in $\mathsf{pureQ\Pi}_i(1 - 1/\exp, 1 - 1/\operatorname{poly})$ as follows. V' receives the following proofs from an honest prover:

$$\big(|\psi_1'\rangle \otimes \cdots \otimes |\psi_{i-1}'\rangle\big)_A = (|\psi_1\rangle \otimes \cdots \otimes |\psi_{i-1}\rangle)_A \tag{23}$$

$$|\psi_i'\rangle_{BC} = \left(\bigotimes_{j=1}^{i-1} |\psi_j\rangle\right)_B^{\otimes m} \otimes |\psi_i\rangle_C^{\otimes m} \tag{24}$$

for $m \in \Theta(n(c-s)^{-2})$, and where A, B and C are used to align with the notation of Lemma 13 (which we use shortly). In words, the last prover sends m copies of the first $i-1$ proofs in register B, and m copies of the last proof $|\psi_i\rangle$ in register C. Then, V' acts as follows:

1. With probability 1/2, apply the APT (Lemma 13) between registers A and B. Accept iff the test accepts.
2. With probability 1/2, apply verifier V m times, taking one proof $|\psi_i\rangle$ from each respective subregister of B. Accept iff at least $(c+s)/2$ measurements accept.

Correctness. *YES case.* Since the i^{th} prover is existentially quantified, it sends the state in Equation (24). Thus, the APT accepts with certainty. Similarly, for parallel repetition of V, each repetition is independent, hence the overall verifier accepts with probability at least $1 - \exp(-(c-s)^2 m/2)$, as desired.

NO case. Now the i^{th} prover is universally quantified, hence it can send us a state entangled across BC.

Suppose the APT accepts with probability $1 - \varepsilon$. By Lemma 13,

$$|\psi_i'\rangle_{BC} = \alpha \left(\bigotimes_{j=1}^{i-1} |\psi_j\rangle\right)_B^{\otimes m} |\phi\rangle_C + \beta|\gamma\rangle_{BC} =: \alpha|\eta\rangle_B|\phi\rangle_C + \beta|\gamma\rangle_{BC} \tag{25}$$

for $|\alpha|^2 \geq 1 - 2m(i-1)\varepsilon$, and arbitrary states $|\phi\rangle, |\gamma\rangle$ satisfying that $|\eta\rangle_A|\phi\rangle_B$ is orthogonal to $|\gamma\rangle_C$. We have

$$\Pr[\text{parallel repetition of } V \text{ accepts } |\eta\rangle_A|\phi\rangle_B] \leq e^{\frac{-(c-s)^2 m}{2}}. \tag{26}$$

We conclude the acceptance probability of V' is at most

$$\frac{1}{2}\Big((1-\varepsilon) + (1 - 2m(i-1)\varepsilon)e^{-(c-s)^2 m/2} + 2m(i-1)\varepsilon \tag{27}$$

$$+ 2\sqrt{2m(i-1)\varepsilon + (2m(i-1))^2\varepsilon}\Big) \leq 1 - \frac{\varepsilon}{2}, \tag{28}$$

where the maximum is attained when $\varepsilon = \Theta(1/m(i-1))$.

Finally, that the measurement operator for the YES case is separable follows via the argument of Harrow and Montanaro for $\mathsf{QMA}(2)$ amplification [16], since our use of the APT is agnostic to whether proofs are universally or existentially quantified. ◀

Lemma 14 now easily generalizes to cover all classes regarding $\mathsf{pureQPH}$ we are concerned with:

▶ **Theorem 3.** *For all $i > 0$ and $c - s \geq 1/p(n)$ for some polynomial p,*

1. *For even $i > 0$:*
   - **a.** $\mathsf{pureQ\Sigma}_i(c, s) \subseteq \mathsf{pureQ\Sigma}_i^{\mathsf{SEP}}(1/np(n)^2, 1/e^n)$
   - **b.** $\mathsf{pureQ\Pi}_i(c, s) \subseteq \mathsf{pureQ\Pi}_i^{\mathsf{SEP}}(1 - 1/e^n, 1 - 1/np(n)^2)$

2. *For odd $i > 0$:*
   a. $\mathsf{pureQ\Sigma}_i(c,s) \subseteq \mathsf{pureQ\Sigma}_i^{\mathsf{SEP}}(1-1/e^n, 1-1/np(n)^2)$
   b. $\mathsf{pureQ\Pi}_i(c,s) \subseteq \mathsf{pureQ\Pi}_i^{\mathsf{SEP}}(1/np(n)^2, 1/e^n)$

Proof. Statement $1b$ is from Lemma 14, and $1a$ follows from $1b$ since we can get a $\mathsf{pureQ\Sigma}_i$ verifier by flipping the answer of a $\mathsf{pureQ\Pi}_i$ verifier corresponding to the complement of our promise problem. The remaining cases are analogous: $2a$ follows from $1b$, and $2b$ from $1a$. ◀

5 Upper and lower bounds on pureQPH

5.1 Lower bound: QCPH versus QPH

We first give a lower bound on pureQPH, by showing that alternatingly-quantified classical proofs can be replaced by pure-state quantum proofs.

▶ **Theorem 15.** QCPH $\subseteq$ pureQPH.

This follows immediately from the following lemma.

▶ **Lemma 16.** *For all even $k \geq 2$, $\mathsf{QC\Pi}_k \subseteq \mathsf{pureQ\Pi}_k$.*

Proof. That k is even implies the kth proof is existentially quantified in the YES case, a fact we will leverage. To begin, Let V be a verifier for $\mathsf{QC\Pi}_k$, so that in the YES case $\forall_1 x_1 \ldots \exists_k x_k : \Pr[V(x_1,\ldots,x_k)=1] \geq c$ and in the NO case $\exists x_1 \ldots \forall x_k : \Pr[V(x_1,\ldots,x_k)=1] \leq s$, where we may assume without loss of generality that c and s are exponentially close to 1 and 0, respectively. We construct a pureQPH verifier V' as follows:

- V' receives k proofs, $|\psi_1\rangle \otimes \cdots \otimes |\psi_k\rangle$.
- The last proof $|\psi_k\rangle$ consists of two registers denoted A and B. We think of A as containing m copies of proofs $1,\ldots,k-1$ (as in the APT, Lemma 13), and B as containing the kth proof for V, x_k.
- V' acts as follows:
  1. With probability 1/2, run the APT (Lemma 13), and accept if and only if the APT accepts.
  2. With probability 1/2, measure proofs $|\psi_1\rangle \otimes \cdots \otimes |\psi_{k-1}\rangle$ in the standard basis to obtain strings $x_1,\ldots,x_{k-1}$, respectively, similarly measure all copies of these proofs in A, and finally measure B in the standard basis to obtain $x_{k,B}$. Let $U = \{1, 3\ldots, k-1\}$ denote the indices of universally quantified proofs (in the YES case). Then:
     a. If there exists an $i \in [k-1]$ such that the strings obtained by measuring all copies of $|\psi_i\rangle$ did *not* equal x_i, let i denote the minimal such index. Accept if $i \in U$, and reject otherwise.
     b. Otherwise, simulate $V(x_1,\ldots,x_{k-1},x_{k,B})$.

Correctness strategy. Since we are trying to simulate $\mathsf{QC\Pi}_k$, ideally we want all proofs to be strings. This can be assumed without loss of generality for existentially quantified proofs, but not for universally quantified proofs, which can be set to any pure state by definition of $\mathsf{pureQ\Pi}_k$. So, for any $i \in U$, write $|\psi_i\rangle = \sum_x \alpha_{i,x}|x\rangle$, where we view $|\alpha_{i,x}|^2$ as a distribution over strings x for proof i. Denote for any $i \in U$ by x_i^* the amplitude of highest weight, i.e. $x_i^* = \arg\max_x |\alpha_{i,x}|$ (ties broken arbitrarily). The key idea is that the existentially quantified proof at index $i+1$ will now send string y_{i+1}^*, where y_{i+1}^* is the same string that a prover for the *original* $\mathsf{QC\Pi}_k$ verifier V would have sent in response to x_i^* on proof i. (For clarity, if $i+1=k$, then y_{i+1}^* is sent in register B.)

YES case. Since the kth proof is existentially quantified, for Step 1 (APT), we may assume all copies in $|\psi_{k,A}\rangle$ (the state in register A) are correctly set, so the APT accepts with probability 1, leaving all proofs invariant. As for Step 2, for each $i \in U$, let X_i be the random variable resulting from measuring $|\psi_i\rangle$ in the standard basis, and X_{ij} the random variables for the jth copy of $|\psi_i\rangle$ in register A. The verification can now fail in one of two ways:

1. There exists an $i \in U$ such that we did not measure the "right" result, i.e. $X_i \neq x_i^*$, but that all measured copies of $|\psi_i\rangle$ returned the same string, i.e. $\forall j\ X_{ij} = X_i$. In this case, our strategy for setting the existentially quantified proof $i+1$ is not necessarily the correct response to X_i. Thus, when Step 2(b) is run, we have no guarantee for the acceptance probability of V.
2. Measuring all $i \in U$ yields the desired outcomes x_i^* (as well as $\forall j\ X_{ij} = X_i$), but V nevertheless rejects due to imperfect completeness, i.e. $c < 1$.

Combining these, via the union bound we thus have for Step 2 that

$$\begin{aligned}\Pr[\text{reject}] &\leq \Pr[\ \exists i \in U : X_i \neq x_i^* \text{ AND } \forall ij : X_i = X_{ij}\ ] + (1-c) &(29)\\ &\leq \min\{p, 1-p\}^m + 1 - c &(30)\\ &\leq 2^{-m} + 1 - c, &(31)\end{aligned}$$

where $p := \max_i |\alpha_{i,x_i}|^2$, since $\Pr[X_i \neq x_i^*] = \Pr[X_{ij} \neq x_i^*] \leq \max\{p, 1-p\}$ because $|\alpha_{i,y}|^2 \leq 1-p$ for $y \neq x_i^*$. Since we assumed the APT accepts with perfect probability, we conclude V' accepts with probability $\geq \frac{1}{2} + \frac{1}{2}(c - 2^{-m}) =: c'$. (As an aside, recall c is exponentially close to 1.)

NO case. The analysis is more subtle in this case, as the set of indices $U = \{1, 3, \ldots, k-1\}$ now refers to *existentially* quantified proofs. Thus, in Step 2(a) when V' accepts iff $i \in U$, this now means it accepts on existentially quantified proofs. This is because V' does not know whether it is in a YES or NO case. For the same reason, the actions and role of the final proof $|\psi_k\rangle$ on registers A and B remain the same, even though it is now universally quantified. Finally, the strategy of any existential prover $i \in U$ is the same as the YES case: Prover i sends the optimal response y_i^* to universally quantified proof x_{i-1}^*. (If $i = 1$, then there is no universal proof to condition on for $|\psi_1\rangle$.)

To proceed, assume for now that the APT would have succeeded in Step 1 with certainty. In Step 2, define again for all $i \in U$, X_i the random variable resulting from measuring $|\psi_i\rangle$ in the standard basis, and X_{ij} the random variables for the jth copy of $|\psi_i\rangle$ in register A. The verifier can now fail in one of two ways, the first of which differs significantly from the YES case:

1. There exists $i \in U$ such that X_i mismatched one of its copies in A, i.e. $\exists j$ such that $X_i \neq X_{ij}$. *A priori*, this seems like a problem – since $|\psi_k\rangle$ is universally quantified, most choices of $|\psi_k\rangle$ will cause a mismatch with X_i with high probability, causing V' to accept with high probability. The crucial insight is that, in order for $|\psi_1\rangle \otimes \cdots \otimes |\psi_k\rangle$ to pass the APT, it must essentially set each copy of $|\psi_i\rangle$ to string y_i^*. Thus, measuring the A register is highly unlikely to produce mismatches on existentially quantified proofs!
2. Measuring all $i \in U$ will yield the desired outcomes y_i^*, since U is existentially quantified. If in addition $\forall j\ X_{ij} = X_i$, running V may nevertheless accept due to imperfect soundness, i.e. $s > 0$.

Then, by a similar argument as for the YES case that in Step 2, in which we first assume the APT passes with certainty, $\Pr[\text{accept}] \leq 2^{-m} + s$. Now, let us assume the APT accepts with probability $\geq 1 - \varepsilon/2mn$. By Lemma 13,

$$\langle\psi_k|(|\psi_1, \ldots, \psi_{k-1}\rangle\langle\psi_1, \ldots, \psi_{k-1}|_A \otimes I_B)|\psi_k\rangle \geq 1 - \varepsilon. \qquad (32)$$

Let $\{|\beta_i\rangle\}$ be an orthonormal basis of register A with $|\beta_1\rangle = |\psi_1, \ldots, \psi_{n-1}\rangle$. Then we can write $|\psi_n\rangle = \sum_i \alpha_i |\beta_i\rangle_A |\gamma_i\rangle_B$ with $|\alpha_1|^2 \geq 1 - \varepsilon$. Hence, V' accepts with probability at most

$$\frac{1}{2}\left(1 - \frac{\varepsilon}{2mn}\right) + \frac{1}{2}\left(\varepsilon + (1-\varepsilon)s\right) \leq 1 - \frac{1}{4mn} + s =: s', \tag{33}$$

where the first inequality follows because, without loss of generality, we may assume $\varepsilon \leq 1/2$, as otherwise the prover cannot hope to succeed make V' accept with probability greater than $3/4$ (whereas $c' \approx 1$). Finally, we can choose c, s, m such that $c' - s' \geq 1/\operatorname{poly}$. ◀

5.2 Upper bound

To complement Section 5.1, we next give a simple but non-trivial upper bound on $\mathsf{pureQPH}$, which may be viewed as an "exponential analogue" of Toda's theorem. For this, let NP^k denote a tower of NP oracles of height k. (For example, $\mathsf{NP}^1 = \mathsf{NP}$ and $\mathsf{NP}^2 = \mathsf{NP}^{\mathsf{NP}}$.) Define NEXP^k analogously, by a tower of NEXP oracles. In [13], it was observed that $\mathsf{Q\Sigma}_i \subseteq \mathsf{NEXP}^i$. We show a sharper bound here.

▶ **Theorem 17.** $\mathsf{pureQPH} \subseteq \mathsf{EXP}^{\mathsf{PP}}$.

▶ **Observation 18.** $\mathsf{pureQ\Sigma}_i \subseteq \mathsf{NEXP}^{\mathsf{NP}^{i-1}}$.

Proof. Replace all proofs by by their exponential-size classical description (up to additive additive inverse exponential additive error in the entries), and simulate the verifier's action on the proofs via exponential-time matrix multiplication. The standard proof technique for showing $\Sigma_i^p \subseteq \mathsf{NP}^i$ now applies, except we only require NEXP at the base level of the oracle tower, i.e. $\mathsf{NEXP}^{\mathsf{NP}^{i-1}}$, since an exponential time base can "inflate" or pad the instance size for its oracle exponentially. ◀

For comparison, the observed bound in [13] of $\mathsf{Q\Sigma}_i \subseteq \mathsf{NEXP}^i$ is overkill, since it allows the first NEXP oracle can use *double* exponential time to process its exponential size input.

▶ **Observation 19.** $\mathsf{NEXP} \subseteq \mathsf{EXP}^{\mathsf{NP}}$.

Proof. Since using an exponential time machine we can "inflate" the instance size to exponential, an NP machine can thereafter simulate the NEXP computation on the inflated instance size. The EXP machine just returns the answer of the NP oracle. ◀

▶ **Observation 20.** $\mathsf{NEXP}^O \subseteq \mathsf{EXP}^{\mathsf{NP}^O}$ *for an oracle to any language O.*

Proof. It is easy to see that the argument in Observation 19 relativizes, since the NP oracle to the EXP machine can make the NEXP queries directly to the oracle O. ◀

▶ **Observation 21.** $\mathsf{EXP}^{\mathsf{P}^{\mathsf{PP}}} \subseteq \mathsf{EXP}^{\mathsf{PP}}$.

Proof. The EXP machine can make at most exponentially many queries to the its oracle, each of which can be of size at most exponential in the size of the input. Therefore an EXP machine can simulate the action of a P^{PP} machine (even on an exponential sized query) by making queries to a PP oracle while simulating the action of a P machine (which will only take time polynomial in size of the query). ◀

▶ **Theorem 22.** *For all $i \geq 1$, $\mathsf{Q\Sigma}_i \subseteq \mathsf{EXP}^{\mathsf{PP}}$.*

Proof. By Observation 18, Observation 20 and Toda's Theorem [29],

$$\mathsf{Q\Sigma}_i \subseteq \mathsf{NEXP}^{\mathsf{NP}^{i-1}} \subseteq \mathsf{EXP}^{\mathsf{NP}^i} \subseteq \mathsf{EXP}^{\mathsf{P}^{\mathsf{PP}}}. \tag{34}$$

The claim now follows from Observation 21. ◀

Theorem 17 now follows immediately from Theorem 22.

References

1 Scott Aaronson and Alex Arkhipov. The Computational Complexity of Linear Optics. In *Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 333–342, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993682.

2 Scott Aaronson, Alexandru Cojocaru, Alexandru Gheorghiu, and Elham Kashefi. On the implausibility of classical client blind quantum computing. *CoRR*, abs/1704.08482, 2017. doi:10.48550/arXiv.1704.08482.

3 Scott Aaronson and Andrew Drucker. A full characterization of quantum advice. *SIAM J. Comput.*, 43(3):1131–1183, 2014. doi:10.1137/110856939.

4 Scott Aaronson, DeVon Ingram, and William Kretschmer. The Acrobatics of BQP. In Shachar Lovett, editor, *37th Computational Complexity Conference (CCC 2022)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2022.20.

5 Avantika Agarwal, Sevag Gharibian, Venkata Koppula, and Dorian Rudolph. Quantum polynomial hierarchies: Karp-lipton, error reduction, and lower bounds, 2024. doi:10.48550/arXiv.2401.01633.

6 Dorit Aharonov, Michael Ben-Or, Fernando G.S.L. Brandão, and Or Sattath. The Pursuit of Uniqueness: Extending Valiant-Vazirani Theorem to the Probabilistic and Quantum Settings. *Quantum*, 6:668, March 2022. doi:10.22331/q-2022-03-17-668.

7 Lennart Bittel, Sevag Gharibian, and Martin Kliesch. The Optimal Depth of Variational Quantum Algorithms Is QCMA-Hard to Approximate. In Amnon Ta-Shma, editor, *38th Computational Complexity Conference (CCC 2023)*, volume 264 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 34:1–34:24, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2023.34.

8 Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. On the complexity and verification of quantum random circuit sampling. *Nature Physics*, 15(2):159–163, February 2019. doi:10.1038/s41567-018-0318-2.

9 Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2126):459–472, August 2010. doi:10.1098/rspa.2010.0301.

10 Chirag Falor, Shu Ge, and Anand Natarajan. A collapsible polynomial hierarchy for promise problems. *CoRR*, abs/2311.12228, 2023. doi:10.48550/arXiv.2311.12228.

11 Merrick Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical systems theory*, 17(1):13–27, December 1984. doi:10.1007/BF01744431.

12 Sevag Gharibian and Julia Kempe. Hardness of Approximation for Quantum Problems. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 387–398, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-31594-7_33.

13 Sevag Gharibian, Miklos Santha, Jamie Sikora, Aarthi Sundaram, and Justin Yirka. Quantum generalizations of the polynomial hierarchy with applications to QMA(2). *computational complexity*, 31(2):13, September 2022. doi:10.1007/s00037-022-00231-8.

14 Oded Goldreich. On Promise Problems: A Survey. In Oded Goldreich, Arnold L. Rosenberg, and Alan L. Selman, editors, *Theoretical Computer Science: Essays in Memory of Shimon Even*, Lecture Notes in Computer Science, pages 254–290. Springer, Berlin, Heidelberg, 2006. doi:10.1007/11685654_12.

15 Sabee Grewal and Justin Yirka. The entangled quantum polynomial hierarchy collapses, 2024. doi:10.48550/arXiv.2401.01453.

16 Aram W. Harrow and Ashley Montanaro. Testing Product States, Quantum Merlin-Arthur Games and Tensor Optimization. *Journal of the ACM*, 60(1):3:1–3:43, February 2013. doi:10.1145/2432622.2432625.

17 Sandy Irani, Anand Natarajan, Chinmay Nirkhe, Sujit Rao, and Henry Yuen. Quantum Search-To-Decision Reductions and the State Synthesis Problem. In Shachar Lovett, editor, *37th Computational Complexity Conference (CCC 2022)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2022.5.

18 J. Lockhart and C. E. González-Guillén. Quantum State Isomorphism. *arXiv preprint arXiv:1709.09622*, 2017. doi:10.48550/arXiv.1709.09622.

19 Rahul Jain and John Watrous. Parallel Approximation of Non-interactive Zero-sum Quantum Games. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 243–253. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.26.

20 Richard M. Karp and Richard J. Lipton. Some Connections Between Nonuniform and Uniform Complexity Classes. In *Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 302–309, New York, NY, USA, 1980. ACM. doi:10.1145/800141.804678.

21 Hirotada Kobayashi, Keiji Matsumoto, and Tomoyuki Yamakami. Quantum Certificate Verification: Single versus Multiple Quantum Certificates, October 2001. doi:10.48550/arXiv.quant-ph/0110006.

22 Hirotada Kobayashi, Keiji Matsumoto, and Tomoyuki Yamakami. Quantum merlin-arthur proof systems: Are multiple merlins more helpful to arthur? *Chic. J. Theor. Comput. Sci.*, 2009, 2009. URL: http://cjtcs.cs.uchicago.edu/articles/2009/3/contents.html.

23 Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, November 1983. doi:10.1016/0020-0190(83)90044-3.

24 Florian Mintert, Marek Kuś, and Andreas Buchleitner. Concurrence of Mixed Multipartite Quantum States. *Physical Review Letters*, 95(26):260502, December 2005. doi:10.1103/PhysRevLett.95.260502.

25 Harumichi Nishimura and Tomoyuki Yamakami. Polynomial time quantum computation with advice. *Information Processing Letters*, 90(4):195–204, May 2004. doi:10.1016/j.ipl.2004.02.005.

26 Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 330–335, New York, NY, USA, December 1983. Association for Computing Machinery. doi:10.1145/800061.808762.

27 Mehdi Soleimanifar and John Wright. Testing matrix product states, January 2022. doi:10.48550/arXiv.2201.01824.

28 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976. doi:10.1016/0304-3975(76)90061-X.

29 Seinosuke Toda. PP is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, October 1991. doi:10.1137/0220053.

30 L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, January 1986. doi:10.1016/0304-3975(86)90135-0.

31 Tomoyuki Yamakami. Quantum NP and a Quantum Hierarchy. In Ricardo Baeza-Yates, Ugo Montanari, and Nicola Santoro, editors, *Foundations of Information Technology in the Era of Network and Mobile Computing: IFIP 17th World Computer Congress — TC1 Stream / 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002) August 25–30, 2002, Montréal, Québec, Canada*, IFIP — The International Federation for Information Processing, pages 323–336. Springer US, Boston, MA, 2002. doi:10.1007/978-0-387-35608-2_27.

Sublinear Time Shortest Path in Expander Graphs

Noga Alon
Princeton University, NJ, USA

Allan Grønlund
Kvantify, Aarhus, Denmark

Søren Fuglede Jørgensen
Kvantify, Aarhus, Denmark

Kasper Green Larsen
Aarhus University, Denmark
Kvantify, Aarhus, Denmark

Abstract

Computing a shortest path between two nodes in an undirected unweighted graph is among the most basic algorithmic tasks. Breadth first search solves this problem in linear time, which is clearly also a lower bound in the worst case. However, several works have shown how to solve this problem in sublinear time in expectation when the input graph is drawn from one of several classes of random graphs. In this work, we extend these results by giving sublinear time shortest path (and short path) algorithms for expander graphs. We thus identify a natural deterministic property of a graph (that is satisfied by typical random regular graphs) which suffices for sublinear time shortest paths. The algorithms are very simple, involving only bidirectional breadth first search and short random walks. We also complement our new algorithms by near-matching lower bounds.

2012 ACM Subject Classification Theory of computation → Shortest paths

Keywords and phrases Shortest Path, Expanders, Breadth First Search, Graph Algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.8

Funding *Noga Alon*: Supported by NSF grant DMS-2154082.
Kasper Green Larsen: Supported by a DFF Sapere Aude Research Leader Grant No. 9064-00068B.

1 Introduction

Computing shortest paths in an undirected unweighted graph is among the most fundamental tasks in graph algorithms. In the single source case, the textbook breadth first search (BFS) algorithm computes such shortest paths in $O(m + n)$ time in a graph with n nodes and m edges. Linear time is clearly also a lower bound on the running time of any algorithm that is correct on all input graphs, even if we only consider computing a shortest s-t path for a pair of nodes s, t, and not a shortest path from s to all other nodes. Initial intuition might also suggest that linear time is necessary for computing a shortest path between two nodes s, t in a random graph drawn from any reasonable distribution, such as an Erdős-Rényi random graph or a random d-regular graph. However, this intuition is incorrect and there exists an algorithm with a sublinear expected running time for many classes of random graphs [6, 10, 18]. Moreover, the algorithm is strikingly simple! It is merely the popular practical heuristic of bidirectional BFS [19]. In bidirectional BFS, one simultaneously runs BFS from the source s and destination t, expanding the two BFS trees by one layer at a time. If the input graph is e.g. an Erdős-Rényi random graph, then it can be shown that the two BFS trees have a node in common after exploring only $O(\sqrt{n})$ nodes in expectation. If the node v is first to be explored in both trees, then the path from $s \to v \to t$ in the two BFS trees form a shortest path between s and t. The fact that only $O(\sqrt{n})$ nodes need to be explored intuitively follows from the birthday paradox and the fact that the nodes nearest

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 8; pp. 8:1–8:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to s and t are uniform random in an Erdős-Rényi random graph (although not completely independent). Note that for sublinear time graph algorithms to be meaningful, we assume that we have random access to the nodes and their neighbors. More concretely, we assume the nodes are indexed by integers $[n] = \{1, \dots, n\}$ and that we can query for the number of nodes adjacent to a node v, as well as query for the j'th neighbor of a node v. We remark that several works have also extended the bidirectional BFS heuristic to weighted input graphs and/or setups where heuristic estimates of distances between nodes and the source or destination are known [19, 20, 12]. There are also works giving sublinear time algorithms for other natural graph problems under the assumption of a random input graph [14].

A caveat of the previous works that give provable sublinear time shortest path algorithms, is that they assume a random input graph. In this work, we identify "deterministic" properties of graphs that may be exploited to obtain sublinear time s-t shortest path algorithms. Concretely, we study shortest paths in expander graphs. An n-node d-regular (all nodes have degree d) graph G, is an (n, d, λ)-graph if the eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$ of the corresponding adjacency matrix A satisfies $\max_{i \neq 1} |\lambda_i| \leq \lambda$. Note that the eigenvalues are real since A is symmetric and real. We start by presenting a number of algorithmic results when the input graph is an expander.

Shortest s-t Path

Our first contribution demonstrates that the simple bidirectional BFS algorithm efficiently computes a shortest path between most pairs of nodes s, t in an expander:

▶ **Theorem 1.** *If G is an (n, d, λ)-graph, then for every node $s \in G$, every $0 < \delta < 1$, it holds for at least $(1 - \delta)n$ nodes t, that bidirectional BFS between s and t, finds a shortest s-t path after visiting $O((d-1)^{\lceil (1/4) \lg_{d/\lambda}(n/\delta) \rceil})$ nodes.*

While the bound in Theorem 1 on the number of nodes visited may appear unwieldy at first, we note that it simplifies significantly for natural values of d and λ. For instance, an (n, d, λ)-graph is Ramanujan if $\lambda \leq 2\sqrt{d-1}$. For Ramanujan graphs, and more generally for graphs with $\lambda = O(\sqrt{d})$, the bound in Theorem 1 simplifies to near-$\sqrt{n}$:

▶ **Corollary 2.** *If G is an $(n, d, O(\sqrt{d}))$-graph, then for every node $s \in G$, every $0 < \delta < 1$, it holds for at least $(1 - \delta)n$ nodes t, that bidirectional BFS between s and t, finds a shortest s-t path after visiting $O((n/\delta)^{1/2 + O(1/\ln d)})$ nodes.*

We also demonstrate that the bound can be tightened even further for Ramanujan graphs:

▶ **Theorem 3.** *If G is a d-regular Ramanujan graph where $d \geq 3$, then for every node $s \in G$, it holds for at least $(1 - o(1))n$ nodes t, that bidirectional BFS between s and t, finds a shortest s-t path after visiting $O(\sqrt{n} \cdot \ln^{3/2}(n))$ nodes.*

Short s-t Path

One drawback of bidirectional BFS in expanders, is that it is only guaranteed to find a shortest path efficiently for *most* pairs of nodes s, t. One can show that this is inherent. In particular, as we sketch in Section 4, for constant d and infinitely many n, there exists $(n, d, 3\sqrt{d})$-graphs with diameter at least $1.998 \lg_{d-1} n$. Picking two nodes s and t of maximum distance in such a graph and running BFS from both will only terminate after having visited $\Omega((d-1)^{(1.998/2) \lg_{d-1} n}) = \Omega(n^{0.999})$ nodes.

Motivated by this shortcoming, we also present a simple randomized algorithm for finding a short, but not necessarily shortest, s-t path. For any parameter $0 < \delta < 1$, the algorithm starts by growing a BFS tree from s until $\Theta(\sqrt{n \ln(1/\delta)})$ nodes have been explored. It then performs $O(\sqrt{n \ln(1/\delta)}/\lg_{d/\lambda}(n))$ random walks starting at t. Each of these random walks run for $O(\lg_{d/\lambda}(n))$ steps. If any of these walks discover a node in the BFS tree, it has found an s-t path of length $O(\lg_{d/\lambda}(n))$.

We show that this BFS + Random Walks algorithm has a high probability of finding an s-t path:

▶ **Theorem 4.** *If G is an (n, d, λ)-graph with $\lambda \leq d/2$, then for every pair of nodes s, t, every $0 < \delta < 1$, it holds with probability at least $1 - \delta$, that BFS + Random Walks between s and t, finds an s-t path of length $O(\lg_{d/\lambda}(n))$ while visiting $O(\sqrt{n \ln(1/\delta)})$ nodes.*

Finally, let us mention the two previous works [9, 7] that have also identified deterministic properties of graphs which suffice for provable speedups from bidirectional BFS. The deterministic properties they identify are vaguely related to expansion, but are not as standard and clean-cut as our results using the standard definition of expanders. The work [16] has also investigated short paths in expanders in the context of multicommodity flow and approximating the maximum number of disjoint paths between pairs of nodes.

Lower Bounds

While bidirectional BFS, or BFS + Random Walks, are natural algorithms for finding s-t paths efficiently, it is not a priori clear that better strategies do not exist. One could e.g. imagine sampling multiple nodes in an input graph, growing multiple small BFS trees from the sampled nodes and somehow use this to speed up the discovery of an s-t path. To rule this approach out, we complement the algorithms presented above with lower bounds. For proving lower bounds, we consider distributions over input graphs and show that any algorithm that explores few nodes fails to find an s-t path with high probability in such a random input graph. As Erdős-Rényi random graphs (with large enough edge probability) and random d-regular graphs are both expanders with good probability, we prove lower bounds for both these random graph models. The distribution of an Erdős-Rényi random graph on n nodes is defined from a parameter $0 < p < 1$. In such a random graph, each edge is present independently with probability p. A random d-regular graph on the other hand, is uniform random among all n-node graphs where every node has degree d.

Our lower bounds hold even for the problem of reporting an arbitrary path connecting a pair of nodes s, t, not just for reporting a short/shortest path. Furthermore, our lower bounds are proved in a model where we allow node-incidence queries. A node-incidence query is specified by a node index v and is returned the set of all edges incident to v. Our first lower bound holds for Erdős-Rényi random graphs:

▶ **Theorem 5.** *Any (possibly randomized) algorithm for reporting an s-t path in an Erdős-Rényi random graph, where edges are present with probability $p \geq 1.5 \ln(n)/n$, either makes $\Omega(1/(p\sqrt{n}))$ node-incidence queries or outputs a valid path with probability at most $o(1) + p$.*

Note that the lower bound assumes $p \geq 1.5 \ln(n)/n$. This is a quite natural assumption since for $p \ll \ln(n)/n$, the input graph is disconnected with good probability. The concrete constant 1.5 is mostly for simplicity of the proof. We remark that the additive p in the success probability is tight as an algorithm always reporting the direct path consisting of the single edge (s, t) is correct with probability p. Also observe that the number of edges discovered after $O(1/(p\sqrt{n}))$ node-incidence queries is about $O(pn/(p\sqrt{n})) = O(\sqrt{n})$ since each node has $p(n-1)$ incident edges in expectation.

For the case of random d-regular graphs, we show the following lower bound for constant degree d:

▶ **Theorem 6.** *Any (possibly randomized) algorithm for reporting an s-t path in a random d-regular graph with $d = O(1)$, either makes $\Omega(\sqrt{n})$ node-incidence queries or outputs a valid path with probability at most $o(1)$.*

We remark that a random d-regular graph is near-Ramanujan with probability $1 - o(1)$ as proved in [13], confirming a conjecture raised in [1]. A near-Ramanujan graph is an (n, d, λ)-expander with $\lambda \leq 2\sqrt{d-1} + o(1)$. Thus our upper bounds in Theorem 1 and Theorem 4 nearly match this lower bound.

Overview

In Section 2, we present our upper bound results and prove the claims in Theorem 1 and Theorem 4. The upper bounds are all simple algorithms and also have simple proofs using well-known facts about expanders.

In Section 3, we prove our lower bounds. These proofs are more involved and constitute the main technical contributions of this work.

2 Upper Bounds

In the following, we present and analyse simple algorithms for various s-t reachability problems in expander graphs.

2.1 Shortest Path

Let G be an (n, d, λ)-graph and consider the following bidirectional BFS algorithm for finding a shortest path between a pair of nodes s, t: grow a BFS tree $\mathcal{T}_s$ from s and a BFS tree $\mathcal{T}_t$ from t simultaneously. In each iteration, the next layer of $\mathcal{T}_s$ and $\mathcal{T}_t$ is computed and as soon as a node v appears in both trees, we have found a shortest path from s to t, namely the path $s \to v \to t$ in the two BFS trees.

We show that this algorithm is efficient for most pairs of nodes s, t as claimed in Theorem 1.

To prove Theorem 1, we show that in any (n, d, λ)-graph G, it holds for every node $s \in G$ that most other nodes have a small distance to s. Concretely, we show the following

▶ **Lemma 7.** *If G is an (n, d, λ)-graph, then for every node $s \in G$, it holds for every $0 < \delta < 1$ that there are no more than δn nodes with distance more than $(1/2)\lg_{d/\lambda}(n/\delta)$ from s.*

Theorem 1 now follows from Lemma 7 by observing that for a pair of nodes s, t of distance k in an (n, d, λ)-graph, the bidirectional searches will meet after expanding for $\lceil k/2 \rceil$ steps from s and t. Since each node explored during breadth first search has at most $d - 1$ neighbors outside the previously explored tree, it follows that the total number of nodes visited is $O((d-1)^{\lceil k/2 \rceil})$. Since it holds for every $s \in G$ that $\operatorname{dist}(s, t) \leq (1/2)\lg_{d/\lambda}(n/\delta)$ for a $1 - \delta$ fraction of all other nodes t, the conclusion follows.

Corollary 2 follows from Theorem 1 by observing that when $\lambda = O(\sqrt{d})$, we have $(1/4)\lg_{d/\lambda}(n/\delta) = (1/2)\lg_{\Omega(d)}(n/\delta)$. Noting that $\lg_{\Omega(d)}(n/\delta) = \ln(n/\delta)/(\ln(d) - O(1)) = (1 + O(1/\ln d))\lg_{d-1}(n/\delta)$, the conclusion follows.

What remains is to prove Lemma 7. While the contents of the lemma is implicit in previous works, we have not been able to find a reference explicitly stating this fact. We thus provide a simple self-contained proof building on Chung's [11] proof that the diameter of an (n, d, λ)-graph is bounded by $\lceil \lg_{d/\lambda} n \rceil$.

Proof of Lemma 7. Let A be the adjacency matrix of an (n, d, λ)-graph G. Letting $d = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ denote the (real-valued) eigenvalues of the real symmetric matrix A, we may write A in its spectral decomposition $A = U\Sigma U^T$ with $\lambda_1, \ldots, \lambda_n$ being the diagonal entries of the diagonal matrix Σ. By definition, we have $\max\{\lambda_2, |\lambda_n|\} = \lambda$.

Notice that $(A^k)_{s,t}$ gives the number of length-k paths from node s to node t in G. Furthermore, we have $A^k = U\Sigma^k U^T$. Now let s be an arbitrary node of G and let $Z \subseteq [n]$ denote the subset of columns t such that $(A^k)_{s,t} = 0$. The eigenvalues of A^k are $\lambda_1^k, \ldots, \lambda_n^k$ and the all-1's vector $\mathbf{1}$ is an eigenvector corresponding to λ_1. Let $\mathbf{1}_Z$ denote the indicator for the set Z, i.e. the coordinates of $\mathbf{1}_Z$ corresponding to $t \in Z$ are 1 and the remaining coordinates are 0. By definition of Z, we have that $e_s^T A^k \mathbf{1}_Z = 0$. At the same time, we may write $\mathbf{1}_Z = (|Z|/n)\mathbf{1} + \beta u$ where u is a unit length vector orthogonal to $\mathbf{1}$ and $\beta = \sqrt{|Z| - |Z|^2/n}$. Hence

$$\begin{aligned} 0 &= e_s^T A^k \mathbf{1}_Z \\ &= e_s^T A^k((|Z|/n)\mathbf{1} + \beta u) \\ &= e_s^T \lambda_1^k (|Z|/n)\mathbf{1} + \beta e_s^T A^k u \\ &\geq d^k|Z|/n - \beta \cdot \|e_s\| \cdot \|A^k u\| \\ &\geq d^k|Z|/n - \beta\lambda^k. \end{aligned}$$

From this we conclude $|Z| \leq (\lambda/d)^k n\beta \leq (\lambda/d)^k n\sqrt{|Z|}$, implying $|Z| \leq (\lambda/d)^{2k} n^2$. For $k = (1/2)\lg_{d/\lambda}(n/\delta)$, this is $|Z| \leq \delta n$. ◀

For the special case of Ramanujan graphs, Theorem 3 claims an even stronger result than Theorem 1. Recall that an (n, d, λ)-graph is Ramanujan if it satisfies that $\lambda \leq 2\sqrt{d-1}$. To prove Theorem 3 we make use of the following concentration result on distances in Ramanujan graphs:

▶ **Theorem 8** ([17]). *Let G be a d-regular Ramanujan graph on n nodes, where $d \geq 3$. Then for every node $s \in G$ it holds that*

$$|\{t \in G : |\mathrm{dist}(s,t) - \lg_{d-1} n| > 3\lg_{d-1}\lg n\}| = o(n).$$

Using Theorem 8, we conclude that for every node $s \in G$, it holds for $(1 - o(1))n$ choices of t that $\mathrm{dist}(s,t) \leq \lg_{d-1} n + 3\lg_{d-1}\lg n$. The middle node v on a shortest path from s to t thus has distance at most $k = \lceil(\lg_{d-1} n + 3\lg_{d-1}\lg n)/2\rceil \leq (1/2)\lg_{d-1} n + (3/2)\lg_{d-1}\lg n + 1$ from s and t. Since the nodes in a layer ℓ of a BFS tree in a d-regular graph G has at most $d-1$ neighbors in layer $\ell + 1$, we conclude that the two BFS trees $\mathcal{T}_s$ and $\mathcal{T}_t$ contain at most $O((d-1)^k) \leq O(\sqrt{n} \cdot \ln^{3/2}(n))$ nodes each upon termination. Note that the same proof shows how to find a shortest path in time $n^{1/2+o(1)}$ between most pairs of vertices s and t in near Ramanujan graphs, as it is also proved in [17] that in such graphs, for every node s there are only $o(n)$ nodes t of distance exceeding $(1 + o(1))\lg_{d-1} n$ from s.

2.2 Connecting Path

In the following, we analyse our algorithm, BFS + Random Walks, for finding a short s-t path in an (n, d, λ)-graph. The algorithm is parameterised by an integer $k \geq \sqrt{n}$ and is as follows: First, run BFS from s until k nodes have been discovered. Call the set of discovered nodes V_s. Next, run $\tau = k/(3\lg_{d/\lambda}(n))$ random walks $\mathbf{p}_1, \ldots, \mathbf{p}_\tau$ from t, with each random walk having a length of $3\lg_{d/\lambda}(n)$. If any of the random walks intersects V_s, we have found an s-t path of length $O(\lg_{d/\lambda}(n))$ as the paths $\mathbf{p}_i$ have length $O(\lg_{d/\lambda}(n))$ and the diameter, and hence the depth of the BFS tree, in an (n, d, λ)-graph is at most $\lceil\lg_{d/\lambda}(n)\rceil$ [11].

To analyse the success probability of the algorithm, we bound the probability that all paths $\mathbf{p}_i$ avoid V_s. For this, we use the following two results

▶ **Theorem 9** ([15]). *Let G be an (n, d, λ)-graph. For any two nodes s, t in G, the probability $p^k_{s,t}$ that a random walk starting in s and of length k ends in the node t, satisfies $|1/n - p^k_{s,t}| \leq (\lambda/d)^k$.*

▶ **Theorem 10** ([3]). *Let G be an (n, d, λ)-graph and let W be a set of w vertices in G and set $\mu = w/n$. Let $P(W, k)$ be the total number of length k paths ($k+1$ nodes) that stay in W. Then*

$$P(W, k) \leq wd^k(\mu + (\lambda/d)(1-\mu))^k.$$

Now consider one of the length $3\lg_{d/\lambda}(n)$ random walks $\mathbf{p} = \mathbf{p}_i$ starting in t. To show that it is likely that the path intersects V_s, we split the random walk $\mathbf{p} = (t, \mathbf{v}_1, \dots, \mathbf{v}_{3\lg_{d/\lambda}(n)+1})$ into two parts, namely the first $2\lg_{d/\lambda}(n)$ steps $\mathbf{p}^{(1)} = (t, \mathbf{v}_1, \dots, \mathbf{v}_{2\lg_{d/\lambda}(n)+1})$ and the remaining $\lg_{d/\lambda}(n)$ steps $\mathbf{p}^{(2)} = (\mathbf{v}_{2\lg_{d/\lambda}(n)+1}, \dots, \mathbf{v}_{3\lg_{d/\lambda}(n)+1})$. Note that we let the last node $e(\mathbf{p}^{(1)}) = \mathbf{v}_{2\lg_{d/\lambda}(n)+1}$ in $\mathbf{p}^{(1)}$ equal the first node $s(\mathbf{p}^{(2)}) = \mathbf{v}_{2\lg_{d/\lambda}(n)+1}$ in $\mathbf{p}^{(2)}$. We use $\mathbf{p}^{(1)}$ to argue that $\mathbf{p}^{(2)}$ has a near-uniform random starting node. We then argue that $\mathbf{p}^{(2)}$ intersects V_s with good probability.

By Theorem 9, it holds for any node $r \in G$ that $\Pr[e(\mathbf{p}^{(1)}) = r] \leq 1/n + 1/n^2$. Next, conditioned on $e(\mathbf{p}^{(1)}) = r$, the path $\mathbf{p}^{(2)}$ is uniform random among the $d^{\lg_{d/\lambda}(n)}$ length $\lg_{d/\lambda}(n)$ paths starting in r. It follows that for any fixed path p of length $\lg_{d/\lambda}(n)$ in G, we have $\Pr[\mathbf{p}^{(2)} = p] \leq \Pr[e(\mathbf{p}^{(1)}) = s(p)]d^{-\lg_{d/\lambda}(n)} \leq (1/n + 1/n^2)d^{-\lg_{d/\lambda}(n)}$. Now by Theorem 10 with $W = V(G) \setminus V_s$ and assuming $\lambda \leq d/2$, there are at most $nd^{\lg_{d/\lambda}(n)}((1 - k/n) + (\lambda/d)(k/n))^{\lg_{d/\lambda}(n)} \leq nd^{\lg_{d/\lambda}(n)}(1-k/(2n))^{\lg_{d/\lambda}(n)} \leq nd^{\lg_{d/\lambda}(n)} \exp(-\lg_{d/\lambda}(n)k/(2n))$ paths in G that stay within $V(G) \setminus V_s$. A union bound over all of them implies that the probability that $\mathbf{p}^{(2)}$ avoids V_s is at most

$$(1/n + 1/n^2)d^{-\lg_{d/\lambda}(n)} nd^{\lg_{d/\lambda}(n)} \exp(-\lg_{d/\lambda}(n)k/(2n)) \leq \exp(-\lg_{d/\lambda}(n)k/(2n) + 1/n).$$

Since the $\tau = k/(3\lg_{d/\lambda}(n))$ random walks $\mathbf{p}_1, \dots, \mathbf{p}_\tau$ are independent, we conclude that the probability they all avoid V_s is no more than

$$\exp(-k^2/(6n) + k/(3\lg_{d/\lambda}(n)n)).$$

Letting $k = \sqrt{7n\ln(1/\delta)}$ and assuming n is at least some sufficiently large constant, we have that at least one path $\mathbf{p}_i$ intersects V_s with probability at least $1 - \delta$. This completes the proof of Theorem 4.

3 Lower Bounds

In this section, we prove lower bounds on the number of queries made by any algorithm for computing an s-t path in a random graph. Our query model allows *node-incidence queries*. Here the n nodes of a graph G are assumed to be labeled by the integers $[n]$. A node-incidence query is specified by a node index $i \in [n]$, and the query algorithm is returned the list of edges (i, j) incident to i.

We start by considering an Erdős-Rényi random graph, as it is the simplest to analyse. We then proceed to random d-regular graphs. For the lower bounds, the task is to output a path between nodes $s = 1$ and $t = n$. An algorithm for finding an s-t path works as follows: In each step, the algorithm is allowed to ask one node-incidence query. We make no assumption about how the algorithm determines which query to make in each step, other than it being computable from all edges seen so far (the responses to the node-incidence queries). For randomized algorithms, the choice of query in each step is chosen randomly from a distribution over queries computable from all edges seen so far.

3.1 Erdős-Rényi

Let $\mathbf{G}$ be an Erdős-Rényi random graph, where each edge is present independently with probability $p \geq 1.5\ln(n)/n$ and let $\mathcal{A}^\star$ be a possibly randomized algorithm for computing an s-t path in $\mathbf{G}$ when $s = 1$ and $t = n$. Let $\alpha^\star$ be the probability that $\mathcal{A}^\star$ outputs a *valid* s-t path (all edges on the reported path are in $\mathbf{G}$) and let q be the worst case number of queries made by $\mathcal{A}^\star$ (for $\mathcal{A}^\star$ making an expected q queries, we can always make it worst case $O(q)$ queries by decreasing α by a small additive constant). Here the probability is over both the random choices of $\mathcal{A}^\star$ and the random input graph $\mathbf{G}$. By linearity of expectation, we may fix the random choices of $\mathcal{A}^\star$ to obtain a deterministic algorithm $\mathcal{A}$ that outputs a valid s-t path with probability $\alpha \geq \alpha^\star$. It thus suffices to prove an upper bound on α for such deterministic $\mathcal{A}$.

For a graph G, let $\pi(G)$ denote the *trace* of running the deterministic $\mathcal{A}$ on G. If $i_1(G), \ldots, i_q(G)$ denotes the sequence of queries made by $\mathcal{A}$ on G and $\mathcal{N}_1(G), \ldots, \mathcal{N}_q(G)$ denotes the returned sets of edges, then

$$\pi(G) := (i_1(G), \mathcal{N}_1(G), i_2(G), \ldots, i_q(G), \mathcal{N}_q(G)).$$

Observe that if we condition on a particular trace $\tau = (i_1, N_1, i_2, \ldots, i_q, N_q)$, then the distribution of $\mathbf{G}$ conditioned on $\pi(\mathcal{A}, \mathbf{G}) = \tau$ is the same as if we condition on the set of edges incident to $i_1, \ldots, i_q$ being precisely $N_1, \ldots, N_q$. This is because the algorithm $\mathcal{A}$ is deterministic and the execution of $\mathcal{A}$ is the same for all graphs G with the same such sets of edges incident to $i_1, \ldots, i_q$. Furthermore, no graph G with a different set of incident edges for $i_1, \ldots, i_q$ will result in the trace τ.

For a trace $\tau = (i_1, N_1, \ldots, i_q, N_q)$, call the trace *connected* if there is a path from s to t using the *discovered* edges

$$\bigcup_{j=1}^{q} N_j.$$

Otherwise, call it *disconnected.* Intuitively, if a trace is disconnected, then it is unlikely that $\mathcal{A}$ will succeed in outputting a valid path connecting s and t as it has to guess some of the edges along such a path. Furthermore, if $\mathcal{A}$ makes too few queries, then it is unlikely that the trace is connected. Letting $\mathcal{A}(G)$ denote the output of $\mathcal{A}$ on the graph G, we have for a random graph $\mathbf{G}$ that

$$\alpha = \Pr[\mathcal{A}(\mathbf{G}) \text{ is valid}] \leq \Pr[\pi(\mathbf{G}) \text{ is connected}] + \Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathbf{G}) \text{ is disconnected}].$$

We now bound the two quantities on the right hand side separately.

The simplest term to bound is

$$\Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathcal{A}, \mathbf{G}) \text{ is disconnected}].$$

For this, let $\tau = (i_1, N_1, \ldots, i_q, N_q)$ be an arbitrary disconnected trace in the support of $\pi(\mathbf{G})$ when $\mathbf{G}$ is an Erdős-Rényi random graph, where each edge is present with probability $p \geq 1.5\ln(n)/n$. Observe that the output of $\mathcal{A}$ is determined from τ. Since τ is disconnected, the path reported by $\mathcal{A}$ on τ must contain at least one edge (u, v) where neither u nor v is among $\cup_j\{i_j\}$ or otherwise the output path is valid with probability 0 conditioned on τ. But conditioned on the trace τ, every edge that is not connected to $\{i_1, \dots, i_q\}$ is present independently with probability p. We thus conclude

$$\Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathbf{G}) = \tau] \le p.$$

Since this holds for every disconnected τ, we conclude

$$\Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathbf{G}) \text{ is disconnected}] \le p.$$

Next we bound the probability that $\pi(\mathbf{G})$ is connected. For this, define for $1 \le k \le q$

$$\pi_k(G) := (i_1(G), \mathcal{N}_1(G), i_2(G), \dots, i_k(G), \mathcal{N}_k(G))$$

as the trace of $\mathcal{A}$ on G after the first k queries. As for $\pi(G)$, we say that $\pi_k(G)$ is connected if there is a path from s to t using the discovered edges

$$E(\pi_k(G)) = \bigcup_{j=1}^{k} \mathcal{N}_j(G)$$

and that it is disconnected otherwise. We further say that $\pi_k(G)$ is *useless* if it is both disconnected and $|E(\pi_k(G))| \le 2pnk$. Since

$$\Pr[\pi_k(\mathbf{G}) \text{ is disconnected}] \ge \Pr[\pi_k(\mathbf{G}) \text{ is useless}]$$

we focus on proving that $\Pr[\pi_k(\mathbf{G})$ is useless$]$ is large. For this, we lower bound

$$\Pr[\pi_k(\mathbf{G}) \text{ is useless} \mid \pi_{k-1}(\mathbf{G}) \text{ is useless}].$$

Note that the base case $\pi_0(\mathbf{G})$ is defined to be useless as s and t are not connected when no queries have been asked and also $|E(\pi_0(G))| = 0 \le 2pn0 = 0$. Let $\tau_{k-1} = (i_1, N_1, \dots, i_{k-1}, N_{k-1})$ be any useless trace. The query $i_k = i_k(\mathbf{G})$ is uniquely determined when conditioning on $\pi_{k-1}(\mathbf{G}) = \tau_{k-1}$ and so is the edge set $E_{k-1} = E(\pi_{k-1}(\mathbf{G}))$. Furthermore, we know that $|E_{k-1}| \le 2pn(k-1)$. We now bound the probability that the query i_k discovers more than $2pn$ new edges. If i_k has already been queried, no new edges are discovered and the probability is 0. So assume $i_k \notin \{i_1, \dots, i_{k-1}\}$. Now observe that conditioned on $\pi_{k-1}(\mathbf{G}) = \tau_{k-1}$, the edges (i_k, i) where $i \notin \{i_1, \dots, i_{k-1}\}$ are independently included in $\mathbf{G}$ with probability p each. The number of new edges discovered is thus a sum of $m \le n$ independent Bernoullis $\mathbf{X}_1, \dots, \mathbf{X}_m$ with success probability p. A Chernoff bound implies $\Pr[\sum_i \mathbf{X}_i > (1+\delta)\mu] < (e^\delta/(1+\delta)^{1+\delta})^\mu$ for any $\mu \ge mp$ and any $\delta > 0$. Letting $\mu = np$ and $\delta = 1$ gives

$$\Pr[\sum_i \mathbf{X}_i > 2np] < (e/4)^{np} < e^{-np/3}.$$

Since we assume $p > 1.5\ln(n)/n$, this is at most $1/\sqrt{n}$.

We next bound the probability that the discovered edges $\mathcal{N}_k(\mathbf{G})$ makes s and t connected in $E(\pi_k(\mathbf{G}))$. For this, let V_s denote the nodes in the connected component of s in the subgraph induced by the edges E_{k-1}. Define V_t similarly. We split the analysis into three cases. First, if $i_k \in V_s$, then $\mathcal{N}_k(\mathbf{G})$ connects s and t if and only if one of the edges $\{i_k\} \times V_t$ is in $\mathbf{G}$. Conditioned on $\pi_{k-1}(\mathbf{G}) = \tau_{k-1}$, each such edge is in $\mathbf{G}$ independently either with probability 0, or with probability p (depending on whether one of the end points is in $\{i_1, \dots, i_{k-1}\}$). A union bound implies that s and t are connected in $E(\pi_k(\mathbf{G}))$ with probability at most $p|V_t|$. A symmetric argument upper bounds the probability by $p|V_s|$ in case $i_k \in V_t$. Finally, if i_k is in neither of V_s and V_t, it must have an edge to both a node in V_s and in V_t to connect s and t. By independence, this happens with probability at most $p^2|V_t||V_s|$. We thus conclude that

$$\Pr[\pi_k(\mathbf{G}) \text{ is connected} \mid \pi_{k-1}(\mathbf{G}) = \tau_{k-1}] \leq p \max\{|V_s|, |V_t|\} \leq p(|E_{k-1}| + 1) \leq 2p^2nk.$$

A union bound implies

$$\Pr[\pi_k(\mathbf{G}) \text{ is useless} \mid \pi_{k-1}(\mathbf{G}) \text{ is useless}] \geq 1 - 2p^2nk - 1/\sqrt{n}.$$

This finally implies

$$\begin{aligned}
\Pr[\pi(\mathbf{G}) \text{ is useless}] &= \prod_{k=1}^{q} \Pr[\pi_k(\mathbf{G}) \text{ is useless} \mid \pi_{k-1}(\mathbf{G}) \text{ is useless}] \\
&\geq \prod_{k=1}^{q} \left(1 - 2p^2nk - 1/\sqrt{n}\right) \\
&\geq 1 - \sum_{k=1}^{q} (2p^2nk + 1/\sqrt{n}) \\
&\geq 1 - p^2n(q+1)^2 - q/\sqrt{n}.
\end{aligned}$$

It follows that

$$\begin{aligned}
\Pr[\pi(\mathbf{G}) \text{ is connected}] &= 1 - \Pr[\pi(\mathbf{G}) \text{ is disconnected}] \\
&\leq 1 - \Pr[\pi(\mathbf{G}) \text{ is useless}] \\
&\leq p^2n(q+1)^2 + q/\sqrt{n}.
\end{aligned}$$

For $q = o(1/(p\sqrt{n}))$ and $p \geq 1.5 \ln(n)/n$, this is $o(1)$. Note that for the lower bound to be meaningful, we need $p = O(1/\sqrt{n})$ as otherwise the bound on q is less than 1. (Indeed, for $p = \Omega(1/\sqrt{n})$, s and t have a common neighbor with probability bounded away from 0 and if so 2 queries suffice). This concludes the proof of Theorem 5.

3.2 d-Regular Graphs

We now proceed to random d-regular graphs. Assume dn is even, as otherwise a d-regular graph on n nodes does not exist. Similarly to our proof for the Erdős-Rényi random graphs, we will condition on a trace of $\mathcal{A}$. Unfortunately, the resulting conditional distribution of a random d-regular graph is more cumbersome to analyse. We thus start by reducing to a slightly different problem.

Let $\mathcal{M}_{n,d}$ denote the set of all graphs on nd nodes where the edges form a perfect matching on the nodes. There are thus $nd/2$ edges in any such graph. We think of the nodes of a graph $G \in \mathcal{M}_{n,d}$ as partitioned into n groups of d nodes each, and we index the nodes by integer pairs (i, j) with $i \in [n]$ and $j \in [d]$. Here i denotes the index of the group. For a graph $G \in \mathcal{M}_{n,d}$ and a sequence of group indices $p := s, i_1, \dots, i_m, t$, we say that p is a valid s-t *meta-path* in G, if for every two consecutive indices a, b in p, there is at least one edge $((a, j_1), (b, j_2))$ in G. A meta-path is thus a valid path if and only if s and t are connected in the graph resulting from contracting the nodes in each group.

Now consider the problem of finding a valid s-t meta-path in a graph $\mathbf{G}$ drawn uniformly from $\mathcal{M}_{n,d}$ (we write $\mathbf{G} \sim \mathcal{M}_{n,d}$ to denote such a graph) while asking *group-incidence queries*. A group-incidence query is specified by a group index $i \in [n]$ and the answer to the query is the set of edges incident to the nodes $\{i\} \times \{1, \ldots, d\}$.

We start by showing that an algorithm $\mathcal{A}^\star$ for finding an s-t path in a random d-regular n-node graph, gives an algorithm $\mathcal{A}$ for finding an s-t meta-path in a random $\mathbf{G} \sim \mathcal{M}_{n,d}$ using group-incidence queries.

▶ **Lemma 11.** *If there is a (possibly randomized) algorithm $\mathcal{A}^\star$ that reports a valid s-t path with probability α in a random d-regular graph on n nodes while making q node-incidence queries, then there is a deterministic algorithm $\mathcal{A}$ that reports a valid s-t meta-path with probability at least* $\exp(-O(d^2))\alpha$ *in a random graph $\mathbf{G} \sim \mathcal{M}_{n,d}$ while making q group-incidence queries.*

Proof. Given an algorithm $\mathcal{A}^\star$ that reports a valid s-t path in a random d-regular graph on n nodes with probability α, we start by fixing its randomness to obtain a deterministic algorithm $\mathcal{A}'$ with the same number of queries that outputs a valid s-t path with probability at least α. Next, let $\mathbf{G} \sim \mathcal{M}_{n,d}$. Let $i_1 \in [n]$ be the first node that $\mathcal{A}'$ queries (which is independent of the input graph). Our claimed algorithm $\mathcal{A}$ for reporting an s-t meta-path in $\mathbf{G}$ starts by querying the group i_1. Upon being returned the set of edges $\{((i_1, 1), (j_1, k_1)), \ldots, ((i_1, d), (j_d, k_d))\}$ incident to $\{i_1\} \times \{1, \ldots, d\}$, we *contract* the groups such that each edge $((i_1, h), (j, k))$ is replaced by (i_1, j). If this creates any duplicate edges or self-edges, $\mathcal{A}$ aborts and outputs an arbitrarily chosen s-t meta-path. Otherwise, the resulting set of edges $\{(i_1, j_1), \ldots, (i_1, j_d)\}$ is passed on to $\mathcal{A}'$ as the response to the first query i_1. The next query i_2 of $\mathcal{A}'$ is then determined and we again ask it as a group-incidence query on $\mathbf{G}$ and proceed by contracting groups in the returned set of edges and passing the result to $\mathcal{A}'$ if there are no duplicate or self-edges. Finally, if we succeed in processing all q queries of $\mathcal{A}'$ without encountering duplicate or self-edges, $\mathcal{A}$ outputs the s-t path reported by $\mathcal{A}'$ as the s-t meta-path.

To see that this strategy has the claimed probability of reporting a valid s-t meta-path, let $\mathbf{G}^\star$ be the graph obtained from $\mathbf{G}$ by contracting *all* groups. Observe that if we condition on $\mathbf{G}^\star$ being a simple graph (no duplicate edges or self-edges), then the conditional distribution of $\mathbf{G}^\star$ is precisely that of a random d-regular graph on n nodes. It is well-known [5, 8, 22, 21] that the contracted graph $\mathbf{G}^\star$ is indeed simple with probability at least $\exp(-O(d^2))$ and the claim follows. ◀

In light of Lemma 11, we thus set out to prove lower bounds for deterministic algorithms that report an s-t meta-path in a random $\mathbf{G} \sim \mathcal{M}_{n,d}$ using group-incidence queries.

Let $\mathcal{A}$ be a deterministic algorithm making q group-incidence queries that reports a valid s-t meta-path with probability α in a random $\mathbf{G} \sim \mathcal{M}_{n,d}$. Similarly to our proof for Erdős-Rényi graphs, we start by defining the trace of $\mathcal{A}$ on a graph $G \in \mathcal{M}_{n,d}$. If $i_1(G), \ldots, i_q(G) \in [n]$ denotes the sequence of group-incidence queries made by $\mathcal{A}$ on G and $\mathcal{N}_1(G), \ldots, \mathcal{N}_q(G)$ denotes the returned sets of edges, then for $1 \le k \le q$, we define

$$\pi_k(G) = (i_1(G), \mathcal{N}_1(G), \ldots, i_k(G), \mathcal{N}_k(G)).$$

We also let $\pi(G) := \pi_q(G)$ denote the full trace. Call a trace $\tau_k = (i_1, N_1, \ldots, i_k, N_k)$ connected if there is a sequence of group indices $p := s, i_1, \ldots, i_m, t$ such that for every two consecutive indices a, b in p, there is an edge $((a, h), (b, k))$ in $\cup_i N_i$. Otherwise, call the trace disconnected. Letting $\mathcal{A}(G)$ denote the output of $\mathcal{A}$ on the graph G, we have

$$\alpha = \Pr[\mathcal{A}(\mathbf{G}) \text{ is valid}] \le \Pr[\pi(\mathbf{G}) \text{ is connected}] + \Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathbf{G}) \text{ is disconnected}].$$

We bound the two terms separately, starting with the latter. So let $\tau = (i_1, N_1, \ldots, i_q, N_q)$ be a disconnected trace in the support of $\pi(\mathbf{G})$. The output meta-path $\mathcal{A}(\mathbf{G}) = p = s, i_1, \ldots, i_m, t$ of $\mathcal{A}$ is determined from τ. Since τ is disconnected, there must be a pair of consecutive indices a, b in p such that there is no edge $((a,h),(b,k)) \in \cup_i N_i$. Fix such a pair a, b. We now consider two cases. First, if either a or b is among $i_1, \ldots, i_q$, then all edges incident to that group are among $\cup_i N_i$ conditioned on $\pi(\mathbf{G}) = \tau$. It thus follows that p is a valid s-t meta-path with probability 0 conditioned on $\pi(\mathbf{G}) = \tau$. Otherwise, neither of a and b are among $i_1, \ldots, i_q$. The set of edges $\cup_i N_i$ specify at most dq edges of the matching $\mathbf{G}$. For any node whose matching edge is not specified by $\cup_i N_i$, the conditional distribution of its neighbor is uniform random among all other nodes whose matching edge is not in $\cup_i N_i$. For each of the d^2 possible edges $((a,h),(b,k))$ between the groups a and b, there is thus a probability at most $1/(nd - 1 - 2dq)$ that the edge is in $\mathbf{G}$ conditioned on $\pi(\mathbf{G}) = \tau$. A union bound over all d^2 such edges finally implies

$$\Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathbf{G}) = \tau] \leq \frac{d^2}{nd - 1 - 2dq}.$$

Since this holds for every disconnected τ, we conclude

$$\Pr[\mathcal{A}(\mathbf{G}) \text{ is valid} \mid \pi(\mathbf{G}) \text{ is disconnected}] \leq \frac{d^2}{nd - 1 - 2dq}.$$

Next, to bound $\Pr[\pi(\mathbf{G}) \text{ is connected}]$, we show that

$$\Pr[\pi_k(\mathbf{G}) \text{ is disconnected} \mid \pi_{k-1}(\mathbf{G}) \text{ is disconnected}]$$

is large. So let $\tau_{k-1} = (i_1, N_1, \ldots, i_{k-1}, N_{k-1})$ be a disconnected trace in the support of $\pi_{k-1}(\mathbf{G})$. The next query $i_k = i_k(\mathbf{G})$ of $\mathcal{A}$ is fixed conditioned on $\pi_{k-1}(\mathbf{G}) = \tau_{k-1}$. We have a two cases. First, if $i_k \in \{i_1, \ldots, i_{k-1}\}$ then no new edges are returned by the query and we conclude

$$\Pr[\pi_k(\mathbf{G}) \text{ is disconnected} \mid \pi_{k-1}(\mathbf{G}) = \tau_{k-1}] = 1.$$

Otherwise, let V_s denote the subset of group-indices j for which there is a meta-path from s to j. Similarly, let V_t denote the subset of group-indices j for which there is a meta-path from t to j. We have $V_s \cap V_t = \emptyset$. Now if $i_k \in V_s$, we have that $\pi_k(\mathbf{G})$ is connected only if there is an edge between a node (i_k, j) with $j \in [d]$ and a node (b, k) with $b \in V_t$. Let $r \in \{0, \ldots, d\}$ denote the number of nodes (i_k, j) with $j \in [d]$ for which the corresponding matching edge is not in $\cup_i N_i$. Conditioned on $\pi_{k-1}(\mathbf{G}) = \tau_{k-1}$, the neighbor of any such node is uniform random among all other nodes for which the corresponding matching edge is not in $\cup_i N_i$. There are at least $nd - 1 - 2d(k-1)$ such nodes. A union bound over at most $rd|V_t| \leq d^2|V_t|$ pairs $((i_k, j),(b,k))$ implies that $\pi_k(\mathbf{G})$ is connected with probability at most $d^2|V_t|/(nd - 1 - 2d(k-1))$. A symmetric arguments gives an upper bound of $d^2|V_s|/(nd - 1 - 2d(k-1))$ in case $i_k \in V_t$. Finally, if i_k is in neither of V_s and V_t, then there must still be an edge $((i_k, j),(a,k))$ for a group $a \in V_s$. We thus conclude

$$\Pr[\pi_k(\mathbf{G}) \text{ is connected} \mid \pi_{k-1}(\mathbf{G}) = \tau_{k-1}] \leq \frac{d^2 \max\{|V_s|, |V_t|\}}{nd - 1 - 2d(k-1)} \leq \frac{d^3 k}{nd - 1 - 2dq}.$$

Since this holds for every disconnected trace τ_{k-1}, we finally conclude

$$\begin{aligned}\Pr[\pi(\mathbf{G}) \text{ is disconnected}] &\geq \prod_{k=1}^{q}\left(1-\frac{d^3k}{nd-1-2dq}\right)\\ &\geq 1-\sum_{k=1}^{q}\frac{d^3k}{nd-1-2dq}\\ &\geq 1-\frac{d^3q^2}{nd-1-2dq},\end{aligned}$$

and thus

$$\Pr[\pi(\mathbf{G}) \text{ is connected}] \leq \frac{d^3q^2}{nd-1-2dq}.$$

For constant degree d, if $q = o(\sqrt{n})$, this is $o(1)$. Together with Lemma 11, we have thus proved Theorem 6.

4 Large Diameter Expanders

In this section, we sketch the claim from Section 1 that there exists large diameter expanders. Concretely, using the techniques in [4] with a slightly different choice of parameters it is not difficult to show that there are (n', d, λ)-graphs with $\lambda < 3\sqrt{d}$ and diameter larger than $(2 - 0.003) \lg_{d-1} n'$ for constant d. Here is a sketch of the argument proving this fact.

Start with a Ramanujan $(n, d, 2\sqrt{d-1})$-graph, with girth $\Omega(\lg_{d-1} n)$ (for example, taking an LPS expander). Choose in it a set X of $2(d-1)^{0.999 \lg_{d-1} n}$ vertices so that the distance between any pair of them is $\Omega(\lg_{d-1} n)$. This can be done by choosing the vertices one by one, always adding a vertex far from all vertices chosen already. Omit these vertices and identify their $2d(d-1)^{0.999 \lg_{d-1} n}$ neighbors with the leaves of two d-regular trees, each of depth $0.999 \lg_{d-1} n$ and each having $d(d-1)^{0.999 \lg_{d-1} n}$ leaves. The graph obtained is d-regular and has n' vertices (the original n plus the vertices of the two trees). The distance between the roots of the two trees is clearly bigger than $(2 - 0.002) \lg_{d-1} n > (2 - 0.003) \lg_{d-1} n'$.

By the argument in [4] (see also [2], Lemma 3.2) based on the delocalization of eigenvectors of high girth graphs it is not difficult to show that the absolute value of every nontrivial eigenvalue of the graph obtained is smaller than $3\sqrt{d}$, implying the required fact. We omit the detailed computation.

References

1 Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

2 Noga Alon. Explicit expanders of every degree and size. *Combinatorica*, 41, February 2021.

3 Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Comput. Complex.*, 5(1):60–75, January 1995.

4 Noga Alon, Shirshendu Ganguly, and Nikhil Srivastava. High-girth near-ramanujan graphs with localized eigenvectors. *Israel Journal of Mathematics*, 246(1), 2021.

5 Edward A. Bender. The asymptotic number of non-negative integer matrices with given row and column sums. *Discret. Math.*, 10:217–223, 1974.

6 Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. Efficient shortest paths in scale-free networks with underlying hyperbolic geometry. *ACM Trans. Algorithms*, 18(2), March 2022. doi:10.1145/3516483.

7 Thomas Bläsius and Marcus Wilhelm. Deterministic performance guarantees for bidirectional bfs on real-world networks. In *Combinatorial Algorithms: 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7–10, 2023, Proceedings*, pages 99–110. Springer-Verlag, 2023.

8 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1:311–316, 1980.

9 Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. An axiomatic and an average-case analysis of algorithms and heuristics for metric properties of graphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 920–939. Society for Industrial and Applied Mathematics, 2017.

10 Michele Borassi and Emanuele Natale. Kadabra is an adaptive algorithm for betweenness via random approximation. *ACM J. Exp. Algorithmics*, 24, February 2019. `doi:10.1145/3284359`.

11 Fan R. K. Chung. Diameters and eigenvalues. *Journal of the American Mathematical Society*, 2:187–196, 1989.

12 Dennis de Champeaux. Bidirectional heuristic search again. *J. ACM*, 30(1):22–32, January 1983.

13 Joel Friedman. *A Proof of Alon's Second Eigenvalue Conjecture and Related Problems*. American Mathematical Society, 2008.

14 Dorit S. Hochbaum. An exact sublinear algorithm for the max-flow, vertex disjoint paths and communication problems on random graphs. *Operations Research*, 40(5):923–935, 1992.

15 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc.*, 43(04):439–562, August 2006.

16 J. Kleinberg and R. Rubinfeld. Short paths in expander graphs. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, page 86. IEEE Computer Society, 1996.

17 Eyal Lubetzky and Yuval Peres. Cutoff on all ramanujan graphs. *Geometric and Functional Analysis*, 26:1190–1216, 2015.

18 Michael Luby and Prabhakar Ragde. A bidirectional shortest-path algorithm with good average-case behavior. *Algorithmica*, 4(1–4):551–567, March 1989.

19 Ira Sheldon Pohl. *Bi-Directional and Heuristic Search in Path Problems*. PhD thesis, Stanford University, Stanford, CA, USA, 1969.

20 Lenie Sint and Dennis de Champeaux. An improved bidirectional heuristic search algorithm. *J. ACM*, 24(2):177–191, April 1977.

21 N. C. Wormald. *Models of Random Regular Graphs*, pages 239–298. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

22 Nicholas C. Wormald. Some problems in the enumeration of labelled graphs. *Bulletin of the Australian Mathematical Society*, 21(1):159–160, 1980.

Quantum Algorithms for Hopcroft's Problem

Vladimirs Andrejevs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Aleksandrs Belovs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Jevgēnijs Vihrovs
Centre for Quantum Computer Science, Faculty of Computing, University of Latvia, Riga, Latvia

Abstract

In this work we study quantum algorithms for Hopcroft's problem which is a fundamental problem in computational geometry. Given n points and n lines in the plane, the task is to determine whether there is a point-line incidence. The classical complexity of this problem is well-studied, with the best known algorithm running in $O(n^{4/3})$ time, with matching lower bounds in some restricted settings. Our results are two different quantum algorithms with time complexity $\widetilde{O}(n^{5/6})$. The first algorithm is based on partition trees and the quantum backtracking algorithm. The second algorithm uses a quantum walk together with a history-independent dynamic data structure for storing line arrangement which supports efficient point location queries. In the setting where the number of points and lines differ, the quantum walk-based algorithm is asymptotically faster. The quantum speedups for the aforementioned data structures may be useful for other geometric problems.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Computational geometry; Theory of computation → Data structures design and analysis

Keywords and phrases Quantum algorithms, Quantum walks, Computational Geometry

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.9

Related Version *Preprint*: https://arxiv.org/abs/2405.01160

Funding *Aleksandrs Belovs*: Supported by QOPT (QuantERA ERA-NET Cofund).
Jevgēnijs Vihrovs: Supported by Latvian Quantum Initiative under European Union Recovery and Resilience Facility project no. 2.3.1.1.i.0/1/22/I/CFLA/001 and QOPT (QuantERA ERA-NET Cofund).

1 Introduction

In this work we investigate the quantum complexity of Hopcroft's problem, a classic problem in computational geometry. Given n lines and n points in the plane, it asks to determine whether some point lies on some line. In a line of research spanning roughly 40 years culminating with a recent paper by Chan and Zheng [19], the classical complexity has settled on $O(n^{4/3})$ time, with matching lower bounds in some models of computation [27]. Along with its natural setting, the problem also captures the essence of a class of other geometric problems with complexity $\widetilde{O}(n^{4/3})$ [26].

There are several reasons why we find Hopcroft's problem interesting in the quantum setting. Firstly, classical algorithms for this problem typically use data structures supporting some fundamental geometric query operations. For example, Hopcroft's problem can be reduced to the *simplex range searching*, in which the data structure stores the given points and each query asks whether a given region contains any of the points [4]. Another approach is to store the given lines instead and answer *point location queries*, that is, for a given point, determine which region of the line configuration it belongs to [40, 25]. Thus, Hopcroft's problem gives a good playground for improving and comparing the complexity

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 9; pp. 9:1–9:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of ubiquitous geometric data structures quantumly. We are also interested in finding new *history-independent* data structures that can be used in quantum walk algorithms, following the ideas started with Ambainis' element distinctness algorithm [7] (see also [1, 15]).

Secondly, Hopcroft's problem is closely related to a large group of geometric tasks that can be solved in the same time $\widetilde{O}(n^{4/3})$. A speedup for Hopcroft's problem may automatically give an improvement for some of those. Erickson studied the class of such problems [26], some examples include detecting/counting intersections in a set of segments and detecting/counting points in a given set of regions. The problem can be also reduced to various other geometric problems, giving fine-grained lower bounds. For example, Hopcroft's problem in d dimensions (replacing lines with hyperplanes) can be reduced to halfspace range checking in $d+1$ dimensions for $d \geq 4$ (are all given points above all given hyperplanes?) and others [26].

In fact, Hopcroft's problem in d dimensions can also be equivalently formulated as follows: given two sets of vectors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{d+1}$, determine whether there are $a \in \mathcal{A}$, $b \in \mathcal{B}$ such that $\langle a, b\rangle = 0$ [46]. The famous Orthogonal Vectors problem (OV) in fine-grained complexity is a special case of Hopcroft's when $\mathcal{A}, \mathcal{B} = \{0,1\}^d$ [44, 3]. The complexities of these problems differ; if $|A| = |B| = n$, then, classically, the complexity of OV is $\Theta(n)$ in $O(1)$ dimensions [45] and $\Theta(n^{2-o(1)})$ in $\operatorname{polylog} n$ dimensions under SETH [2, 18]. In contrast, the complexity of Hopcroft's problem in d dimensions is $O(n^{2d/(d+1)})$ [19]. Quantumly, the complexity of OV was settled in [1]; for $O(1)$ dimensions, it is $\Theta(\sqrt{n})$ and for $\operatorname{polylog} n$ dimensions, $\Theta(n^{1+o(1)})$ under QSETH, the quantum analogue of SETH. In this work we also examine the quantum complexity of Hopcroft's problem in an arbitrary number of dimensions d.

In general, we are interested in investigating quantum speedups for computational geometry problems. In recent years, there have been several works researching this topic. First, Ambainis and Larka gave a nearly optimal $O(n^{1+o(1)})$ quantum algorithm for the Point-On-3-Lines (detecting whether three lines are concurrent among the n given) and similar problems [11]. This problem is closely connected to fine-grained complexity as well, as it is an instance of the 3-Sum-Hard problem class. Classically, it is conjectured that 3-Sum cannot be solved faster than $O(n^2)$; the authors also conjectured a quantum analogue that 3-Sum cannot be solved quantumly faster than $O(n)$, and Buhrman et al. used this conjecture to prove conditional quantum lower bounds on various geometrical problems [14]. Aaronson et al. studied the quantum complexity of the Closest Pair problem (finding the closest pair of points among the n given), proving an optimal $\widetilde{\Theta}(n^{2/3})$ running time in $O(1)$ dimensions using a quantum walk algorithm with a dynamic history-independent data structure for storing points that is able to detect ϵ-close pairs of points [1]. For Bipartite Closest Pair problem in d dimensions (finding the closest pair of points between two sets of size n), they gave an $O(n^{1-1/2d+\delta})$ time quantum algorithm for any $\delta > 0$. For more results in quantum algorithms for computational geometry, see [39, 38, 12, 42, 43, 31].

For Hopcroft's problem, the best classical results give complexity $O((nm)^{2/3} + m\log n + n\log m)$, where n and m are the number of lines and points, respectively. In d dimensions, these generalize to $O((nm)^{d/(d+1)} + m\log n + n\log m)$ complexity [19]. The first complexity is unconditionally optimal if the algorithm needs to list all incidences, since there exists a planar construction with $\Omega((nm)^{2/3})$ incidence pairs ([25], Section 6.5.). It is also believed to be optimal for detection as well, with matching lower bounds in some models [27]. The dependence on n and m is symmetric since Hopcroft's problem is self-dual, in the sense that there is a geometric transformation which maps lines to points and vice versa, while preserving the point-line incidences ([25], Section 14.3.).

Finally, quite often the quantum query complexity of a problem matches its time complexity, like in Unstructured Search [29, 13], Element Distinctness [9, 8, 32], Closest Pair [1], Claw Finding [41, 47], just to name a few. In other cases even the precise query complexity is not yet clear, for example, Triangle Finding [33] or Boolean Matrix Product Verification [16, 24]. In the case of Hopcroft's problem, its quantum query complexity can be easily characterized to be $\Theta((nm)^{1/3} + \sqrt{n} + \sqrt{m})$ from known results, see Theorem 5. The query-efficient algorithm does not immediately generalize to time complexity; therefore, the main focus here falls to improving the performance of the relevant classical data structures quantumly, which we find interesting.

1.1 Our results

In this work we show two quantum algorithms for Hopcroft's problem with time complexity $\widetilde{O}(n^{5/6})$. This constitutes a polynomial speedup over the classical $O(n^{4/3})$ time. We obtain our results by speeding up classical geometric data structures using different quantum techniques. We look at two underlying fundamental problems one usually encounters on the way to solve Hopcroft's problem.

1. **Simplex range searching.** In simplex range searching, the input is a set of n points in the d-dimensional space. A query then asks whether a given simplex contains any of the given points. The query may also ask to list or count the points in the simplex, among other variants [4]. Usually, there is some *preprocessing time* to precompute the data structure and some *query time* to answer each query. Classically, these complexities are well-understood; in a nutshell, a data structure of size m can be constructed in $\widetilde{O}(m)$ time and each query can then be answered in $\widetilde{O}(n/m^{1/d})$ time [35], and this is matched by lower bounds in the semigroup model [23, 22]. If the allowed memory size is linear, then preprocessing and query times become respectively $O(n \log n)$ and $O(n^{1-1/d})$ [17]. In this paper we require a variant of simplex range queries which we call *hyperplane emptiness queries*, where we have to determine whether a query hyperplane contains any of the given points. We show that quantumly we can speed up query time quadratically by using Montanaro's quantum algorithm for searching in the backtracking trees [36]:

   ▶ **Theorem 1.** *There is a bounded-error quantum algorithm that can answer hyperplane emptiness queries in $\widetilde{O}(\sqrt{n^{1-1/d}})$ time.*

   We note that this result is not really specific to the hyperplane emptiness queries, as all what we are doing is speeding up search in the *partition tree* data structure [17], thus this result can applied to other types of queries as well. However, this speedup does not extend to the counting version of simplex queries, since essentially, our procedure implements a search for a marked vertex in a tree using quantum walk. Using this result, we show a quantum speedup for Hopcroft's problem in d dimensions:

   ▶ **Theorem 2.** *There is a bounded-error quantum algorithm which solves Hopcroft's problem with n hyperplanes and $m \leq n$ points in d dimensions in time:*
   - $\widetilde{O}(n^{\frac{d}{2(d+1)}} m^{1/2})$, *if* $m \geq n^{\frac{d}{d+1}}$;
   - $\widetilde{O}(n^{1/2} m^{\frac{d-1}{2d}})$, *if* $m \leq n^{\frac{d}{d+1}}$.

   If $n = m$, then the algorithm has complexity $\widetilde{O}(n^{1-\frac{1}{2(d+1)}})$.

   In particular, the complexity is $\widetilde{O}(n^{5/6})$ in 2 dimensions for $n = m$.
2. **Planar point location.** The second approach is to use point location queries. One can use usually use classical point location data structures to determine whether a query point lies on the boundary of a planar region it belongs to. More specifically, we consider only *planar* point location data structures in the *line arrangements.* A set of n lines partitions the plane into $O(n^2)$ regions; this an old and well-researched topic, with many approaches to construct a data structure that holds the description of these regions in $O(n^2)$ time, the same amount of space and polylog n point location query time [25] (in fact, $O(n^d)$ preprocessing time and space and $O(\log n)$ query time in d dimensions [21, 20]).

In addition, there are also dynamic data structures with the same preprocessing and query times. More specifically, one can insert or remove a line in time $O(n)$ (or $O(n^{d-1})$ in d dimensions) [37]. We take an opportunity to employ such a data structure in a *quantum walk* algorithm on a Johnson graph to solve Hopcroft's problem. In particular, we develop a history-independent randomized data structure for storing an arrangement of an r-subset of n lines with ability to perform line insertion/removal in $O(r \operatorname{polylog} n)$ time and point location in $\operatorname{polylog} n$ time, requiring $O(r \operatorname{polylog} n)$ memory storage.

To do that, we store k-levels of the line arrangement in the history-independent skip lists a la Ambainis [9]. A k-level of a line arrangement is a set of segments of lines such that there are exactly k lines above each edge. Turns out that skip lists are ideal for encoding the k-levels. For example, when a new line is inserted, it splits each k-level in two parts, one of which will still belong to the k-level, but the other will belong to the $(k+1)$-level. We can then "reglue" these two tails to the correct levels of the arrangements in $\operatorname{polylog} n$ time by utilizing the properties of the skip list, all while keeping the history independence of the data structure. Using this data structure, we show the following quantum speedup for Hopcroft's problem in 2 dimensions:

▶ **Theorem 3.** *There is a bounded-error quantum algorithm that solves Hopcroft's problem with n lines and $m \leq n$ points in the plane in time:*

- $\widetilde{O}(n^{1/3}m^{1/2})$, *if* $n^{2/3} \leq m$;
- $\widetilde{O}(n^{2/5}m^{2/5})$, *if* $n^{1/4} \leq m \leq n^{2/3}$;
- $\widetilde{O}(n^{1/2})$, *if* $m \leq n^{1/4}$.

In particular, the complexity is $\widetilde{O}(n^{5/6})$ when $n = m$.

Both of Theorems 2 and 3 have their pros and cons. Theorem 2 is arguably simpler, since it is a quite direct application of the quantum speedup for backtracking. It also has a lower polylogarithmic factor hidden in the $\widetilde{O}$ notation, only $\log n$ compared to $\log^6 n$ in Theorem 3. However, Theorem 3 gives better asymptotic complexity if the number of lines n differ from the number of points m. On the other hand, Theorem 2 gives a speedup in the case of an arbitrary number of dimensions, while Theorem 3 has something to say only about the planar case; we leave a possible generalization of the quantum walk approach to larger dimensions for future research.

2 Preliminaries

We assume that the sets of points and lines are both in a general position (no two lines are parallel, no three lines intersect at the same point, no three points lie on the same line). Our algorithms work in the standard quantum circuit model augmented with Quantum Random Access Gates that allow to perform read/write operations in superposition; for details, see Appendix A. One of our building blocks is the following version of Grover's search:

▶ **Theorem 4** (Grover's search with bounded-error inputs [5, 30])**.** *Let $\mathcal{A} : [N] \to \{0,1\}$ be a bounded-error quantum procedure with running time T. Then there exists a bounded-error quantum algorithm that computes $\bigvee_{i\in[N]} \mathcal{A}(i)$ with running time $O(\sqrt{N}(T + \log N))$.*

Effectively, this result states that even if the inputs to Grover's search are faulty with constant probability, no error boosting is necessary, which would add another logarithmic factor to the complexity. We say that an algorithm is *bounded-error* if its probability of incorrect output is some constant strictly less than 1/2.

3 Query complexity

Before examining time-efficient quantum algorithms, we take a look at the quantum query complexity of Hopcroft's problem, which in this case can be fully characterized. We assume that with a single query, we can obtain the description of any given line or point. In Appendix B, we prove the following:

▶ **Theorem 5.** *The quantum query complexity of Hopcroft's problem on n lines and m points in two dimensions is $\Theta(n^{1/3}m^{1/3} + \sqrt{n} + \sqrt{m})$.*

In particular, this proves that Theorem 3 is asymptotically optimal for $m \le n^{1/4}$. The query complexity and the complexities of our algorithms are shown graphically in Figure 1.

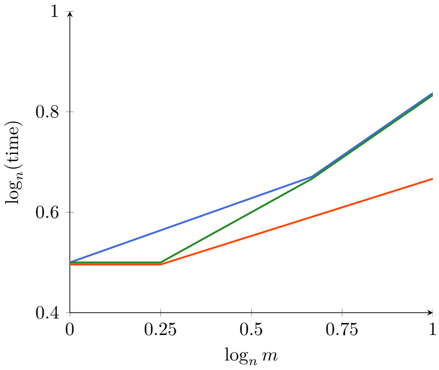


Figure 1 Quantum complexity of Hopcroft's problem in 2 dimensions on n lines and m points, assuming $m \le n$. The blue line shows the complexity of the quantum algorithm with the partition tree (Theorem 2); the green line shows the complexity of the quantum walk algorithm with the line arrangement data structure (Theorem 3); the red line shows the query complexity (Theorem 5).

The upper bound in Theorem 5 is given by the algorithm of Tani [41]. There is an obvious hurdle in implementing it in the same time complexity here. Their quantum walk would require a history-independent dynamic data structure for storing a set of lines and a set of points supporting the detection of incidence existence. Even assuming the most optimistic quantum versions of the known data structures, the sufficiently powerful speedup looks unfeasible. The next two sections describe the algorithms we have obtained by speeding up two fundamental geometrical query data structures quantumly.

4 Algorithm 1: quantum backtracking with partition trees

We begin with a brief overview of the classical partition tree data structure and its quantum speedup, and then proceed with the description of the quantum algorithm.

4.1 Partition trees

The partition tree is a classical data structure designed for the task of *simplex range searching*. In this problem, one is given n points in a d-dimensional space; the task is to answer queries where the input is a simplex and the answer is the number of points inside that simplex. The *partition tree* is a well-known data structure which can be used to solve this task [4].

This data structure can be described as a tree in the following way. The tree stores n points and each subtree stores a subset of these points. Each interior vertex v is attributed with a simplex $\Delta(v)$ such that all of the points stored in this subtree belong to the interior of $\Delta(v)$. For each interior vertex v, the subsets of the points stored in its children subtrees form a partition of the points stored in the subtree of v. Each leaf vertex stores a constant number of points in a list. For our purposes, each interior vertex is attributed only with information about its children and no information about the points stored in its subtree.

In this work, we are interested in the *hyperplane emptiness queries.* Given n points in the d-dimensional space, the task is to answer queries where the input is an arbitrary hyperplane and the answer is whether there is a point that lies on the given hyperplane. These queries can also be answered using partition trees:

▶ **Lemma 6** (Hyperplane emptiness query procedure)**.** *Let $\mathcal{T}$ be a partition tree storing a set of points. Let the* tree query cost *$c(\mathcal{T})$ be the maximum number of simplices of $\mathcal{T}$ that intersect an arbitrary hyperplane. Then the hyperplane emptiness query can be answered in $O(c(\mathcal{T}))$ time.*

Proof. The procedure for answering a query is as follows. We start at the root and traverse $\mathcal{T}$ recursively. If the current vertex is an interior vertex v and the query hyperplane is h, then we recurse only in the children of v such that $\Delta(v)$ intersects h. If the current vertex is a leaf vertex, we check whether any of its points lies on h. The running time is evidently linear in the number of simplices intersecting h. ◀

There are different ways to construct partition trees, but a long chain of works in computational geometry resulted in an optimal version of the partition tree [17]. Even though their goal is to answer simplex queries, in fact the main result gives an upper bound on $c(\mathcal{T})$ for their partition tree:

▶ **Theorem 7** (Partition tree [17])**.** *For any set of n points in d dimensions, there is a partition tree $\mathcal{T}$ such that:*

- *it can be built in $O(n \log n)$ time and requires $O(n)$ space;*
- *$c(\mathcal{T}) = O(n^{1-1/d})$; hence, a hyperplane emptiness query requires $O(n^{1-1/d})$ time;*
- *each vertex has $O(1)$ children and the depth of the tree is $O(\log n)$.*

To speed up the emptiness query time of the partition tree quantumly we use the quantum backtracking algorithm [36]. Their quantum algorithm searches for a *marked* vertex in a tree $\mathcal{S}$. The markedness is defined by a black-box function $P : V(\mathcal{T}) \to \{\textsc{true}, \textsc{false}, \textsc{indeterminate}\}$. For leaf vertices v, we have $P(v) \in \{\textsc{true}, \textsc{false}\}$. A vertex v is marked if $P(v) = \textsc{true}$, and the task is to determine whether $\mathcal{S}$ contains a marked vertex.

The root of $\mathcal{S}$ is known and the rest of the tree is given by two other black-box functions. The first, given a vertex v, returns the number of children $d(v)$ of v. The second, given v and an index $i \in [d(v)]$, returns the i-th child of v. The main result is a quantum algorithm for detecting a marked vertex in $\mathcal{S}$:

▶ **Theorem 8** (Quantum algorithm for backtracking [36, 10])**.** *Suppose you are given a tree $\mathcal{S}$ by the black boxes described above and upper bounds T and h on the size and the height of the tree. Additionally suppose that each vertex of $\mathcal{S}$ has $O(1)$ children. Then there is a bounded-error quantum algorithm that detects a marked vertex in $\mathcal{S}$ with query and time complexity $O(\sqrt{Th})$.*

When we apply it to the partition tree from Theorem 7, we get:

▶ **Theorem 1.** *There is a bounded-error quantum algorithm that can answer hyperplane emptiness queries in $\widetilde{O}(\sqrt{n^{1-1/d}})$ time.*

Proof. The procedure of Lemma 6 examines a subtree $\mathcal{S}$ of $\mathcal{T}$. We will apply Theorem 8 to $\mathcal{S}$. Suppose that h is a query hyperplane. The black box P returns INTERMEDIATE for any interior vertex v and for a leaf vertex v returns TRUE iff some point stored in v lies on h. The second black box returns the number of children of v in $\mathcal{T}$ if $\Delta(v)$ intersects h and 0 otherwise. The black box returning the i-th child simply fetches it from the partition tree that is stored in memory. All of these black boxes require only constant time to implement. Since we know that $|\mathcal{S}| = O(n^{1-1/d})$ and the height of $\mathcal{S}$ is $O(\log n)$ from Theorem 7, there is a quantum algorithm that solves the problem in $O(\sqrt{n^{1-1/d} \cdot \log n})$ time by Theorem 8. ◀

4.2 Quantum algorithm

Now we can apply the previous theorem to Hopcroft's problem.

▶ **Theorem 2.** *There is a bounded-error quantum algorithm which solves Hopcroft's problem with n hyperplanes and $m \le n$ points in d dimensions in time:*

- $\widetilde{O}(n^{\frac{d}{2(d+1)}} m^{1/2})$, *if* $m \ge n^{\frac{d}{d+1}}$;
- $\widetilde{O}(n^{1/2} m^{\frac{d-1}{2d}})$, *if* $m \le n^{\frac{d}{d+1}}$.

If $n = m$, then the algorithm has complexity $\widetilde{O}(n^{1-\frac{1}{2(d+1)}})$.

Proof. In the first case, we partition the whole set of points into m/r groups of size $r = n^{\frac{d}{d+1}}$. Using Grover's search, we search for a group that contains a point belonging to some line. To determine whether it's true for a fixed group, first we build a partition tree of Theorem 7 to store these points. Then we run Grover's search over all lines and determine whether a line contains some point from the group using the quantum query procedure from Theorem 1. Overall, the complexity of this algorithm (without logarithmic factors) is

$$O\left(\sqrt{\frac{m}{r}}\left(r + \sqrt{n} \cdot \sqrt{r^{1-1/d} \cdot \log r}\right)\right) = O\left(\sqrt{mr} + \sqrt{\frac{nm}{r^{1/d}} \log r}\right) = O\left(\sqrt{mn^{\frac{d}{d+1}} \log n}\right)$$

If we use the variation of Grover's search with bounded-error inputs (Theorem 4), then we do not incur extra logarithmic factors. If $m \le n^{\frac{d}{d+1}}$, then we simply build the partition tree on all m points, then use Grover's search over all lines and query the partition tree for each of them. The complexity in that case is

$$O\left(m \log m + \sqrt{n} \cdot \left(\sqrt{m^{1-1/d} \cdot \log m} + \log n\right)\right) = O\left(\sqrt{nm^{1-1/d}} \cdot \log n\right),$$

because the second term dominates the first (up to logarithmic factors). ◀

5 Algorithm 2: quantum walk with line arrangements

First we describe a classical history-independent data structure for storing an arrangement of a set of lines. After that, we describe the quantum walk algorithm that uses it for solving Hopcroft's problem.

5.1 Line arrangements

We begin with a few definitions (for a thorough treatment, see e.g. [25]). For a set of lines L, the *line arrangement* $\mathcal{A}(L)$ is the partition of the plane into connected regions bounded by the lines. The convex regions with no other lines crossing them are called *cells* and their sides are called the *edges* of the arrangement (note that some cells may be infinite). The intersection points of the lines are called the *vertices* of the arrangement.

For a set of lines L in a general position, the *k-level* is the set of edges of $\mathcal{A}(L)$ such that there are exactly k lines above each edge (for the special case of a vertical line, we consider points to the left of it to be "above"). By this construction each k-level forms a polygonal chain. Our data structure will store the line arrangement of a subset S of lines by keeping track of all $|S|$ levels, with each level being stored in a skip list. We will be able to support the following operations:

- Answering whether a point lies on some line in $O(\log^6 n)$ time.
- Inserting or removing a line in $O(|S| \log^4 n + \log^6 n)$ time.

5.2 Skip lists

We will need a history-independent data structure which can store a set of elements and support polylogarithmic time insertion/removal operations. For that purpose use the skip list data structure by Ambainis from the Element Distinctness algorithm [9]. Among other applications, it was also used by [1] for the Closest Pair problem, where they also gave a brief description. Here we shortly describe only the details required in our algorithm and rely on the facts already proved in these papers.

Suppose that the skip list stores some set of elements $S \subseteq [N]$, according to some order such that comparing two elements requires constant time. In a skip list, each element $i \in S$ is assigned an integer $\ell_i \in [0, \ldots, \ell_{\max}]$, where $\ell_{\max} = \lceil \log_2 N \rceil$. The skip list itself then consists of $\ell_{\max} + 1$ linked lists where the k-th list contains all $i \in S$ such that $\ell_i \geq k$. We will call the k-th linked list by the k-th layer (to not confuse them with k-levels). In other words, each element $i \in S$ is attributed with $\ell_i + 1$ pointers, where the k-th of them points to the smallest element j such that $j > i$ and $\ell_j \geq k$, or to `Null`, if there is no such j. The first element of the skip list is called the *head* and it only stores the $\ell_{\max}$ pointers, the beginnings of each layer (it is convenient to imagine this element storing value 0, which is smaller then any element of S).

The search of an element $i \in S$ is implemented in the following way. First, we traverse the $\ell_{\max}$-th layer to find the last element j such that $j \leq i$. If $j = i$, we are done; otherwise, traverse the layer below starting from j to find the last element $j' \leq i$ there. By repeating such iterations, we will find i. Insertion of $i \notin S$ is implemented similarly: first we find the last element $j_k \in S$ such that $j < i$, for all layers k. Then we update the pointers: for each layer $k \leq l_i$, we set the pointer from i to be be equal to the pointer from j_k; then we set the pointer from j_k to i. If an operation requires more than $O(\log^4 N)$ time, it is terminated.

To store the elements in memory, a specific hash table is used. Element's entry contains the description of the element together with other data attributed to it (in particular, the values of the pointers). We will not describe the details of the implementation, as it is the same as Ambainis'. The whole data structure can sometimes malfunction (e.g., the hash table buckets can overflow or the operation of the skip list can take too long), but it is shown in [9] that probability of such is small. More specifically, the probability that at least one operation malfunctions among $O(N)$ operations is only $O(1/\sqrt{N})$. Thus, as we are aiming at a sublinear algorithm, we don't need to worry about the error probability of the skip lists. We also note that the memory requirement of Ambainis' skip list is $O(r \log^3 N)$, if at most r elements need to be stored.

5.3 Data structure

Our data structure will operate mainly using the intersection points of the lines. To keep the unique description of the data for history independence, we describe each intersection point in the following way. Suppose the given lines are labeled $\ell_1, \ldots, \ell_n$. For any two lines ℓ_i and

ℓ_j, let $P_{i,j}$ be its intersection point. In an arrangement which includes both of these lines, we describe the left and right edges of ℓ_i connected to $P_{i,j}$ by $\texttt{left}(\ell_i, \ell_j)$ and $\texttt{right}(\ell_i, \ell_j)$. In that case there will be a k such that the k-level contains the edges $\texttt{left}(\ell_i, \ell_j)$ and $\texttt{right}(\ell_j, \ell_i)$, and the $(k+1)$-level contains the edges $\texttt{left}(\ell_j, \ell_i)$ and $\texttt{right}(\ell_i, \ell_j)$. We describe $P_{i,j}$ in the first case by the by the pair of integers $\nu_{i,j} = (i, j)$ and by $\nu_{j,i} = (j, i)$ in the second case, see Figure 2. We call these pairs *path points* of $P_{i,j}$. Note that we can calculate the coordinates of any path point in constant time as it description consists of the indices of the lines.

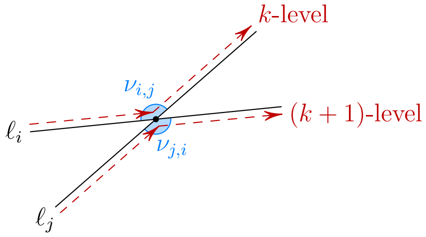


Figure 2 Path points of a line intersection.

Now we will describe the data structure, which stores an arrangement of a subset of given lines. It will operate with multiple skip lists each storing a set of path points of the arrangement. To ensure the unique representation of the data, we encode the pointers of the skip lists with the values of the path points themselves. To represent the beginning of a level from line ℓ_i, we use a "fictitious" starting path point $\nu_{i,i} = (i, i)$. The last element of skip lists we encode with a special "null" path point $\nu_{\texttt{Null}}$. We implement the following skip lists:

- The skip lists that contain the path points of the current k-levels in order from left to right. These skip lists are stored implicitly, since adding and removing lines changes the indexing of the levels and the levels themselves. For each path point ν stored in such a skip list, we additionally store an array $\texttt{Next}_\nu[0 \ldots l_{\max}]$ storing the values of next path points of its skip list for each skip list layer.
- $\texttt{Start}$ – contains the heads of all level skip lists in the current arrangement. If the first edge of a k-level belongs to ℓ_i, the head of its skip list is $\nu_{i,i}$, and we additionally store $\texttt{Next}_{\nu_{i,i}}$ to access the respective level skip list. The heads are ordered by the slope of the lines with the x-axis corresponding to the head path points.

Further we will describe the implementation of the operations.

5.3.1 Point location

To detect whether a point belongs to some line, we essentially binary search through all k-levels and check whether the given point is strictly above, below or belongs to that level. The binary search is implicitly performed by searching in the skip list $\texttt{Start}$. For a given level, we then search for its edge such that the x-coordinate of the point belongs to the projection of that edge on the x-axis. When this edge is found, we check the relative vertical position of this point in constant time.

Essentially we have two nested searches in the skip list structure, so the complexity of this step is $O(\log^8 n)$, but we can show a better estimate. Ambainis ([9], see the proof of Lemma 6) showed that in a skip list operation, at most $O(\log^2 n)$ pointer accesses are necessary. We run the search in the inner skip list only after accessing a pointer. Therefore, the outer search requires $O(\log^4 n)$ steps and the inner searches altogether require $O(\log^6 n)$ steps, so we improve our estimate to $O(\log^6 n)$.

5.3.2 Line insertion and removal

We will only describe the procedure of inserting a line in the data structure, as removing a line can be implemented by a reverse quantum circuit. Suppose the line to be inserted is ℓ_i; our task is to correctly update the pointers of the skip lists. As we will see, conveniently it suffices to update only the pointers of the new edges created by the insertion of the new line.

Our first step is to construct the edge along the given line. Figure 3 shows an example of the new edges collinear with ℓ_i being created. Suppose that P_{i,j_1} and P_{i,j_2} are two consecutive intersection points with ℓ_i (P_{i,j_1} is left of P_{i,j_2}). Some k-level will pass through an edge connecting P_{i,j_1} and P_{i,j_2}. This level will also pass through $\texttt{left}(\ell_{j_1}, \ell_i)$ and $\texttt{right}(\ell_{j_2}, \ell_i)$, as all edges of a level are directed from left to right. Therefore, the edge should connect $\nu_{j_1,i}$ with ν_{i,j_2}. There are two special cases for the first and the last edge; in the first case, the first path point is $\nu_{i,i}$ and for the second, the second path point is $\nu_{\texttt{Null}}$. According to this order, we insert all path points to the skip list starting with $\nu_{i,i}$ (and using the pointers $\texttt{Next}$). However, we don't insert $\nu_{i,i}$ into $\texttt{Start}$ yet.

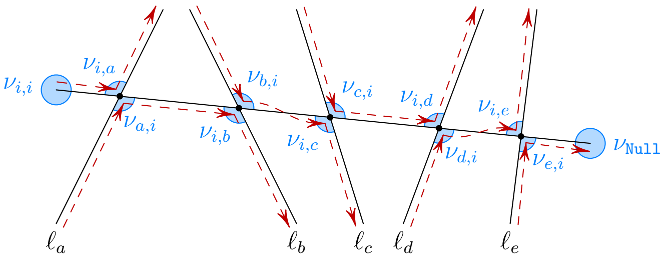


Figure 3 New edges along the inserted line.

Next we will correct all of the level skip lists according to the updated arrangement. Essentially, our algorithm performs a sweep line from right to left which swaps the tails of skip lists at the intersection points of ℓ_i with other lines. First, we create an array containing the same set of path points as the skip list of ℓ_i except $\nu_{i,i}$ and $\nu_{\texttt{Null}}$ and sort them by the x coordinate (at the end of the procedure we null the array by applying this in reverse). We then examine the intersection points of ℓ_i with the other lines from right to left.

Suppose we examine the intersection point $P_{i,j}$ of ℓ_i with ℓ_j.Then there is some edge from the old arrangement from $\nu^{(1)}$ to $\nu^{(2)}$ along ℓ_j which intersects ℓ_i at $P_{i,j}$. The respective pair of path points is $\nu_{i,j}$ and $\nu_{j,i}$. Then some k-level will pass along ℓ_i through $\nu_{i,j}$ and $\nu^{(2)}$, and some adjacent level (either $(k+1)$-level or $(k-1)$-level) will pass along ℓ_j from $\nu^{(1)}$ to $\nu_{j,i}$. Observe that the tails of these levels (from this intersection point to the right) have been correctly updated by the sweep line. Therefore, we just need to swap the tails of these two level skip lists.

To find the edge from $\nu^{(1)}$ to $\nu^{(2)}$, we use the point location operation with $P_{i,j}$. Since we know that this point will belong to some k-level, we modify the point location operation so as to return the head $\nu_{h,h}$ of this k-level. Note that although the level skip lists are partially updated, they still represent the old arrangement to the left of the previously examined intersection point, since the sweepline operates from right to left. Now we have to swap the tails of the skip list with head $\nu_{i,i}$ after $\nu_{i,j}$ and the skip list with head $\nu_{h,h}$ after $\nu^{(1)}$.

Generally, suppose that we wish to swap the tails of skip lists with heads $\nu^{(a_h)}$ and $\nu^{(b_h)}$ after elements $\nu^{(a_t)}$ and $\nu^{(b_t)}$, respectively. By searching $\nu^{(a_t)}$ in the $\nu^{(a_h)}$ skip list, we find all $l_{\max}$ path nodes $\nu^{(a_l)}$ such that $\texttt{Next}_{\nu^{(a_l)}}[l]$ points to a path node after $\nu^{(a_t)}$, for each $l \in [l_{\max}]$. Similarly we define and find $\nu^{(b_l)}$ path nodes. Then we simply swap the values of $\texttt{Next}_{\nu^{(a_l)}}[l]$ and $\texttt{Next}_{\nu^{(b_l)}}[l]$ for all $l \in [0, l_{\max}]$.

To conclude the procedure, we insert $\nu_{i,i}$ (together with $\texttt{Next}_{\nu_{i,i}}$) into $\texttt{Start}$. As we only performed $O(r)$ skip list searching and insertion operations (swapping the tails has the same complexity as an element insertion, as it's only updating $2l_{\max} + 2$ pointers) and a point location operation, the complexity of the procedure is $O(r \log^4 n + \log^6 n)$. If $r = n^p$ for some $p > 0$, this simplifies to $O(r \log^4 n)$.

5.4 Quantum algorithm

We use the MNRS framework quantum walk on the Johnson graph [9, 34]. In this framework, we search for a marked vertex in an ergodic reversible Markov chain on a state space X defined by the transition matrix $P = (p_{x,y})_{x,y \in X}$. Let the subset of marked states be $M \subseteq X$. To perform the quantum walk, the following procedures need to be implemented:

- Setup operation with complexity S. This procedure prepares the initial state of the quantum walk:

$$|0\rangle\,|0\rangle \mapsto \sum_{x \in X} \sqrt{\pi_x}\,|x\rangle\,|0\rangle\,,$$

  where π_x is the stationary distribution of P.
- Update operation with complexity U. This procedure essentially performs a step of the quantum walk by applying the transformation:

$$|x\rangle\,|0\rangle \mapsto |x\rangle \sum_{y \in X} \sqrt{p_{x,y}}\,|y\rangle\,.$$

- Checking operation with complexity C. This procedure performs the phase flip on the marked vertices:

$$|x\rangle\,|y\rangle \mapsto \begin{cases} -\,|x\rangle\,|y\rangle & \text{if } x \in M, \\ |x\rangle\,|y\rangle & \text{otherwise.} \end{cases}$$

We examine the Johnson graph on the state space X being the set of all size r subsets of $[n]$. Two vertices $x, y \in X$ are connected in this graph if the intersection of the corresponding subsets has size $r - 1$. For the Markov chain, the transition probability is $p_{x,y} = \frac{1}{r(n-r)}$ for all edges. Then we have the following theorem:

▶ **Theorem 9** (Quantum walk on the Johnson graph [9, 34])**.** *Let P be the random walk on the Johnson graph on size r subsets of $[n]$ with intersection size $r - 1$, where $r = o(n)$. Let M be either empty or the set of all size r subsets that contain a fixed element. Then there is a bounded-error quantum algorithm that determines whether M is empty, with complexity*

$$O\left(S + \frac{1}{\sqrt{r/n}}\left(\frac{1}{\sqrt{1/r}} \cdot U + C\right)\right) = O\left(S + \sqrt{n} \cdot U + \sqrt{\frac{n}{r}} \cdot C\right).$$

We can now prove our result:

▶ **Theorem 3.** *There is a bounded-error quantum algorithm that solves Hopcroft's problem with n lines and $m \leq n$ points in the plane in time:*

- $\widetilde{O}(n^{1/3} m^{1/2})$, *if* $n^{2/3} \leq m$;
- $\widetilde{O}(n^{2/5} m^{2/5})$, *if* $n^{1/4} \leq m \leq n^{2/3}$;
- $\widetilde{O}(n^{1/2})$, *if* $m \leq n^{1/4}$.

In particular, the complexity is $\widetilde{O}(n^{5/6})$ when $n = m$.

Proof. By the duality of Hopcroft's problem, we can exchange n and m; from here on, suppose that $m \geq n$. We do this, since in Theorem 2 it is important that the number of lines is larger and here it is important that the number of points is larger, but we wish to keep the meaning of n and m to avoid confusion. Our algorithm is a quantum walk on the Johnson graph of size r subsets of the given n lines. We will choose r depending on m, but r will always be m^p for some $p > 0$. A set S is marked if it contains a line such that there exists a point from the set of m given points that belongs to this line.

For the implementation, we follow the description of [1] for the quantum algorithm for closest points. For a set S, the state of the walk will be $|S, d(S)\rangle$, where $d(S)$ is our data structure for the line arrangement of S. We then implement the quantum walk procedures:

- For the Johnson graph, π is the uniform distribution. Thus, we first generate a uniform superposition over all subsets S in $O(\log \binom{n}{r}) = O(r \log n)$ time. Then we create $d(S)$ by inserting all lines of S into an initially empty data structure, requiring $O(r^2 \log^4 n)$ time.
- Suppose that S and S' are two size r subsets with $|S \cap S'| = r - 1$ so that $S' = (S \setminus \{i\}) \cup \{j\}$. We then represent a state $|S, d(S)\rangle |S', d(S')\rangle$ with $|S, d(S)\rangle |i, j\rangle$. As the Markov chain probabilities are the same for all edges, we need to implement the transition $|S, d(S)\rangle |0, 0\rangle \mapsto \sum_{i \in S} \sum_{j \notin S} |S', d(S)'\rangle |j, i\rangle$. To do that, first we create a uniform superposition of all $i \in S$ and $j \notin S$ in $O(r \log^4 n)$ time (see Appendix C), obtaining $\sum_{i \in S} \sum_{j \notin S} |S, d(S)\rangle |i, j\rangle$. Then, for fixed i and j, we remove ℓ_i from $d(S)$ and insert ℓ_j, obtaining $d(S')$; this takes $O(r \log^4 n)$ time. Finally, we swap the indices i and j in the second register in $O(\log n)$ time.
- The checking operation runs Grover's search over all m points and for each of them performs the point location operation. The complexity is $O(\sqrt{m} \log^6 n)$.

By Theorem 9 the complexity of the algorithm is

$$O\left(r^2 \log^4 n + \sqrt{n} r \log^4 n + \sqrt{\frac{n}{r}} \sqrt{m} \log^6 n\right).$$

Suppose that $m^{2/3} \leq n$ and pick $r = m^{1/3}$. Then the second term dominates the first and we can simplify the expression to

$$O\left(\sqrt{n} \log^4 n \left(r + \sqrt{\frac{m}{r}} \log^2 n\right)\right) = O(n^{1/2} m^{1/3} \log^6 n).$$

If we have $m^{1/4} \leq n \leq m^{2/3}$ we pick $r = (nm)^{1/5}$. Then we have $r \geq (n^{1+3/2})^{1/5} = \sqrt{n}$, and this time the first term in the complexity dominates the second, and the complexity is

$$O\left(r^2 \log^4 n + \sqrt{\frac{nm}{r}} \log^6 n\right) = O(n^{2/5} m^{2/5} \log^6 n).$$

Finally, for $n \leq m^{1/4}$, we don't have to use either the quantum walk or the history-independent data structure. First, we build any classical data structure for point location in a line arrangement with $O(n^2 \log n)$ build time and space and $O(\log n)$ query time (e.g. see [25], Chapter 11). Then we run Grover's search over all points and for each check whether it belongs to some line. The complexity in this case is

$$O(n^2 \log n + \sqrt{m}(\log m + \log n)) = O(\sqrt{m} \log m). \qquad \blacktriangleleft$$

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. In Shubhangi Saraf, editor, *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:43, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CCC.2020.16`.

2 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2015, pages 218–230, USA, 2015. Society for Industrial and Applied Mathematics. `doi:10.5555/2722129.2722146`.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of Faster Alignment of Sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 39–51, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. `doi:10.1007/978-3-662-43948-7_4`.

4 Pankaj K. Agarwal. *Simplex Range Searching and Its Variants: A Review*, pages 1–30. Springer International Publishing, Cham, 2017. `doi:10.1007/978-3-319-44479-6_1`.

5 Jonathan Allcock, Jinge Bao, Aleksandrs Belovs, Troy Lee, and Miklos Santha. On the quantum time complexity of divide and conquer, 2023. `arXiv:2311.16401`.

6 Jonathan Allcock, Jinge Bao, João F. Doriguello, Alessandro Luongo, and Miklos Santha. Constant-depth circuits for Uniformly Controlled Gates and Boolean functions with application to quantum memory circuits, 2023. `arXiv:2308.08539`.

7 Andris Ambainis. Polynomial Degree vs. Quantum Query Complexity. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS 2003, pages 230–239, Washington, DC, USA, 2003. IEEE Computer Society. `doi:10.1109/SFCS.2003.1238197`.

8 Andris Ambainis. Polynomial Degree and Lower Bounds in Quantum Complexity: Collision and Element Distinctness with Small Range. *Theory of Computing*, 1(3):37–46, 2005. `doi:10.4086/toc.2005.v001a003`.

9 Andris Ambainis. Quantum Walk Algorithm for Element Distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. `doi:10.1137/S0097539705447311`.

10 Andris Ambainis and Martins Kokainis. Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 989–1002, New York, NY, USA, 2017. ACM. `doi:10.1145/3055399.3055444`.

11 Andris Ambainis and Nikita Larka. Quantum Algorithms for Computational Geometry Problems. In Steven T. Flammia, editor, *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:10, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TQC.2020.9`.

12 A. Bahadur, C. Dürr, T. Lafaye, and R. Kulkarni. Quantum query complexity in computational geometry revisited. In Eric J. Donkor, Andrew R. Pirich, and Howard E. Brandt, editors, *Quantum Information and Computation IV*, volume 6244, page 624413. International Society for Optics and Photonics, SPIE, 2006. `doi:10.1117/12.661591`.

13 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. `doi:10.1137/S0097539796300933`.

14 Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. Limits of Quantum Speed-Ups for Computational Geometry and Other Problems: Fine-Grained Complexity via Quantum Walks. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:12, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ITCS.2022.31`.

15 Harry Buhrman, Bruno Loff, Subhasree Patro, and Florian Speelman. Memory Compression with Quantum Random-Access Gates. In François Le Gall and Tomoyuki Morimae, editors, *17th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2022)*, volume 232 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TQC.2022.10.

16 Harry Buhrman and Robert Špalek. Quantum verification of matrix products. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA 2006, pages 880–889, USA, 2006. Society for Industrial and Applied Mathematics. arXiv:quant-ph/0409035.

17 Timothy M. Chan. Optimal Partition Trees. *Discrete & Computational Geometry*, 47:661–690, 2012. doi:10.1007/s00454-012-9410-z.

18 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1), 2021. doi:10.1145/3402926.

19 Timothy M. Chan and Da Wei Zheng. Hopcroft's Problem, Log-Star Shaving, 2D Fractional Cascading, and Decision Trees. *ACM Trans. Algorithms*, 2023. doi:10.1145/3591357.

20 Timothy M. Chan and Da Wei Zheng. *Simplex Range Searching Revisited: How to Shave Logs in Multi-Level Data Structures*, pages 1493–1511. SODA 2023. Society for Industrial and Applied Mathematics, 2023. doi:10.1137/1.9781611977554.ch54.

21 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.

22 Bernard Chazelle and Burton Rosenberg. Simplex range reporting on a pointer machine. *Computational Geometry*, 5(5):237–247, 1996. doi:10.1016/0925-7721(95)00002-X.

23 Bernard Chazelle and Emo Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete & Computational Geometry*, 4:467–489, 1989. doi:10.1007/BF02187743.

24 Andrew M. Childs, Shelby Kimmel, and Robin Kothari. The Quantum Query Complexity of Read-Many Formulas. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, pages 337–348, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. arXiv:1112.0548.

25 Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, Heidelberg, 1987. doi:10.1007/978-3-642-61568-9.

26 Jeff Erickson. On the relative complexities of some geometric problems. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, pages 85–90. Carleton University, 1995. URL: https://jeffe.cs.illinois.edu/pubs/relative.html.

27 Jeff Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16:389–418, 1996. doi:10.1007/BF02712875.

28 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum Random Access Memory. *Phys. Rev. Lett.*, 100:160501, 2008. doi:10.1103/PhysRevLett.100.160501.

29 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 1996, pages 212–219, New York, NY, USA, 1996. Association for Computing Machinery. doi:10.1145/237814.237866.

30 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum Search on Bounded-Error Inputs. In *Automata, Languages and Programming*, ICALP 2003, pages 291–299, Berlin, Heidelberg, 2003. Springer-Verlag. doi:10.1007/3-540-45061-0_25.

31 J. Mark Keil, Fraser McLeod, and Debajyoti Mondal. Quantum Speedup for Some Geometric 3SUM-Hard Problems and Beyond, 2024. arXiv:2404.04535.

32 Samuel Kutin. Quantum Lower Bound for the Collision Problem with Small Range. *Theory of Computing*, 1(2):29–36, 2005. doi:10.4086/toc.2005.v001a002.

33 François Le Gall. Improved Quantum Algorithm for Triangle Finding via Combinatorial Arguments. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 216–225, 2014. doi:10.1109/FOCS.2014.31.

34 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via Quantum Walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. doi:10.1137/090745854.

35 Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10:157–182, 1993. doi:10.1007/BF02573972.

36 Ashley Montanaro. Quantum-Walk Speedup of Backtracking Algorithms. *Theory of Computing*, 14(15):1–24, 2018. doi:10.4086/toc.2018.v014a015.

37 Ketan Mulmuley and Sandeep Sen. Dynamic point location in arrangements of hyperplanes. *Discrete & Computational Geometry*, 8:335–360, 1992. doi:10.1007/BF02293052.

38 Kunihiko Sadakane, Noriko Sugarawa, and Takeshi Tokuyama. Quantum Computation in Computational Geometry. *Interdisciplinary Information Sciences*, 8(2):129–136, 2002. doi:10.4036/iis.2002.129.

39 Kunihiko Sadakane, Norito Sugawara, and Takeshi Tokuyama. Quantum Algorithms for Intersection and Proximity Problems. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation*, pages 148–159, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-45678-3_14.

40 Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986. doi:10.1145/6138.6151.

41 Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009. Mathematical Foundations of Computer Science (MFCS 2007). doi:10.1016/j.tcs.2009.08.030.

42 Nilton Volpato and Arnaldo Moura. Tight Quantum Bounds for Computational Geometry Problems. *International Journal of Quantum Information*, 07(05):935–947, 2009. doi:10.1142/S0219749909005572.

43 Nilton Volpato and Arnaldo Moura. A fast quantum algorithm for the closest bichromatic pair problem, 2010. URL: https://www.ic.unicamp.br/~reltech/2010/10-03.pdf.

44 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004). doi:10.1016/j.tcs.2005.09.023.

45 Ryan Williams. Pairwise comparison of bit vectors, 2017. URL: https://cstheory.stackexchange.com/q/37369.

46 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2014, pages 1867–1877, USA, 2014. Society for Industrial and Applied Mathematics. doi:10.5555/2634074.2634209.

47 Shengyu Zhang. Promised and Distributed Quantum Search. In Lusheng Wang, editor, *Computing and Combinatorics*, pages 430–439, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11533719_44.

A Model

We use the standard quantum circuit model together with Quantum Random Access Gates (QRAG) (see, for example, [6]). This gate implements the following mapping:

$$|i\rangle\,|b\rangle\,|x_1,\dots,x_N\rangle \mapsto |i\rangle\,|x_i\rangle\,|x_1,\dots,x_{i-1},b,x_{i+1},\dots,x_N\rangle\,.$$

Here, the last register represents the memory space of N bits. Essentially, QRAG gates allow both for reading memory in superposition as well as writing operations. We note that both our algorithms require "read-write" quantum memory, so it is not sufficient to use the weaker "read-only" QRAM gate, which is enough for some quantum algorithms.

To keep the analysis of the algorithms clean, we abstract the complexity of basic underlying operations under the "unit cost". This includes the running time of:

- basic arithmetic operations on $O(\log n)$ bits;
- the implementation of QRAG and elementary gates;
- the running time of a quantum oracle, which with a single query can return the description of any point or line.

In the end, we measure the time complexity in the total amount of unit cost operations. The unit cost can be taken as the largest running time of the operations listed above, which will add a multiplicative factor in the complexity.

Assuming that an application of a QRAG gate takes unit time is also useful for utilizing the existing classical algorithms in the RAM model. The classical algorithms that we use work in the real RAM model, where arithmetic operations and memory calls on $O(\log n)$ bits are considered to be executed in constant time. Thus, we work in the quantum analogue of the real RAM, and if there is a time T classical real RAM algorithm, then we can use it in time $O(T)$ in this model. The actual implementation of QRAG is an area of open research and debate; however, there exist theoretical proposals that realize such operations in time polylogarithmic in the size of the memory, like the bucket brigade architecture of [28].

B Query complexity

Proof of Theorem 1. For the upper bound, Hopcroft's problem can be seen as an instance of the bipartite *subset-finding* problem, in which one is given query access to two sets X and Y of sizes n and m, respectively, and needs to detect whether there is a pair $x \in X$, $y \in Y$ satisfying some predicate $R : X \times Y \to \{0, 1\}$. For this problem, Tani gave a quantum algorithm with query complexity $O(n^{1/3}m^{1/3} + \sqrt{n} + \sqrt{m})$ [41].

For the lower bound, we can reduce the bipartite element distinctness problem (also known as Claw Finding) to Hopcroft's problem. In this problem, we have two sets of variables $x_1, \ldots, x_n \in [N]$ and $y_1, \ldots, y_m \in [N]$ and we need to detect whether $x_i = y_j$ for some i, j. Zhang proved that for this problem $\Omega(n^{1/3}m^{1/3} + \sqrt{n} + \sqrt{m})$ quantum queries are needed [47]. We reduce each x_i to a line $x = x_i$ and each y_j to a point $(y_j, 0)$. Then $x_i = y_j$ only iff some point belongs to some line, so the statement follows. ◀

C Implementation details

To generate a uniform state proportional to $\sum_{i \in S} |i\rangle$ from $|S, d(S)\rangle$, we can proceed as follows. First generate the uniform superposition $\sum_{k \in [r]} |k\rangle$ in $O(\log n)$ time. Then we can find the number of the k-th line in S by iterating over the elements of the `Start` skip list as in the usual linked list until we find the k-th one; this requires $O(r \log^4 n)$ time. We then null the register $|k\rangle$ by applying a reverse procedure and decrementing k in each step. Overall, we obtain the state proportional to $\sum_{i \in S} |i\rangle$ in $O(r \log^4 n)$ time.

To generate a uniform state proportional to $\sum_{j \notin S} |j\rangle$ from $|S, d(S)\rangle$, we can apply a different procedure. First, we can generate the state $\sum_{k \in [n]} |k\rangle |0\rangle$ in $O(\log n)$ time. For a fixed k, we can check whether $k \in S$ using $d(S)$ in $O(\log^4 n)$ time and write 1 in that case in the second register. By measuring the second register, we obtain the required state with probability $1 - r/n > 1/2$. We can use $O(\log n)$ copies of such state to obtain the required state with error probability only $O(1/n)$, which doesn't impact the final constant success probability. Overall, this step requires $O(\log^5 n)$ time, which is negligible compared to $O(r \log^4 n)$.

A New Characterization of FAC^0 via Discrete Ordinary Differential Equations

Melissa Antonelli
HIIT & University of Helsinki, Finland

Arnaud Durand
Université Paris Cité, France

Juha Kontinen
University of Helsinki, Finland

Abstract

Implicit computational complexity is an active area of theoretical computer science, which aims at providing machine-independent characterizations of relevant complexity classes. One of the seminal works in this field appeared in 1965, when Cobham introduced a function algebra closed under bounded recursion on notation to capture **FP**. Later on, several complexity classes have been characterized using *limited* recursion schemas. In this context, a new approach was recently introduced, showing that ordinary differential equations (ODEs) offer a natural tool for algorithmic design and providing a characterization of **FP** by an ODE-schema. The overall goal of the present work is precisely that of generalizing this approach to parallel computation, obtaining an original ODE-characterization for the small circuit classes $\mathbf{FAC}^0$ and $\mathbf{FTC}^0$.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Circuit complexity

Keywords and phrases Implicit computational complexity, parallel computation, ordinary differential equations, circuit complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.10

Funding *Melissa Antonelli*: Supported by HIIT and Maupertuis' SMR Programme, on behalf of the Institute Français in Helsinki, the French Embassy to Finland, the French Ministry of Higher Education, Research and Innovation in partnership with the Finnish Society for Science and Letters and the Finnish Academy of Sciences and Letters.
Arnaud Durand: Supported by Maupertuis' SMR Programme.
Juha Kontinen: Supported by the Academy of Finland grant 345634.

1 Introduction

As computability theory investigates the limits of what is algorithmically computable, complexity theory classifies functions based on the amount of resources, typically time and space, required by a machine to compute them. Taking a different point of view, implicit computational complexity (ICC) aims at providing machine-independent characterizations, which in turn have offered remarkable insights on the corresponding classes and led to relevant meta-theorems in various domains, such as database theory and constraint satisfaction.

One of the major approaches to computability and (implicit) complexity theory is constituted by the study of recursion. A foundational work in this area is due to Cobham [15], who gave a characterization of the class of poly-time computable functions **FP**, relying on a restricted recursion mechanism, called bounded recursion on notation (BRN). This groundbreaking result has inspired several characterizations based on *limited* recursion schemas for various other classes, but also alternative implicit ways to capture **FP**, for instance, via safe recursion [5] and ramification [25, 26].

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 10; pp. 10:1–10:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Cobham's work [15], together with other early results in recursion theory [20, 6, 29, 27], have even paved the way to recursion-theoretic characterizations for small circuit classes [11, 12, 16, 1, 14, 9]. Specifically, in [11, 12], an algebra based on the so-called *concatenation recursion on notation* schema (CRN) was introduced and shown able to capture functions computable in (**Dlogtime**-uniform) $\mathbf{AC}^0$ (i.e. computable by families of polynomial size and constant depth circuits) [11, 12]. This result was extended to $\mathbf{AC}^i$ and $\mathbf{NC}^i$ due to the notions of *k-bounded* and *weak bounded recursion on notation* [12]. A few years later, a similar function algebra to capture $\mathbf{TC}^0$ was introduced [14]. Other characterizations for subclasses of **NC** were independently presented in [16, 1, 9].

Related, but alternative, approaches to capture small circuit classes have also been provided in the framework of model- and proof-theory. In particular, it is well-known that there is an equivalence between $\mathbf{AC}^0$ and first-order logic, which naturally generalizes to extensions of the latter and larger circuit classes [23, 4, 22, 21, 28]. In [16], both a function algebra, based on so-called *upward recursion tree*, and a logical system to capture $\mathbf{NC}^1$ are presented. On the proof-theoretical side, in [1], together with the corresponding algebra, Allen defined a proof system *à la Buss* to capture **NC**. Another bounded theory to capture **NC** is introduced in [13] and then extended to several small circuit classes [14]. Theories for $\mathbf{TC}^0$ have been developed in [24] and, in the setting of second-order theories, in [18]. Alternative, proof-theoretical characterizations for $\mathbf{NC}^1$ were presented in [2], and, in the context of two-sorted theories, in [17].

A different descriptive approach to complexity, based on discrete ordinary differential equations (ODEs), was recently introduced in [10]. Informally speaking, its objective is to characterize functions computable in a given complexity class as solutions of a corresponding type of ODE. In this vein, in [10] a purely syntactic characterization of **FP** was given by linear systems of equations deriving along a logarithmically growing function. Intuitively, the latter condition controls the number of steps, while linearity controls the growth of objects generated during the computation. Recently, this approach has also been generalized to the continuous setting [7, 8].

Although small circuit classes have been characterized in multiple ways, it is an interesting and challenging question whether they can be studied through an ODE-based approach and whether this would shed some new light on these well-known classes. Interesting, because for a descriptive approach based on ODEs to make sense and be fruitful it has to be able to cope with very subtle and restricted modes of computation. Challenging, because even simple and useful mathematical functions may not be computable (e.g. multiplication is not in $\mathbf{AC}^0$), thus tools at hand and the naturalness of the approach are drastically restricted. In the present paper, we investigate these questions, and show that, in fact, natural ways to introduce ODE-oriented function algebras to capture small circuit classes can be found. Our approach relies on the introduction of ODE-schemas, still using derivation along the logarithmic function and allowing for bit shifting operations through restricted forms of linear equations. Our case study focuses on the smallest classes $\mathbf{AC}^0$ and $\mathbf{TC}^0$, but is intended as the first step towards a uniform characterization of other relevant classes in the **AC** and **NC** hierarchies.

Structure of the paper. The paper is divided into two main sections. In Section 2, we introduce notions and results at the basis of our ODE-style characterizations. In particular, in Section 2.1, we summarize salient aspects of the approach by [10], which we aim to generalize from the study of **FP** to that of small circuit classes. In Section 2.2, we briefly recap basic notions in parallel complexity and recall the function algebra approach of [11, 12, 14] to capture $\mathbf{AC}^0$ and $\mathbf{TC}^0$. In Section 3, we present our ODE-characterizations for the mentioned classes. Specifically, in Section 3.1, we introduce restricted ODE-schemas, and, using them, in Sections 3.2 and 3.3, we define ODE-function algebras capturing the analog for functions of $\mathbf{AC}^0$ and $\mathbf{TC}^0$, respectively. Then, in Section 3.4, we provide alternative and direct completeness proofs for both classes in the non-uniform setting, namely assuming that functions describing the circuits are given as basic functions. Finally, in Section 4, we briefly point at possible directions of future research.

2 Preliminaries

2.1 Capturing Complexity Classes via ODEs

We suppose the reader familiar with the basics of complexity theory [3, 30]. In order to introduce the approach to complexity delineated in [10], we start by cursorily recalling Cobham's result, capturing **FP** by the following BRN schema.

▶ **Definition 1** (Bounded Recursion on Notation, BRN). *Given $g : \mathbb{N}^p \to \mathbb{N}, k : \mathbb{N}^{p+1} \to \mathbb{N}$ and $h_i : \mathbb{N}^{p+2} \to \mathbb{N}$, with $i \in \{0,1\}$, $f : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by BRN from g, h_0, h_1 and k if*

$$f(0,\mathbf{y}) = g(\mathbf{y}) \qquad \text{and} \qquad f(\mathsf{s}_i(x),\mathbf{y}) = h_i(f(x,\mathbf{y}),x,\mathbf{y}), \text{ for } x \neq 0$$

with $f(x,\mathbf{y}) \leq k(x,\mathbf{y})$ and $\mathsf{s}_j(x) = 2x + j$ ($j \in \{0,1\}$) being the binary successor functions.

The growth of f is controlled by the function k (in **FP**), while the number of induction steps is kept under control by the application of the binary successor functions s_i. Such a schema is not fully satisfactory as it imposes an explicit bound on recursion in the form of an already known function.

As anticipated, Cobham's paper not only led to a variety of implicit characterizations for classes other that **FP**, but also inspired alternative approaches to capture this class. Among them, the proposal by [10] has the specificity of not imposing any explicit bound on the recursion schema or assigning specific role to variables. Instead, it is based on Discrete Ordinary Differential Equation (a.k.a. Difference Equations) and combines two peculiar features: derivation along specific functions, so to control the number of computation steps, and a special syntactic form of the equation itself (here linearity), allowing to control the object size. We present the basics of the method as necessary to formulate our new results.

Recall that the *discrete derivative of* $\mathbf{f}(x)$ is defined as $\Delta\mathbf{f}(x) = \mathbf{f}(x+1) + \mathbf{f}(x)$, and that ODEs are expressions of the form:

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x} = \mathbf{h}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}),$$

where $\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial x}$ stands for the derivative of $\mathbf{f}(x,\mathbf{y})$ considered as a function of x, for a fixed value for $\mathbf{y}$. When some initial value $\mathbf{f}(0,\mathbf{y}) = \mathbf{g}(\mathbf{y})$ is added, this is called *Initial Value Problem* (IVP). Then, in order to deal with complexity, some restrictions are needed. First, the notion of derivation has to be generalized, allowing to *derive along functions.*

▶ **Notation 1.** For $x \neq 0$, let $\ell(x)$ denote the length of x written in binary, i.e. $\lceil \log_2(x+1) \rceil$, and $\ell(0) = 0$. For $u \geq 0$, $\alpha(u) = 2^u - 1$ denotes the greatest integer the binary length of which is u. Also, we use $\div 2$ to denote integer division by 2, i.e. for all $x \in \mathbb{Z}$, $x \div 2 = \left\lfloor \frac{x}{2} \right\rfloor$.

▶ **Definition 2** (λ-ODE Schema). *Let $\mathbf{f}, \lambda : \mathbb{N}^p \to \mathbb{Z}$ and $\mathbf{h} : \mathbb{N}^{p+1} \to \mathbb{Z}$ be functions. Then,*

$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \lambda} = \frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \lambda(x,\mathbf{y})} = \mathbf{h}\big(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y}\big) \tag{1}$$

is a formal synonym of $\mathbf{f}(x+1,\mathbf{y}) = \mathbf{f}(x,\mathbf{y}) + \big(\lambda(x+1,\mathbf{y}) - \lambda(x,\mathbf{y})\big) \times \mathbf{h}(\mathbf{f}(x,\mathbf{y}),x,\mathbf{y})$. *When $\lambda(x,\mathbf{y}) = \ell(x)$, we call* (1) *a* length-ODE.

Intuitively, one of the key properties of the λ-ODE schema is its dependence on the number of distinct values of λ, i.e., the value of $\mathbf{f}(x, \mathbf{y})$ changes only when the value of $\lambda(x, \mathbf{y})$ does.

The computation of solutions of λ-ODEs have been fully described in [10]. Here, we focus on the special case of $\lambda = \ell$, which is particularly relevant for our characterizations. First, observe that the value of $\ell(x)$ changes (i.e. increases by 1) when x goes from $2^t - 1$ to 2^t, i.e. from $\alpha(t)$ to $\alpha(t) + 1$. So, if $\mathbf{f}$ is a solution of (1) with $\lambda = \ell$ and initial value $\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y})$, then $\mathbf{f}(1, \mathbf{y}) = \mathbf{f}(0, \mathbf{y}) + \mathbf{h}\big(\mathbf{f}(\alpha(0), \mathbf{y}), \alpha(0), \mathbf{y}\big)$, and, more generally, for all x and $\mathbf{y}$, $\mathbf{f}(x, \mathbf{y}) = \mathbf{f}(x-1, \mathbf{y}) + \Delta(\ell(x-1)) \times \mathbf{h}\big(\mathbf{f}(x-1, \mathbf{y}), x-1, \mathbf{y}\big) = \mathbf{f}(\alpha(\ell(x)-1), \mathbf{y}) + \mathbf{h}\big(\mathbf{f}\big(\alpha\big(\ell(x)-1\big), \mathbf{y}\big), \alpha(\ell(x)-1), \mathbf{y}\big)$, where $\Delta\big(\ell(t-1)\big) = \ell(t) - \ell(t-1)$. Starting from $t = x \geq 1$ and taking successive decreasing values of t, the first difference such that $\Delta(t) \neq 0$ is given by the biggest $t-1$ such that $\ell(t-1) = \ell(x) - 1$, i.e. $t - 1 = \alpha(\ell(x) - 1)$. Hence, by induction, it is established that:

$$\mathbf{f}(x, \mathbf{y}) = \sum_{u=-1}^{\ell(x)-1} \mathbf{h}\big(\mathbf{f}(\alpha(u), \mathbf{y}), \alpha(u), \mathbf{y}\big),$$

with $\mathbf{h}(\cdot, \alpha(-1), \mathbf{y}) = \mathbf{f}(0, \mathbf{y})$ and, as seen, $\alpha(u) = 2^u - 1$.

The second crucial novelty is the introduction of a special concept of *linearity*, utilized to control the growth of functions defined by ODEs. First, we present the notion of *degree for a polynomial expression*, which is a generalized (and slightly modified) version of the corresponding definition in [10]. Here, the degree of an expression is considered in relation to a set of variables, instead of a single one.

Let $\mathsf{sg} : \mathbb{Z} \to \mathbb{Z}$ be the sign function over $\mathbb{Z}$, taking value 1 for $x > 0$ and 0 otherwise.

▶ **Definition 3.** *A* sg-polynomial expression $P(x_1, \ldots, x_h)$ *is an expression built over the signature* $\{+, -, \times\}$*, the* sg *function and a set of variables* $X = \{x_1, \ldots, x_h\}$ *plus integer constants. Given a list of variables* $\mathbf{x} = x_{i_1}, \ldots, x_{i_m}$*, for* $i_1, \ldots, i_m \in \{1, \ldots, h\}$ *the* degree of a set $\mathbf{x}$ in a sg-polynomial expression P, $\deg(\mathbf{x}, P)$*, is inductively defined as follows:*

- $\deg(\mathbf{x}, P) = 0$ *for* P *constant*
- $\deg(\mathbf{x}, x_{i_j}) = 1$*, for* $x_{i_j} \in \{x_{i_1}, \ldots x_{i_m}\}$*, and* $\deg(\mathbf{x}, x_{i_j}) = 0$*, for* $x_{i_j} \notin \{x_{i_1}, \ldots, x_{i_m}\}$
- $\deg(\mathbf{x}, P + Q) = \deg(\mathbf{x}, P - Q) = \max\{\deg(\mathbf{x}, P), \deg(\mathbf{x}, Q)\}$
- $\deg(\mathbf{x}, P \times Q) = \deg(\mathbf{x}, P) + \deg(\mathbf{x}, Q)$
- $\deg(\mathbf{x}, \mathsf{sg}(P)) = 0.$

A sg-polynomial expression P is said to be *essentially constant in a set of variables* $\mathbf{x}$ when $\deg(\mathbf{x}, P) = 0$. A sg-polynomial expression P is said to be *essentially linear in a set* $\mathbf{x}$, when $\deg(\mathbf{x}, P) = 1$.

In what follows, we consider functions $\mathbf{f} : \mathbb{N}^{p+1} \to \mathbb{Z}^d$, i.e. vectors of functions $\mathbf{f} = f_1, ..., f_d$ from $\mathbb{N}^{p+1}$ to $\mathbb{Z}$, and we introduce the linear λ-ODE schema.

▶ **Definition 4** (Linear λ-ODE)**.** *Given* $\mathbf{g} : \mathbb{N}^p \to \mathbb{N}^d$*,* $\mathbf{h} : \mathbb{N}^{p+1} \to \mathbb{Z}$ *and* $\mathbf{u} : \mathbb{Z} \times \mathbb{N}^{p+1} \to \mathbb{Z}^d$*, the function* $\mathbf{f} : \mathbb{N}^{p+1} \to \mathbb{Z}^d$ *is* linear λ-ODE *definable from* $\mathbf{g}$*,* $\mathbf{h}$ *and* $\mathbf{u}$ *if it is the solution of the IVP with initial value* $\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y})$ *and such that:*

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \lambda} = \mathbf{u}\big(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}\big),$$

where $\mathbf{u}$ *is* essentially linear in $\mathbf{f}(x, \mathbf{y})$*. For* $\lambda = \ell$*, such schema is called* linear length ODE.

If $\mathbf{u}$ is *essentially linear in* $\mathbf{f}(x,\mathbf{y})$, there exist matrices $\mathbf{A}$ and $\mathbf{B}$, whose coefficients are essentially constant in $\mathbf{f}(x,\mathbf{y})$ and such that:

$$\mathbf{f}(0,\mathbf{y}) = \mathbf{g}(\mathbf{y})$$
$$\frac{\partial \mathbf{f}(x,\mathbf{y})}{\partial \ell} = \mathbf{A}\big(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y}\big) \times \mathbf{f}(x,\mathbf{y}) + \mathbf{B}\big(\mathbf{f}(x,\mathbf{y}),\mathbf{h}(x,\mathbf{y}),x,\mathbf{y}\big).$$

Then, for all x and $\mathbf{y}$,

$$\mathbf{f}(x,\mathbf{y}) = \sum_{u=-1}^{\ell(x)-1} \left(\prod_{t=u+1}^{\ell(x)-1} \Big(1+\mathbf{A}\big(\mathbf{f}(\alpha(t),\mathbf{y}),\mathbf{h}(\alpha(t),\mathbf{y}),\alpha(t),\mathbf{y}\big)\Big)\right) \times \mathbf{B}\big(\mathbf{f}(\alpha(u),\mathbf{y}),\mathbf{h}(\alpha(u),\mathbf{y}),\alpha(u),\mathbf{y}\big),$$

with the convention that $\prod_{x}^{x-1} \kappa(x) = 1$ and $\mathbf{B}(\cdot,\cdot,\alpha(-1),\mathbf{y}) = \mathbf{f}(0,\mathbf{y})$.

▶ **Example 5** (Function $2^{\ell(x)}$)**.** The function $x \mapsto 2^{\ell(x)}$ can be seen as the solution of the IVP with initial value $f(0) = 1$ and such that $\frac{\partial f(x)}{\partial \ell} = f(x)$, i.e. where $A = 1$, $B = 0$. Indeed, the solution of this system is of the form $f(x) = \prod_{u=0}^{\ell(x)-1} 2 = 2^{\ell(x)}$. In addition, the function $(x,y) : x,y \mapsto 2^{\ell(x)} \times y$ can be captured by the same equation, with initial value $f(0,y) = y$.

One of the main results of [10] is the implicit characterization of **FP** by the algebra made of basic functions $0, 1, \pi_i^p, \ell, +, -, \times, \mathsf{sg}$ and closed under composition ($\circ$) and ℓ-ODE:

$$\mathbb{LDL} = [0, 1, \pi_i^p, \ell, +, -, \times, \mathsf{sg}; \circ, \ell\text{-ODE}].$$

2.2 On Parallel Computation and Function Algebras for $\mathbf{FAC}^0$

Boolean circuits are vertex-labeled directed acyclic graphs whose nodes are either *input nodes* (no incoming edges), *output nodes* (no outgoing edges) or labeled with a Boolean function from the set $\{\wedge, \vee, \neg\}$. A Boolean circuit with majority gates allows in addition gates labeled by the function Maj, that outputs 1 when the majority of its inputs are 1's. A family of circuits (C_n) is **Dlogtime**-uniform if there is a Turing machine (with a random access tape) that decides in deterministic logarithmic time the *direct connection language of the circuit*, i.e. which, given 1^n, a, b and $t \in \{\wedge, \vee, \neg\}$, decides if a is of type t and b is a predecessor of a in the circuit (and analogously for input and output nodes). When dealing with circuits, the resources of interests are *size*, i.e. the number of its gates, and *depth*, i.e. the length of the longest path from the input to the output (see [30] for more details and related results).

▶ **Definition 6** (Classes $\mathbf{FAC}^i$ and $\mathbf{FTC}^i$)**.** *For $i \in \mathbb{N}$, the class $\mathbf{AC}^i$ (resp., $\mathbf{TC}^i$) is the class of languages recognized by a* **Dlogtime***-uniform family of Boolean circuits (resp. circuits including majority gates) of polynomial size and depth $O((\log n)^i)$. We denote by $\mathbf{FAC}^i$ and $\mathbf{FTC}^i$ the corresponding function classes.*

For $\mathbf{FAC}^0$, a particularly relevant recursion-theoretic characterization was provided by Clote [11, 12]. It relies on the schema of concatenation recursion on notation.

▶ **Definition 7** (Concatenation Recursion on Notation, CRN)**.** *A function f is defined by* concatenation recursion on notation *from g and h_i, with $i \in \{0,1\}$, if for all $x, \mathbf{y}$:*

$$f(0,\mathbf{y}) = g(\mathbf{y}) \quad \textit{and} \quad f(\mathsf{s}_i(x),\mathbf{y}) = \mathsf{s}_{h_i(x,\mathbf{y})}(f(x,\mathbf{y})), \textit{ for } x \neq 0.$$

Then, Clote's function algebra is defined as the class:

$$\mathcal{A}_0 = [0, \pi_i^p, \mathsf{s}_0, \mathsf{s}_1, \ell, \mathsf{BIT}, \#; \circ, \text{CRN}],$$

where $\mathsf{BIT}(x,y)$ returns the y^{th} value in the binary representation of x, and $x\#y = 2^{\ell(x)\times\ell(y)}$ is the smash function. This class was also proved able to capture $\mathbf{FAC}^0$ [11, 12].

▶ **Theorem 8** ([11, 12]). $\mathcal{A}_0 = \mathbf{FAC}^0$.

That $\mathcal{A}_0 \subseteq \mathbf{FAC}^0$ is proved by passing through the (function version of the) logarithmic time hierarchy **FLH**, which is known to be equivalent to $\mathbf{FAC}^0$. Then, $\mathcal{A}_0 \subseteq \mathbf{FLH}$ is established by showing that basic functions are computable in log-time and that **FLH** is closed under composition and CRN. That $\mathbf{FLH} \subseteq \mathcal{A}_0$ is proved by the arithmetization of log-time bounded RAM. Remarkably, in [14], these results are even generalized to $\mathbf{FTC}^0$, which is proved equivalent to the function algebra:

$$\mathcal{TC}_0 = [0, \pi_i^p, \mathsf{s}_0, \mathsf{s}_1, \ell, \mathsf{BIT}, \times, \#; \circ, \mathrm{CRN}].$$

3 Towards an ODE-Characterization of FAC^0

In this section, we provide the first implicit characterization of $\mathbf{FAC}^0$ in the ODE-setting. We start by introducing the new ODE-schemas which are at the basis of our characterizations of $\mathbf{FAC}^0$ and $\mathbf{FTC}^0$ and intuitively corresponding to left- and right-shifting (Sec. 3.1). Due to them, we introduce the function algebra $\mathbb{ACDL}$, the defining feature of which is precisely the presence of these special ODE-schemas, and prove this class able to capture $\mathbf{FAC}^0$ (Sec. 3.2). This is established passing through Clote's $\mathcal{A}_0$. As a byproduct, we obtain a similar ODE-characterization for $\mathbf{FTC}^0$ (Sec. 3.3). Finally, an alternative, direct proof of completeness is provided for both classes in a non-uniform setting (Sec. 3.4).

▶ Remark 9. Here, functions can take images in $\mathbb{Z}$. Accordingly, a convention for the binary representation of integers must be adopted, e.g. by assuming that, in any binary sequence, the first bit indicates the sign. Then, all arithmetic operations can be easily re-designed to handle encodings of possibly negative integers by circuits of the same size and depth.

3.1 Discrete ODE-Schemas for Shifting

We start by considering the ODE-schemas which are at the basis of our characterizations of $\mathbf{FAC}^0$ and $\mathbf{FTC}^0$. Observe that they will sometimes include $\times$. This is admissible since, as we shall see, the "kind of multiplication" we consider is actually limited to special cases (namely, multiplication by 2^i), which are proved to be computable in $\mathbf{FAC}^0$.

The ℓ-ODE$_1$ and ℓ-ODE$_2$ Schemas. We start with the limited ℓ-ODE$_1$ schema, intuitively corresponding to left shifting(s) and (possibly) adding a bit.

▶ **Definition 10** (ℓ-ODE$_1$ Schema). *Given $g : \mathbb{N}^p \to \mathbb{N}$ and $h : \mathbb{N}^{p+1} \to \mathbb{N}$, such that h takes values in $\{0,1\}$ only, the function $f : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by ℓ-ODE$_1$ from functions g and h when it is the solution of the IVP with initial value $f(0,\mathbf{y}) = g(\mathbf{y})$ and such that:*

$$\frac{\partial f(x,\mathbf{y})}{\partial \ell} = f(x,\mathbf{y}) + h(x,\mathbf{y}).$$

The definition of the function $2^{\ell(x)}$ (and $2^{\ell(x)} \times y$) given in Example 5 is a special case of a ℓ-ODE$_1$ (with $h(x,\mathbf{y}) = 0$).

▶ Remark 11. An equivalent, purely-syntactical formulation of Definition 10 is obtained by substituting the explicit constraint that $h(x,\mathbf{y}) \in \{0,1\}$ with the assumption that $h(x,\mathbf{y})$ is of the form $\mathsf{sg}\big(B(h_1,\ldots,h_m,x,\mathbf{y})\big)$, where B is an expression built over the signature $\{+,-,\div 2,\mathsf{sg}\}$ and calling previously defined $\mathbf{FAC}^0$ functions $h_1(x,\mathbf{y}),\ldots,h_m(x,\mathbf{y})$.

In terms of circuits, this schema intuitively allows us to iteratively left-shifting (the binary representation of) a given number, each time possibly adding 1 to its final position. This is clarified by the proof below, establishing that $\mathbf{FAC}^0$ is closed under the mentioned schema.

▶ **Proposition 12.** *If f is defined by ℓ-ODE$_1$ from g and h in $\mathbf{FAC}^0$, then f is in $\mathbf{FAC}^0$.*

Proof Sketch. By Def. 10, for all x and $\mathbf{y}$: $f(x, \mathbf{y}) = \sum_{u=-1}^{\ell(x)-1} \Big(\prod_{t=u+1}^{\ell(x)-1} 2\Big) \times h\big(\alpha(u), \mathbf{y}\big) = \sum_{u=-1}^{\ell(x)-1} 2^{\ell(x)-u-1} \times h(\alpha(u), \mathbf{y})$, with the convention that $\alpha(u) = 2^u - 1$, $\prod_x^{x-1} \kappa(x) = 1$ and $h\big(\alpha(-1), \mathbf{y}\big) = f(0, \mathbf{y})$. Clearly, the multiplication here involved is always by a power of 2 (which basically corresponds to left-shifting, so is computable in $\mathbf{FAC}^0$). Since $h(x, \mathbf{y}) \in \{0, 1\}$, the outermost sum amounts to a concatenation (which again can be computed in $\mathbf{FAC}^0$). ◀

Notice that this schema is not as weak as it may seem since, together with sg, it is enough to express bounded quantification.

▶ Remark 13. Let $R \subseteq \mathbb{N}^{p+1}$ and h_R be its characteristic function. Then, for all x and $\mathbf{y}$, it holds that $(\exists z \leq \ell(x))R(z, \mathbf{y}) = \mathsf{sg}\big(f(x, \mathbf{y})\big)$, where f is such that $f(0, \mathbf{y}) = h_R(0, \mathbf{y})$ and

$$\frac{\partial f(x, \mathbf{y})}{\partial \ell} = f(x, \mathbf{y}) + h_R\big(\ell(x+1), \mathbf{y}\big).$$

Clearly, $f(x, \mathbf{y})$ is an instance of ℓ-ODE$_1$. Intuitively, $f(x, \mathbf{y}) \neq 0$ when, for some z smaller than x, $R(z, \mathbf{y})$ is satisfied (i.e. $h_R(z, \mathbf{y}) = 1$): when such instance exists, our bounded search ends with a positive answer. Universally bounded quantification can be expressed in a similar way, considering $\mathsf{cosg}(f(x, \mathbf{y}))$, where f is defined substituting the value of h_R with its co-sign (such that, $\mathsf{cosign}(x) = 1 - \mathsf{sg}(x)$).

The more general schema ℓ-ODE$_2$ allows multiple left-shifting, such that each "basic operation" corresponds to shifting a given value of a number of digits determined by $\ell(k(\mathbf{y}))$.

▶ **Definition 14** (ℓ-ODE$_2$ Schema). *Given $g : \mathbb{N}^p \to \mathbb{N}$, $h : \mathbb{N}^{p+1} \to \mathbb{N}$ and $k : \mathbb{N}^p \to \mathbb{N}$, the function $f : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by ℓ-ODE$_2$ from g, h and k if it is the solution of the IVP with initial value $f(0, \mathbf{y}) = g(\mathbf{y})$ and such that:*

$$\frac{\partial f(x, \mathbf{y})}{\partial \ell} = \big(2^{\ell(k(\mathbf{y}))} - 1\big) \times f(x, \mathbf{y}) + h(x, \mathbf{y}),$$

where $h(x, \mathbf{y}) \in \{0, 1\}$, and if, for some x and $\mathbf{y}$, $h(x, \mathbf{y}) \neq 0$, then $k(\mathbf{y}) \neq 0$.

Since this schema is introduced to characterize $\mathbf{FAC}^0$, the constraint imposing $k(\mathbf{y}) \neq 0$, when at some point $h(x, \mathbf{y})$ takes value 1, is really essential. Indeed, as we shall see (Sec. 3.3), if we omit it, ℓ-ODE$_2$ will be too strong, as able to capture binary counting (which is not in $\mathbf{FAC}^0$).

Observe that ℓ-ODE$_1$ is a special case of ℓ-ODE$_2$, such that $\ell(k(\mathbf{y})) = 1$, and that, also for it, the following closure result holds.

▶ **Proposition 15.** *If f is defined by ℓ-ODE$_2$ from $\mathbf{FAC}^0$-functions g, k and h, then f is in $\mathbf{FAC}^0$.*

Proof Sketch. There are two cases: if $k(\mathbf{y}) \neq 0$, the proof is analogous to that of Prop. 12; if $k(\mathbf{y}) = 0$ (and $h(x, \mathbf{y}) = 0$), for all x and $\mathbf{y}$, $f(x, \mathbf{y}) = g(\mathbf{y})$, in $\mathbf{FAC}^0$ by hypothesis. ◀

The Schema ℓ-ODE$_3$. Let us now consider ℓ-ODE$_3$, intuitively corresponding to (basic) right-shifting operations.

▶ **Definition 16** (ℓ-ODE$_3$ Schema). *Given $g : \mathbb{N}^p \to \mathbb{N}$, the function $f : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by ℓ-ODE$_3$ from g if it is the solution of the IVP with initial value $f(0, \mathbf{y}) = g(\mathbf{y})$ and such that:*

$$\frac{\partial f(x, \mathbf{y})}{\partial \ell} = -\left\lceil \frac{f(x, \mathbf{y})}{2} \right\rceil$$

where $\lceil \frac{z}{2} \rceil$ is a shorthand for $z - (z \div 2)$.

▶ **Proposition 17.** *If f is defined by ℓ-ODE$_3$ from g in $\mathbf{FAC}^0$, then f is in $\mathbf{FAC}^0$ as well.*

Proof Sketch. The proof is similar to the one for Prop. 12. By Def. 16 and since $(a \div 2^b) \div 2 = a \div 2^{b+1}$, it can be shown by induction that for all x and $\mathbf{y}$: $f(x, \mathbf{y}) = g(\mathbf{y}) \div 2^{\ell(x)-1}$. This intuitively corresponds to right-shifting $g(\mathbf{y})$ a number of times equal to $\ell(x) - 1$ and can be easily shown computable by a constant-depth circuit. ◀

3.2 An ODE-Characterization of $\mathbf{FAC}^0$

We now define a new class of functions, crucially relying on the ODE-schemas just introduced:

$$\mathbb{ACDL} = [0, 1, \pi_i^p, \ell, +, -, \div 2, \mathsf{sg}; \circ, \ell\text{-ODE}_2, \ell\text{-ODE}_3].$$

Observe that all its basic functions and (restricted) schemas are natural in the context of differential equations and calculus. In $\mathbb{ACDL}$, multiplication is, of course, not allowed. Compared to $\mathbb{LDL}$, the linear-length ODE schema is substituted by the two schemas ℓ-ODE$_2$ and ℓ-ODE$_3$, characterized by a very limited form of "multiplication" and intuitively allowing to capture left and right shifting.

In order to prove that $\mathbb{ACDL}$ captures $\mathbf{FAC}^0$ we start by providing an *indirect* proof that $\mathbf{FAC}^0 \subseteq \mathbb{ACDL}$. This is established by showing that any basic function and schema defining Clote's $\mathcal{A}_0$ (so its arithmetization of log-time bounded RAM) can be simulated in our setting by functions and schemas of $\mathbb{ACDL}$. Preliminarily, observe that some important operations "come for free" by composition. For instance, the modulo 2 operation is defined as $(x \bmod 2) = x - \lfloor \frac{x}{2} \rfloor - \lfloor \frac{x}{2} \rfloor$, while binary successor functions are expressed in our setting as $\mathsf{s}_0(x) = x + x$ and $\mathsf{s}_1(x) = \mathsf{s}_0(x) + 1$ (being the constant 0 and $+$ basic functions of $\mathbb{ACDL}$).

The smash function $2^{\ell(x)\times\ell(y)}$. The smash function $x\#y : x, y \mapsto 2^{\ell(x)\times\ell(y)}$ is rewritten as the solution of the IVP defined by the initial value $f(0, y) = 1$ and such that $\frac{\partial f(x,y)}{\partial \ell} = (2^{\ell(y)} - 1) \times f(x)$. This is clearly an instance of ℓ-ODE$_2$, such that $g(\mathbf{y}) = 1$, $k(\mathbf{y}) = y$ and $h(x, \mathbf{y}) = 0$. Recall that, since $h(x, \mathbf{y}) = 0$, even the limit case of $y = 0$ is properly captured.

The BIT function. Intuitively, the function $\mathsf{BIT}(x, y)$ returns the y^{th} bit in the binary representation of x. In order to capture it, a series of auxiliary functions are needed:

- the *log most significant part function* $\mathsf{msp}(x, y) : x, y \mapsto \left\lfloor \frac{y}{2^{\ell(x)}} \right\rfloor$, which can be rewritten via ℓ-ODE$_3$,
- the *basic conditional function* $\mathsf{if}(x, y, z)$, returning y if $x = 0$ and z otherwise, can be rewritten in our setting by composition, using, in particular, the "shift function" $2^{\ell(x)} \times y$ (defined using ℓ-ODE$_1$),

- the *special bit function* $\mathsf{bit}(x, y)$, returning 1 when the $\ell(y)^{th}$ bit of x is 1, can be rewritten in $\mathbb{ACDL}$ due to msp,
- the *bounded exponentiation function* $\mathsf{bexp}(x, y)$, that, for any $y \leq \ell(x)$, returns 2^y, can be obtained relying on functions in $\mathbb{ACDL}$, including, in particular, msp, if and $2^{\ell(\cdot)}$ together with ODE-schemas.

Then, using bexp and bit, the desired BIT function can be rewritten in our setting by composition: $\mathsf{BIT}(x, y) = \mathsf{bit}(x, \mathsf{bexp}(x, y) - 1)$.

The CRN Schema. A function f defined by CRN from g, h_0 and h_1 can be simulated in $\mathbb{ACDL}$ via the ℓ-ODE$_1$ schema. Let us consider the IVP with initial value $F(0, x, \mathbf{y}) = g(\mathbf{y})$ and such that:

$$\frac{\partial F(t, x, \mathbf{y})}{\partial \ell(t)} = F(t, x, \mathbf{y}) + h(t + 1, x, \mathbf{y})$$

where $h(t, x, \mathbf{y}) \in \{0, 1\}$ is, in turn, defined as:

$$\mathsf{if}\big(\mathsf{bit}(x, 2^{\ell(x)-\ell(t)} - 1), h_0(\mathsf{msp}(2^{\ell(x)-\ell(t)}, x), \mathbf{y}), h_1(\mathsf{msp}(2^{\ell(x)-\ell(t)}, x), \mathbf{y})\big).$$

The function $F(t, x, \mathbf{y})$ is clearly an instance of ℓ-ODE$_1$, and $h(t, x, \mathbf{y})$ is defined by composition from functions proved to be in $\mathbb{ACDL}$. Then, we set $f(x, \mathbf{y}) = F(x, x, \mathbf{y})$.

We now have all the ingredients to prove our main result.

▶ **Theorem 18.** $\mathbb{ACDL} = \mathbf{FAC}^0$.

Proof. $\mathbb{ACDL} \subseteq \mathbf{FAC}^0$. All basic functions of $\mathbb{ACDL}$ are computable in $\mathbf{FAC}^0$. Moreover, the class is closed under composition and, by Prop. 15 and 17, under ℓ-ODE$_2$ and ℓ-ODE$_3$. $\mathbf{FAC}^0 \subseteq \mathbb{ACDL}$ since, as we just proved, functions and schemas constituting $\mathcal{A}_0$ have been rewritten in $\mathbb{ACDL}$. ◀

A careful analysis of the above definitions shows that ℓ-ODE$_2$ is actually used only to capture the smash # function. One obtains a class equivalent to $\mathbb{ACDL}$ by allowing # and by replacing ℓ-ODE$_2$ with the simpler ℓ-ODE$_1$ schema.

▶ **Corollary 19.** $\mathbf{FAC}^0 = [0, 1, \pi_i^p, \ell, +, -, \div 2, \mathsf{sg}, \#; \circ, \ell\text{-}ODE_1, \ell\text{-}ODE_3]$

3.3 An ODE-Characterization of $\mathbf{FTC}^0$

As a byproduct, an ODE-characterization for $\mathbf{FTC}^0$ is easily obtained, this time passing through $\mathcal{TC}_0$ [14]. We consider an extension of $\mathbb{ACDL}$ endowed with the basic function ×:

$$\mathbb{TCDL} = [0, 1, \pi_i^p, \ell, +, -, \div 2, \times, \mathsf{sg}; \circ, \ell\text{-ODE}_2, \ell\text{-ODE}_3].$$

▶ **Proposition 20.** *Let f be defined by ℓ-ODE$_2$ from functions in $\mathbf{FTC}^0$. Then, f is in $\mathbf{FTC}^0$.*

Proof Sketch. The proof is similar to that of Prop. 15. The main difference concerns the computation of level 0, i.e. that of the initial values $g(\mathbf{y})$ and $h(x, \mathbf{y})$, which are now expressions possibly including ×. This does not affect the overall structure of the circuit. ◀

▶ **Proposition 21.** *Let f be defined by ℓ-ODE$_3$ from functions in $\mathbf{FTC}^0$. Then, f is in $\mathbf{FTC}^0$.*

Proof Sketch. Straightforward generalization of Prop. 17, with the same provisos of Prop. 20. ◀

Then, the desired characterization easily follows from Propositions 20 and 21 and from [14], rewritten in our ODE-setting (see Sec. 3.2).

▶ **Theorem 22.** $\mathbb{TCDL} = \mathbf{FTC}^0$.

An alternative characterization of $\mathbf{FTC}^0$ is obtained by considering the following class:

$$\mathbb{TCDL}^* = [0, 1, \pi_i^p, \ell, +, -, \div 2, \mathsf{sg}; \circ, \ell\text{-ODE}_2^*, \ell\text{-ODE}_3].$$

where $\mathbb{TCDL}^*$ does not include $\times$ as a basic function, but allows the generalized schema ℓ-ODE$_2^*$, with no constraint over the function k:

▶ **Definition 23** (ℓ-ODE$_2^*$ Schema). *Let $g : \mathbb{N}^p \to \mathbb{N}, h : \mathbb{N}^{p+1} \to \mathbb{N}$ and $k : \mathbb{N}^p \to \mathbb{N}$, where h takes values in $\{0, 1\}$. Then, the function $f : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by ℓ-ODE$_2^*$ from g, h and k when it is the solution of the IVP with initial value $f(0, \mathbf{y}) = g(\mathbf{y})$ and such that:*

$$\frac{\partial f(x, \mathbf{y})}{\partial \ell} = \left(2^{\ell(k(\mathbf{y}))} - 1\right) \times f(x, \mathbf{y}) + h(x, \mathbf{y}).$$

▶ **Example 24** (bcount). Observe that if $k(\mathbf{y}) = 0$, then ℓ-ODE$_2^*$ is enough to express the binary counting function $\mathsf{bcount}(x)$, that outputs the sum of the bits of x. Indeed, $\mathsf{bcount}(x) = f(x, x)$ where f is the solution of the IVP with initial value $f(0, \mathbf{y}) = \mathsf{bit}(0, y)$ and such that:

$$\frac{\partial f(x, \mathbf{y})}{\partial \ell} = \mathsf{bit}(x, \mathbf{y}).$$

Since such function is not in $\mathbf{FAC}^0$ (see [19]), this also illustrates that ℓ-ODE$_2^*$ is really more expressive than ℓ-ODE$_2$.

It is easy to see that ℓ-ODE$_2^*$ is enough to capture majority (and to "simulate" multiplication, see [30]), which is the essential step to show that $\mathbb{TCDL}^* = \mathbf{FTC}^0$.

This observation, together with the fact that what really makes ℓ-ODE$_2^*$ more expressive than ℓ-ODE$_2$ is its behavior for $k(\mathbf{y}) = 0$, leads us to an alternative characterization for $\mathbf{FTC}^0$, in line with the one of Corollary 19. Let's consider the following schema.

▶ **Definition 25** (ℓ-ODE$_1^*$ Schema). *Let $g : \mathbb{N}^p \to \mathbb{N}$ and $h, k : \mathbb{N}^{p+1} \to \{0, 1\}$. Then the function $f : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by ℓ-ODE$_1^*$ from g, h and k, when it is the solution of the IVP with initial value $f(0, \mathbf{y}) = g(\mathbf{y})$ and such that:*

$$\frac{\partial f(x, \mathbf{y})}{\partial \ell} = k(x, \mathbf{y}) \times f(x, \mathbf{y}) + h(x, \mathbf{y}).$$

By straightforward inspection and Example 24, it is easily seen that $\mathbf{FTC}^0$ is again captured by adding the basic function # and by replacing ℓ-ODE$_2^*$ with the simpler schema ℓ-ODE$_1^*$.

▶ **Corollary 26.** $\mathbf{FTC}^0 = \mathbb{TCDL}^* = [0, 1, \pi_i^p, \ell, +, -, \div 2, \mathsf{sg}, \#; \circ, \ell\text{-}ODE_1^*, \ell\text{-}ODE_3]$.

3.4 Alternative Direct Proofs

In this section, we introduce alternative classes $\mathbb{ACDL}_{\mathcal{C}}$ and $\mathbb{TCDL}_{\mathcal{C}}$, (resp.) extending $\mathbb{ACDL}$ and $\mathbb{TCDL}$ with new basic functions that arithmetize the circuit families of polynomial size and constant depth $(C_n)_{n \geq 0}$ used for computation. In this non-uniform context, we prove both $\mathbf{FAC}^0 \subseteq \mathbb{ACDL}_{\mathcal{C}}$ (Sec. 3.4.1) and $\mathbf{FTC}^0 \subseteq \mathbb{TCDL}_{\mathcal{C}}$ (Sec. 3.4.2) *directly*, i.e. without any references to results in [11, 14], .

3.4.1 Direct Completeness for $\mathbb{ACDL}_{\mathcal{C}}$

Let $\mathcal{C} = (C_n)_{n \geq 0}$ be a class of circuits of polynomial size n^k, for some $k \in \mathbb{N}$, and constant depth d. We assume that each circuit C_n is in a special normal form, such that it strictly alternates between $\wedge$ and $\vee$ (and edges are only between gates of consecutive layers): input gates are all at level 0, negation gates are all at level 1, even levels are all $\wedge$ gates, odd levels (other than 1) are all $\vee$ gates, and the depth d is even (so output gates are $\wedge$ gates).

In this context we keep the basic functions of $\mathbb{ACDL}$, but add a set $\mathbf{circ}_{\mathcal{C}} = \{C, L_0^{in}, L_0^{\neg}, L_e\}$ of characteristic functions associated to the following predicates. The predicate $C \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ describes the underlying graph of the circuit: for any integers x, α, β, $(x, \alpha, \beta) \in C$ when, in $C_{\ell(x)}$, the $\alpha^{th} \leq \ell(x)^k$ gate of some level is a predecessor of the $\beta^{th} \leq \ell(x)^k$ gate of the next level (thus, in this encoding, α and β are exponentially smaller than x). The relations $L_0^{in}, L_0^{\neg}, L_e \subseteq \mathbb{N} \times \mathbb{N}$, for $e \in \{1, \dots, d\}$, describe the level of gates (and, implicitly, their type): L_0^{in} refers to input gates, $L_0^{\neg}$ to negation gates, and L_e to $\wedge$ and $\vee$ gates, depending on e being odd or even. Since we aim at defining functions over integers, we assume that input gates are numbered from $n-1$ to 0 and output gates from $m-1$ to 0, with $m \leq \ell(x)^k$. By considering the functions corresponding to the given relations, we obtain the desired ODE-style family of classes (parameterized by $\mathcal{C}$):

$$\mathbb{ACDL}_{\mathcal{C}} = [0, 1, \pi_i^p, \ell, +, -, \div 2, \mathsf{sg}, \mathbf{circ}_{\mathcal{C}}; \circ, \ell\text{-ODE}_2, \ell\text{-ODE}_3].$$

▶ Remark 27. If the family $\mathcal{C}$ is **Dlogtime**-uniform, then the functions in $\mathbf{circ}_{\mathcal{C}}$ are computable in $\mathcal{A}_0$. Consequently, in this case, it holds that $\mathbb{ACDL}_{\mathcal{C}} = \mathbb{ACDL}$.

In this non-necessarily uniform context, a completeness proof still holds.

▶ **Proposition 28.** *If a function $f : \mathbb{N} \to \mathbb{N}$ is computable by a family $\mathcal{C} = (C_n)_{n \geq 0}$ of polynomial size and constant-depth circuits, then it is in $\mathbb{ACDL}_{\mathcal{C}}$.*

To prove it we need the following lemma (which is easily established relying on Remark 13).

▶ **Lemma 29.** *Let g and h be functions computable in $\mathbb{ACDL}_{\mathcal{C}}$ and $k \in \mathbb{N}$. Then, the function $\mathsf{min}_{i \leq \ell(x_1)^k}\{g(i, \mathbf{x}) : h(i, \mathbf{x}) \rhd j\}$, for $\rhd \in \{<, \leq, >, \geq, =\}$ and $j \in \{0, 1\}$, is in $\mathbb{ACDL}_{\mathcal{C}}$.*

Proof of Prop. 28. Let $\mathsf{Eval}(t, x)$ be a function that returns the value of the t^{th} output gate of the circuit $C_{\ell(x)}$ of input x when $t \leq m-1$ (and $\mathsf{Eval}(t, x) = 0$ for $t > m-1$). Then, the following expression defines a function f such that $f(2^{\ell(x)^k}, x)$ outputs the value of the computation of C_n (for $n = \ell(x)$) on input x:

$$\frac{\partial f(y, x)}{\partial \ell(y)} = f(y, x) + \mathsf{Eval}(\ell(x) - \ell(y) - 1, x)$$

with $f(0, x) = 0$. Intuitively, the function above computes the successive suffixes of the output word, starting from the bits of bigger weights. Remarkably, this is an instance of the ℓ-ODE$_1$ schema (indeed, $\mathsf{Eval}(y, x) \in \{0, 1\}$). So, the given f can be rewritten in $\mathbb{ACDL}_{\mathcal{C}}$.

It remains to describe how the function $\mathsf{Eval}(t, x)$ is computed. Again, we assume that C_n has depth d, is in the normal form described above, and d is even. Concretely, we start by defining a special (bounded) minimum operator function such that, given $k \in \mathbb{N}$ and two functions g and h, with $h(t, \mathbf{x}) \in \{0, 1\}$ for $t \in \mathbb{N}$ and $\mathbf{x} = x_1, \dots, x_h$,

$$\mathsf{min}_{i \leq \ell(x_1)^k}\{g(i, \mathbf{x}) : h(i, \mathbf{x}) \rhd 0\},$$

with $\rhd \in \{<, \leq, >, \geq, =\}$ and $j \in \{0, 1\}$. Intuitively, given $i \in \{0, \dots, \ell(x_1)\}$, this function computes the minimum of the values of $g(i, \mathbf{x})$, for i and $\mathbf{x}$ such that $h(i, \mathbf{x}) \rhd j$.

The inductive definition of Eval relies on those of the $d+1$ functions $\mathsf{Eval}_0, \dots, \mathsf{Eval}_d$, with $\mathsf{Eval}_d = \mathsf{Eval}$:

- $\mathsf{Eval}_0(t,x)$ is equal to $\mathsf{BIT}(t,x)$ if $L_0^{in}(t,x)$ holds and to $1-\mathsf{BIT}(t,x)$ if $L_0^{\neg}(t,x)$ does. For t not corresponding to gate index, $\mathsf{Eval}_0(t,x)$ is set to an arbitrary value, say 0. Recall that, since BIT can be rewritten in $\mathbb{ACDL}$ (Sec. 3.2), Eval_0 is in $\mathbb{ACDL}_{\mathcal{C}}$ as well.
- $\mathsf{Eval}_{2e}(y,x)$ is equal to $\mathsf{min}_{i\leq\ell(x)^k}\{\mathsf{Eval}_{2e-1}(i,x) : C(x,i,t)=1\}$, for $L_{2e}(t,x)$ (i.e., if t is the index of a gate at this level). The evaluation for the i^{th} gate of the level $2e$ (a $\wedge$-gate) is the minimum of the evaluations of its predecessor gates of level $2e-1$. As seen, min is in $\mathbb{ACDL}$ (Lemma 29), so Eval_{2e} can also be rewritten in this class.
- Similarly, $\mathsf{Eval}_{2e+1}(y,x)$ is the $1-\mathsf{min}_{i\leq\ell(x)^k}\{1-\mathsf{Eval}_{2e}(i,x) : C(x,i,t)=1\}$. The evaluation for the t^{th} gate of level $2e+1$ (a $\wedge$-gate) is the maximum among evaluations of its predecessor gates of level $2e$. As for Eval_{2e}, Eval_{2e+1} can be rewritten in $\mathbb{ACDL}$. ◀

▶ Remark 30. The following converse to Proposition 28 also holds (by inspecting the proofs of Proposition 15 and 17): If a function $f:\mathbb{N}\to\mathbb{N}$ is in $\mathbb{ACDL}_{\mathcal{C}}$ for some family $\mathcal{C}$ of polynomial size and constant-depth circuits, then there exists a family $\mathcal{C}'$ of polynomial size and constant-depth circuits that computes it.

3.4.2 Direct Completeness for FTC^0

Let us now consider the similar, direct characterization for $\mathbf{FTC}^0$. Suppose that C_n strictly alternates between $\wedge,\vee$ and Maj gates, and that input gates and their negation are all at level 0, $\vee$ gates are at levels $3e+1$, $\wedge$ gates at levels $3e+2$, and Maj gates at levels $3e$. Accordingly, in this case L_e describes the level of a gate of a type not limited to $\wedge,\vee$, but including Maj. Then, the desired family of classes is defined as:

$$\mathbb{TCDL}_{\mathcal{C}} = [0, 1, \pi_i^p, \ell, +, -, \div 2, \mathsf{sg}, \mathbf{circ}_{\mathcal{C}}; \circ, \ell\text{-ODE}_2^*, \ell\text{-ODE}_3]$$

where, with a slight abuse of notation, we use $\mathbf{circ}_{\mathcal{C}}$ to denote the function corresponding to a (set of) relation(s), this time including the *extended* L_e. The proof that non-uniform $\mathbf{FTC}^0 \subseteq \mathbb{TCDL}_{\mathcal{C}}$ is similar to the one from Section 3.4.1.

▶ **Proposition 31.** *If a function $f:\mathbb{N}\to\mathbb{N}$ is computable by a family $\mathcal{C}=(C_n)_{n\geq 0}$ of polynomial-size and constant-depth circuits including* Maj *gates, then it is in $\mathbb{TCDL}_{\mathcal{C}}$.*

Proof Sketch. The functions f and Eval are globally defined as before. Modifications only affect the definition of Eval_d and, in particular, the inductive levels corresponding to Maj. Specifically, it is obtained as follows:

- for a given function h and integer k, $\mathsf{bcount}_h(t,x)=\sum_{i\leq\ell(x)^k} h(i,t,x)$. Notice that this function is in $\mathbb{TCDL}_{\mathcal{C}}$, since it can be rewritten as an instance of ℓ-ODE$_2^*$, and h is in $\mathbb{TCDL}^*$ (due to Lemma 29),
- for any i such that $L_{3e-1}(i)$ and $3e-1<d$: $v^0_{3e-1}(i,t,x)=\mathsf{sg}(C(x,i,t))$ and $v^1_{3e-1}(i,t,x)=\mathsf{if}(C(x,i,t),\mathsf{Eval}_{3e-1(i,x)},0)$. The value of $v^0_{3e-1}(i,t,x)$ is 1 when i is a predecessor of gate t, and $v^1_{3e-1}(i,t,x)$ is 1 when, in addition, the value of gate i, on input x, is 1,
- for t such that $L_{3e}(t)$, $\mathsf{Eval}_{3e}(t,x)$ is defined as $\mathsf{sg}\big(\mathsf{bcount}_{v^0_{3e-1}}(t,x)-2\times\mathsf{bcount}_{v^1_{3e-1}}(t,x)\big)$. The function outputs 1 when more than half of the inputs of gate t are 1. ◀

4 Conclusion

We have presented new characterizations for $\mathbf{FAC}^0$ and $\mathbf{FTC}^0$ through the prism of discrete differential equations. Although the use of classical arithmetical functions is intrinsically limited by the low computational power of these classes, the ODEs used are surprisingly natural restrictions of linear ODEs. More generally, this work is intended as the first step of a project aiming at characterizing other relevant classes, starting with $\mathbf{FAC}^k$ and $\mathbf{FNC}^k$. Another challenging direction of future research would be to develop logical and proof-theoretical counterparts to ODE-style algebras, e.g. by defining *natural* rule systems (oriented by the ODE design) to syntactically characterize the corresponding classes.

References

1 B. Allen. Arithmetizing uniform NC. *Ann. Pure Appl. Logic*, 53:1–50, 1991.

2 T. Arai. A bounded arithmetic AID for Frege systems. *Ann. Pure Appl. Logic*, 103:155–199, 2000.

3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2007.

4 D.A.M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. of Comput. and Syst. Sc.*, 41:274–306, 1990.

5 S. Bellantoni and S. Cook. A new recursion-theoretic characterization of poly-time functions. *Comput. Complex.*, 2:97–110, 1992.

6 J.H. Bennett. *On Spectra*. PhD thesis, Princeton University, 1962.

7 M. Blanc and O. Bournez. A characterisation of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. In *Proc. MCU*, pages 58–74, 2022.

8 M. Blanc and O. Bournez. A characterisation of functions computable in polynomial time and space over the reals with discrete ordinary differential equations: simulation of Turing machines with analytic discrete ODEs. In *Proc. MFCS*, pages 21:1–21:15, 2023.

9 G. Bonfante, R. Kahle, J.-Y. Marion, and I. Oitavem. Two function algebras defining functions in NC^k boolean circuits. *Inf. and Comput.*, 2016.

10 O. Bournez and A. Durand. A characterization of functions over the integers computable in polynomial time using discrete differential equations. *Comput. Complex.*, 32(7), 2023.

11 P.G. Clote. A sequential characterization of the parallel complexity class NC. Technical report, Boston College, 1988.

12 P.G. Clote. Sequential, machine-independent characterizations of the parallel complexity classes AlogTIME, AC^k, NC^k and NC. In Buss S.R. and Scott P.J., editors, *Feasible Mathematics: A Mathematical Sciences Institute Workshop, Ithaca, New York, June 1989*, Progress in Computer Science and Applied Logic, pages 49–69. Birkhäuser, Boston, MA, 1990.

13 P.G. Clote and G. Takeuti. Bounded arithmetic for NC, ALogTIME, L and NL. *Ann. Pure and Appl. Logic*, 56:73–117, 1992.

14 P.G. Clote and G. Takeuti. First order bounded arithmetic and small complexity classes. In P.G. Clote and J.B. Remmel, editors, *Feasible Mathematics II*, pages 154–218. Birkhäuser, 1995.

15 A. Cobham. The intrinsic computational difficulty of functions. In *Logic, Methodology and Phylosophy of Science: Proc. 1964 International Congress*, pages 24–30. Amsterdam: North-Holland, 1965.

16 K.J. Compton and C. Laflamme. An algebra and a logic for NC^1. *Inf. Comput.*, 87(1/2):240–262, 1990.

17 S. Cook and T. Morioka. Quantified propositional calculus and a second-order theory for NC^1. *Arch. Math. Logic*, 44:711–749, 2005.

18 S. Cook and P. Nguyen. Theories for TC^0 and other small circuit classes. *Log. Meth. Comput Sci.*, 2:1–40, 2006.

19 M.L. Furst, J.B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. In *Proc. 22nd Annual Symposium on Foundations of Computer Science*, pages 260–270, 1981.

20 A. Grzegorczyk. Some classes of recursive functions. *Rozptawy Matematyczne*, 4, 1953.

21 Y. Gurevich and H. Lewis. A logic for constant-depth circuit. *Inf. Control*, 61:65–74, 1984.

22 L. Hella, J. Kontinen, and K. Luosto. Regular representations of uniform TC^0. arXiv:2309.06926.

23 N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16:760–778, 1987.

24 J. Johannsen. A bounded arithmetic theory for constant depth threshold circuits. In *GÖDEL '96*, volume 6 of *Springer Lecture Notes in Logic*, pages 224–234. Hájek, P., 1996.

25 D. Leivant. Ramified recurrence and computational complexity I: Word recurrence and polytime. In P.G. Clote and J.B. Remmel, editors, *Feasible Mathematics II*, Progress in Computer Science and Applied Logic, pages 320–343. Birkhüser, 1994.

26 D. Leivant and Y.-Y. Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19(1,2):167–184, 1993.

27 J.C. Lind. *Computing in Logarithmic Space.* PhD thesis, Massachusetts Institute of Technology, 1974.

28 S. Lindell. A purely logical characterization of circuit uniformity. In *7th Structure in Complexity Theory Conf.*, pages 185–192, 1992.

29 R.W. Ritchie. Classes of predicability computable functions. *Trans. Am. Math. Soc.*, 106:139–173, 1963.

30 H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer, 1999.

A Proofs from Section 3

A.1 The Schemas ℓ-ODE$_1$ and ℓ-ODE$_2$

Proof of Proposition 12. By Definition 10, for all x and $\mathbf{y}$:

$$\begin{aligned} f(x,\mathbf{y}) &= \sum_{u=-1}^{\ell(x)-1} \left(\prod_{t=u+1}^{\ell(x)-1} 2 \right) \times h\big(\alpha(u),\mathbf{y}\big) \\ &= \sum_{u=-1}^{\ell(x)-1} 2^{\ell(x)-u-1} \times h(\alpha(u),\mathbf{y}) \end{aligned}$$

with the convention that $\alpha(u) = 2^u - 1$, $\prod_x^{x-1} \kappa(x) = 1$ and $h\big(\alpha(-1),\mathbf{y}\big) = f(0,\mathbf{y})$. Notice that the given multiplication is always by a power of 2 decreasing for each increasing value of u, which basically corresponds to left-shifting (which can be computed in $\mathbf{FAC}^0$). Hence, since by Definition 10, $h(x,\mathbf{y}) \in \{0,1\}$, the outermost sum amounts to a concatenation (which again can be computed in $\mathbf{FAC}^0$).

Concretely, for any inputs x and $\mathbf{y}$, the desired polynomial-sized and constant-depth circuit to compute $f(x,\mathbf{y})$ is defined as follows:

- In parallel compute the values of $g(\mathbf{y})$ and of each $h(\alpha(u),\mathbf{y})$, for any $u \in \{0,\dots,\ell(x)-1\}$. For hypothesis, g and h are computable in $\mathbf{FAC}^0$, and, since there are $\ell(x)+1$ initial values to be computed, the whole desired computation can be done in polynomial size and constant depth.
- In one step, (left-)shift the value of $h\big(\alpha(-1),\mathbf{y})\big) = g(\mathbf{y})$ by padding $\ell(x)$ zeros on the right and, for $u \geq 0$, (left-)shift each value $h\big(\alpha(u),\mathbf{y}\big)$ by padding on the right $\ell(x)-u-1$ zeros (this corresponds to multiply by $2^{\ell(x)-u-1}$) and padding on the left $u + \ell(g(\mathbf{y}))$ zeros.
- Compute, bit-by-bit, the disjunction of all values computed above. Clearly, this is done in constant depth. ◀

Proof of Proposition 15. There are two main cases to be taken into account. If $k(\mathbf{y}) \neq 0$, the proof is similar to that of Proposition 12. Indeed, for all x and $\mathbf{y}$:

$$\begin{aligned} f(x, \mathbf{y}) &= \sum_{u=-1}^{\ell(x)-1} \left(\prod_{t=u+1}^{\ell(x)-1} 2^{\ell(k(\mathbf{y}))} \right) \times h\big(\alpha(u), \mathbf{y}\big) \\ &= \sum_{u=-1}^{\ell(x)-1} 2^{\ell(k(\mathbf{y})) \times (\ell(x)-u-1)} \times h\big(\alpha(u), \mathbf{y}\big) \end{aligned}$$

with the convention that $\alpha(u) = 2^u - 1$, $\prod_x^{x-1} \kappa(x) = 1$ and $h\big(\alpha(-1), \mathbf{y}\big) = f(0, \mathbf{y})$. Observe that here multiplication corresponds to a left-shifting where "basic shifting" corresponds to a left movement of $\ell(k(\mathbf{y}))$ digits. As for the basic case, it can be easily shown that this operation can be implemented by a constant-depth circuit. Then, analogously to Proposition 12, the outermost iterated sum amounts to concatenation (as, by construction, $h(\alpha(u), \mathbf{y}) \in \{0, 1\}$).

Concretely, we can construct a constant-depth circuit generalizing the procedure defined for the special case of ℓ-ODE$_1$:

- In parallel, compute the values of $g(\mathbf{y})$ and $h(\alpha(u), \mathbf{y})$, for each $u \in \{0, \dots, \ell(x) - 1\}$. This can be done in constant depth by hypothesis.
- In one step, shift the value of $g(\mathbf{y})$ by padding $\ell(k(\mathbf{y})) \times \ell(x)$ zeros on the right, and, for $u \geq 0$, shift all values $h(\alpha(u), \mathbf{y})$ by padding on the right $\ell(k(\mathbf{y})) \times (\ell(x) - u - 1)$ zeros (i.e. multiplying by $2^{\ell(k(\mathbf{y})) \times (\ell(x)-u-1)}$) and by padding on the left $\ell(g(\mathbf{y})) + \ell(k(\mathbf{y})) \times (u + 1) - 1$ zeros.
- Compute, bit-by-bit, the disjunction of all the values above.

In the special case of $k(\mathbf{y}) = 0$ and $h(x, \mathbf{y}) = 0$, it holds that for all x and $\mathbf{y}$, $f(x, \mathbf{y}) = g(\mathbf{y})$. This is clearly computable in $\mathbf{FAC}^0$, as corresponding to compute $g(\mathbf{y})$, which is computable in constant depth by hypothesis. ◀

A.2 The Schema ℓ-ODE$_3$

For $x > 0$, the equation characterizing Definition 16 can be re-written as:

$$f(x, \mathbf{y}) = f(x - 1, \mathbf{y}) - \Delta\ell(x - 1) \times \left\lceil \frac{f(x-1, \mathbf{y})}{2} \right\rceil,$$

where, as seen, $\Delta\ell(x - 1) = \ell(x) - \ell(x - 1)$. Observe that also in this case we are using $\times$ with a slight abuse of notation: indeed, we are dealing with "bit multiplication" and multiplying a number by 0 or 1 can be easily done in $\mathbf{FAC}^0$ (and easily rewritten in our setting using the basic conditional function if, which, in turn, can be defined in $\mathbb{ACDL}$, see Sec. 3.3). In other words,

$$\begin{aligned} f(x, \mathbf{y}) &= \begin{cases} f(x-1, \mathbf{y}) & \text{if } \ell(x) = \ell(x-1) \\ f(x-1, \mathbf{y}) - \left\lceil \frac{f(x-1, \mathbf{y})}{2} \right\rceil & \text{otherwise} \end{cases} \\ &= \begin{cases} f(x-1, \mathbf{y}) & \text{if } \ell(x) = \ell(x-1) \\ \left\lfloor \frac{f(x-1, \mathbf{y})}{2} \right\rfloor & \text{otherwise} \end{cases} \\ &= \begin{cases} f(x-1, \mathbf{y}) & \text{if } \ell(x) = \ell(x-1) \\ f(x-1, \mathbf{y}) \div 2 & \text{otherwise.} \end{cases} \end{aligned}$$

A bit more formally,

$$f(x,\mathbf{y}) = \left\lfloor \frac{f(\beta(\ell(x)-1),\mathbf{y})}{2} \right\rfloor = \left\lfloor \frac{f(2^{\ell(x)-1}-1,\mathbf{y})}{2} \right\rfloor = f\big(2^{\ell(x)-1}-1,\mathbf{y}\big) \div 2$$

where $\beta\big(\ell(z)\big) = 2^{\ell(z)}-1$ is the greatest integer the length of which is $\ell(z)$, i.e. here, $2^{\ell(x)-1}-1$ is the greatest integer the length of which is $\ell(x)-1$. Hence, starting with $x>0$, there are $\ell(x)-1$ jumps of values.

Proof of Proposition 17. By Definition 16 (as clarified by the remarks above),and, since $(a \div 2^b) \div 2 = a \div 2^{b+1}$, it is easily shown by induction that, for all x and $\mathbf{y}$:

$$\begin{aligned} f(x,\mathbf{y}) &= \left\lfloor \prod_{u=1}^{\ell(x)-1} \frac{g(\mathbf{y})}{2} \right\rfloor \\ &= \left\lfloor \frac{g(\mathbf{y})}{2^{\ell(x)-1}} \right\rfloor \\ &= g(\mathbf{y}) \div 2^{\ell(x)-1}. \end{aligned}$$

This corresponds to right-shifting $g(\mathbf{y})$ a number of times equal to $\ell(x)-1$, which can be easily implemented by a constant-depth circuit. ◀

A.3 Rewriting the Function BIT in $\mathbb{ACDL}$

First, the *log most significant part function* $\mathsf{msp}(x,y) : x,y \mapsto \left\lfloor \frac{y}{2^{\ell(x)}} \right\rfloor$ can be rewritten as the solution of the IVP:

$$\begin{aligned} f(0,y) &= y \\ \frac{\partial f(x,y)}{\partial \ell} &= -\left\lceil \frac{f(x,y)}{2} \right\rceil \end{aligned}$$

which is clearly an instance of ℓ-ODE$_3$.

Second, we introduce the basic conditional function:

$$\mathsf{if}(x,y,z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{otherwise.} \end{cases}$$

Notice that this function is also crucial to rewrite the CRN schema. As seen, the "shift function" $2^{\ell(x)} \times y$ can be easily rewritten via ℓ-ODE$_1$. Thus, $\mathsf{if}(x,y,z)$ is simulated in our setting by composition from shift, addition and subtraction:

$$\mathsf{if}(x,y,z) = \big(2^{\ell(1-\mathsf{sg}(x))} \times y - y\big) + \big(2^{\ell(\mathsf{sg}(x))} \times z - z\big).$$

Indeed, as desired, if $x=0$, then $\mathsf{sg}(x)=0$ and $\ell(\mathsf{sg}(x))=0$, so that $\mathsf{if}(0,y,z) = \big(2^{\ell(1)} \times y - y\big) + \big(2^{\ell(0)\times z - z}\big) = y$; similarity, for $x \neq 0$, $\mathsf{if}(0,y,z) = \big(2^{\ell(0)} \times y - y\big) + \big(2^{\ell(1)} \times z - z\big) = z$. Generalizing this definition we can capture a more general conditional function below:

$$\mathsf{cond}(x,v,y,z) = \begin{cases} y & \text{if } x < v \\ z & \text{otherwise} \end{cases}$$

Then, we consider the special bit function $\mathsf{bit}(x,y)$ returning 1 when the $\ell(y)^{th}$ bit of x is 1. This can be rewritten in $\mathbb{ACDL}$ due to msp:

$$\mathsf{bit}(x,y) = \mathsf{msp}(y,x) - 2 \times \mathsf{msp}(2y+1,x).$$

Finally, we introduce the function $\mathsf{bexp}(x, y)$, that, for any $y \leq \ell(x)$, returns 2^i. We start by defining $f_{aux}(t, x, i)$ by the ℓ-ODE$_1$ schema below:

$$f_{aux}(0, x, i) = \mathsf{if}(i, 1, 0)$$
$$\frac{\partial f_{aux}(t, x, i)}{\partial \ell(t)} = f_{aux}(t, x, i) + h_{aux}(t, i)$$

where

$$h_{aux}(t, i) = \mathsf{if}(\ell(t) - i, 1, 0).$$

Observe that, as seen, if can be rewritten in $\mathbb{ACDL}$ (while ℓ and subtraction are basic functions). Then, for $i \leq \ell(x)$, we obtain $f_{aux}(x, x, i) = 2^{\ell(x)-i}$. The function bexp is then defined as follows:

$$\mathsf{bexp}(x, i) = \mathsf{msp}\big(f_{aux}(x, x, i) - 1, 2^{\ell(x)}\big) = \left\lfloor \frac{2^{\ell(x)}}{2^{\ell(x)-i}} \right\rfloor = 2^i,$$

as the length of $f_{aux}(x, x, i) - 1$ is $\ell(x) - i$. Clearly, the function bexp is also in $\mathbb{ACDL}$ as all the functions involved in its definitions (namely, msp, f_{aux} and $2^{\ell(\cdot)}$) are in $\mathbb{ACDL}$.

We conclude by showing that, due to bexp and bit, the desired BIT function can be rewritten in our setting by composition:

$$\mathsf{BIT}(x, y) = \mathsf{bit}(x, \mathsf{bexp}(x, y) - 1).$$

Observe that alternative proofs are possible, but the one proposed here, and based on the introduction of bexp, not only has the advantage of being straightforward, but also avoids the unnatural use of function most significant part function, MSP.

A.4 On the ODE-Characterization of FTC^0

Proof of Proposition 20. The proof is similar to that of Proposition 15. The main difference concerns $g(\mathbf{y})$ and $h(x, \mathbf{y})$, which are now expressions possibly including ×.

As seen, for all x and $\mathbf{y}$:

$$f(x, \mathbf{y}) = \sum_{u=-1}^{\ell(x)-1} \left(\prod_{t=u+1}^{\ell(x)-1} 2^{\ell(k(\mathbf{y}))} \right) \times h\big(\alpha(u), \mathbf{y}\big)$$

with the convention that $\alpha(u) = 2^u - 1$, $\prod_x^{x-1} \kappa(x) = 1$ and $h\big(\alpha(-1), \mathbf{y}\big) = f(0, \mathbf{y})$. Intuitively, the (constant-depth) circuit we are going to construct is equivalent to that of Proposition 15, but the values to be initially computed in parallel are obtained even via ×. Yet, the introduction of multiplication does not affect the overall structure of the circuit, as $h\big(\alpha(u), \mathbf{y}\big) \in \{0, 1\}$, so that the final sum again corresponds to a simple bit-concatenation (without carries).

More precisely, the desired constant-depth circuit is defined as follows:

- In parallel, compute the values of $g(\mathbf{y})$ and, for any $u = 0, \ldots, \ell(x) - 1$, of $h\big(\alpha(u), \mathbf{y}\big)$. Observe that this can be done in $\mathbf{FTC}^0$, but possibly not in $\mathbf{FAC}^0$, as now these arithmetic expressions may include ×.
- The value of $g(\mathbf{y})$ is shifted by padding $2^{\ell(k(\mathbf{y})) \times \ell(x)}$ zeros on the right and, for $u \geq 0$, all values $h\big(\alpha(u), \mathbf{y}\big)$ are shifted by padding on the right $\ell(k(\mathbf{y}))$ zeros $\ell(x) - u - 1$ times, and by padding on the left $\ell(k(\mathbf{y}))$ zeros $\ell(g(\mathbf{y})) + u$ times.
- the disjunction of the above values is computed bit by bit. ◀

Proof of Proposition 22. $\mathbb{TCDL} \subseteq \mathbf{FTC}^0$. All basic functions are computable in $\mathbf{FTC}^0$, and the class is closed under composition and, by Propositions 20 and 21, under ℓ-ODE$_2$ and ℓ-ODE$_3$.

$\mathbf{FTC}^0 \subseteq \mathbb{TCDL}$. By mimicking the arithmetization proof provided in [14] (see Sec. 2.2), as all the functions defining $\mathcal{TC}_0$ can be rewritten in $\mathbb{TCDL}$ (this time including ×, which is basic in $\mathbb{TCDL}$). ◀

Switching Classes: Characterization and Computation

Dhanyamol Antony
Department of Computer Science and Automation, Indian Institute of Science, Bengaluru, India

Yixin Cao
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Sagartanu Pal
Department of Computer Science & Engineering, Indian Institute of Technology Dharwad, India

R. B. Sandeep
Department of Computer Science & Engineering, Indian Institute of Technology Dharwad, India

Abstract

In a graph, the switching operation reverses adjacencies between a subset of vertices and the others. For a hereditary graph class $\mathcal{G}$, we are concerned with the maximum subclass and the minimum superclass of $\mathcal{G}$ that are closed under switching. We characterize the maximum subclass for many important classes $\mathcal{G}$, and prove that it is finite when $\mathcal{G}$ is minor-closed and omits at least one graph. For several graph classes, we develop polynomial-time algorithms to recognize the minimum superclass. We also show that the recognition of the superclass is NP-hard for H-free graphs when H is a sufficiently long path or cycle, and it cannot be solved in subexponential time assuming the Exponential Time Hypothesis.

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

Keywords and phrases Switching, Graph modification, Minor-closed graph class, Hereditary graph class

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.11

Related Version *Full Version*: https://arxiv.org/abs/2403.04263 [1]

Funding Supported by SERB grants CRG/2022/006770 and MTR/2022/000692, RGC grant 15221420, and NSFC grants 61972330 and 62372394.

1 Introduction

In a graph G, the operation of *switching* a subset A of vertices is to reverse the adjacencies between A and $V(G) \setminus A$. Two vertices $x \in A$ and $y \in V(G) \setminus A$ are adjacent in the resulting graph if and only if they are not adjacent in G. The switching operation, introduced by van Lint and Seidel [35] (see more at [29, 30, 31]), is related to many other graph operations, most notably variations of graph complementation. The *complement* of a graph G is a graph defined on the same vertex set of G, where a pair of distinct vertices are adjacent if and only if they are not adjacent in G. The *subgraph complementation* on a vertex set A is to replace the subgraph induced by A with its complement, while keeping the other part, including connections between A and the outside, unchanged [2]. Switching A is equivalent to taking the complement of the graph itself and the subgraphs induced by A and $V(G) \setminus A$. Indeed, the widely used *bipartite complementation* operation of a bipartite graph is nothing but switching one part of the bipartition. A special switching operation where A consists of a single vertex is also well studied. It is a nice exercise to show that switching A is equivalent to switching the vertices in A one by one. This is somewhat related to the local complementation operation [28].

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 11; pp. 11:1–11:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Two graphs are *switching equivalent* if one can be obtained from the other by switching. Colbourn and Corneil [9] proved that deciding whether two graphs are switching equivalent is polynomial-time equivalent to the graph isomorphism problem. Another interesting topic is to focus on graphs from a hereditary graph class $\mathcal{G}$ – a class is *hereditary* if it is closed under taking induced subgraphs. There are two natural questions in this direction. Given a graph G,

- whether G can be switched to a graph in $\mathcal{G}$? and
- whether all switching equivalent graphs of G are in $\mathcal{G}$?

We use the *upper $\mathcal{G}$ switching class* and the *lower $\mathcal{G}$ switching class*, respectively, to denote the set of positive instances of these two problems. Since switching the empty set does not change the graph, the answer of the first question is yes for every graph in $\mathcal{G}$, while the answer of the second question can only be yes for a graph in $\mathcal{G}$. Thus, the class $\mathcal{G}$ is sandwiched in between these two switching classes. Note that the three classes collapse into one when $\mathcal{G}$ is closed under switching, e.g., complete bipartite graphs.

Both switching classes are also hereditary. For the upper switching class, if a graph G can be switched to a graph H in $\mathcal{G}$, then any induced subgraph of G can be switched to an induced subgraph of H, which is in $\mathcal{G}$ because $\mathcal{G}$ is hereditary. For the lower switching class, recall that a hereditary graph class $\mathcal{G}$ can be characterized by a (not necessarily finite) set $\mathcal{F}$ of forbidden induced subgraphs. A graph is in $\mathcal{G}$ if and only if it does not contain any forbidden induced subgraph. If G contains any induced subgraph that is switching equivalent to a graph in $\mathcal{F}$, then G cannot be in the lower $\mathcal{G}$ switching class. Thus, the forbidden induced subgraphs of the lower $\mathcal{G}$ switching class are precisely all the graphs that are switching equivalent to some graphs in $\mathcal{F}$.

Even when $\mathcal{G}$ has an infinite set of forbidden induced subgraphs, the lower $\mathcal{G}$ switching class may have very simple structures. The list of forbidden induced subgraphs obtained as above is usually not minimal. For example, Hertz [18] showed that the lower perfect switching class has only four forbidden induced subgraphs, all switching equivalent to the five-cycle. In the same spirit as Hertz [18], we characterize the lower $\mathcal{G}$ switching classes of a number of important graph classes.

▶ **Theorem 1.** *The lower $\mathcal{G}$ switching class is characterized by a finite number of forbidden induced subgraphs when $\mathcal{G}$ is one of the following graph classes: weakly chordal, comparability, co-comparability, permutation, distance-hereditary, Meyniel, bipartite, chordal bipartite, complete multipartite, complete bipartite, chordal, strongly chordal, interval, proper interval, Ptolemaic, and block.*

Indeed, since the forbidden induced subgraphs of threshold graphs are $2K_2, C_4$, and P_4 [8], by the arguments given above, the forbidden subgraphs of the lower threshold switching class are all graphs on four vertices (every graph on four vertices is switching equivalent to a graph in $\{2K_2, C_4, P_4\}$). This class, consisting of only graphs of order at most three, is finite. Also finite are lower switching classes of minor-closed graph classes that are nontrivial[1] (there exists at least one graph not in this class).

▶ **Theorem 2.** *Let $\mathcal{G}$ be a nontrivial minor-closed graph class, and let p be the smallest order of a forbidden minor of $\mathcal{G}$. Then $|V(G)| = O(p\sqrt{p})$, for graphs G in lower $\mathcal{G}$ switching class.*

[1] We thank an anonymous reviewer for the bound in Theorem 2, which improves the bound in a previous version of this manuscript.

Theorems 1 and 2 immediately imply polynomial-time and constant-time algorithms, respectively, for recognizing these lower switching classes, i.e., deciding whether a graph is in the class. We remark that there are classes $\mathcal{G}$ such that the lower $\mathcal{G}$ switching class has an infinite number of forbidden induced subgraphs.

The upper $\mathcal{G}$ switching classes turn out to be more complicated. These classes are nontrivial even for the class of H-free graphs for a fixed graph H. Although $\mathcal{G}$ has only one forbidden induced subgraph, the number of forbidden induced subgraphs of the upper $\mathcal{G}$ switching class is usually infinite. Based on our current knowledge, exceptions do exist but are rare [19]. Even so, for many graph classes $\mathcal{G}$, polynomial-time algorithms for recognizing the upper $\mathcal{G}$ switching class exist, e.g., bipartite graphs [16]. Our understanding of this problem is very limited, even for classes defined by forbidding a single graph H. For all graphs H on at most three vertices, polynomial-time algorithms are known for recognizing the upper H-free switching class [16, 17, 24]. Of a graph H on four vertices, the four-path [18] and the claw [19] have been settled. We present a polynomial-time algorithm for paw-free graphs. If two graphs H_1 and H_2 are complements to each other, then the recognition of the upper H_1-free switching class is polynomially equivalent to that of the upper H_2-free switching class. Thus, the remaining cases on four vertices are the diamond, the cycle, and the complete graph. We made attempt to them by solving the class of forbidding the four-cycle and its complement, which is known as pseudo-split graphs.

▶ **Theorem 3.** *The upper $\mathcal{G}$ switching class can be recognized in polynomial time when $\mathcal{G}$ is one of the following graph classes: paw-free graphs, pseudo-split graphs, split graphs, $\{K_{1,p}, \overline{K_{1,q}}\}$-free graphs, and bipartite chain graphs.*

In Theorem 3, we want to highlight the algorithms for pseudo-split graphs and for split graphs. We actually show a stronger result. Any input graph G has only a polynomial number of ways to be switched to a graph in these two classes, and we can enumerate them in polynomial time. Thus, the algorithms can apply to hereditary subclasses of pseudo-split graphs, provided that these subclasses themselves can be recognized in polynomial time. This is only possible when the lower switching classes of them are finite. It is unknown whether the other direction also holds true.

Jelínková and Kratochvíl [19] found graphs H such that the upper H-free switching class is hard to recognize. The smallest graph they found is on nine vertices. More specifically, they showed that, for all $k \geq 3$, there is a graph of order $3k$ with this property. The graph is obtained from a three-vertex path by substituting one degree-one vertex with an independent set of k vertices, and each of the other two vertices with a clique of k vertices. We show that the recognition of the upper H-free switching class is already hard when H is a cycle on seven vertices or a path on ten vertices. Our proofs can be adapted to longer ones.

▶ **Theorem 4.** *Deciding whether a graph is switching equivalent to a P_{10}-free graph or a C_7-free graph is NP-complete, and it cannot be solved in subexponential time (on $|V(G)|$) assuming the Exponential Time Hypothesis.*

Since the problem admits a trivial $2^{|V(G)|} \cdot |V(G)|^{O(1)}$-time algorithm, by enumerating all subsets of $V(G)$, our bound in Theorem 4 is asymptotically tight. We conjecture that it is NP-complete to decide whether a graph can be switched to an H-free graph when H is a cycle or path of length six.

Theorem 1 and 2 are proved in Section 3, Theorem 3 is proved in Section 4, and Theorem 4 is proved in Section 5. Due to space constraints, most of the proofs are left to a full version of the paper.

Other related work

Jelínková et al. [20] studied the parameterized complexity of the recognition problem of the upper switching classes. Let us remark that there is also study on the upper switching classes for non-hereditary graph classes. For example, we can decide in polynomial time whether a graph can be switching equivalent to a Hamiltonian graph [11] or to an Eulerian graph [16], but it is NP-complete to decide whether a graph can be switching equivalent to a regular graph [23]. Cameron [6] and Cheng and Wells Jr. [7] generalized the switching operation to directed graphs. Foucaud et al. [13] studied switching operations in a different setting.

Seidel [30] showed that the size of a maximum set of switching inequivalent graphs on n vertices is equivalent to the number of two-graphs of size n. This is further shown to be the same as the number Eulerian graphs on n vertices [25] and graphs on $2n$ vertices admitting certain coloring [26]. Bodlaender and Hage [4] showed that the switching operation does not change the cliquewidth of a graph too much, though it may change the treewidth significantly. The switching equivalence between graphs in certain classes can be decided in polynomial time. For example, acyclic graphs because two forests are switching equivalent if and only if they are isomorphic [14]. In a complementary study, Hage and Harju [15] characterized graphs that cannot be switched to any forest. They are either a small graph on at most nine vertices, or switching equivalent to a cycle.

From a graph G on n vertices, we can obtain n graphs by switching each vertex, called the *switching deck* of G. The *switching reconstruction conjecture* of Stanley [32] asserts that for any $n > 4$, if two graphs on n vertices have the same switching deck, they must be isomorphic. The conjecture remains widely open, and we know that it holds on triangle-free graphs [12]. A similar question in digraph is also studied [5].

2 Preliminaries

All the graphs discussed in this paper are finite and simple. The vertex set and edge set of a graph G are denoted by, respectively, $V(G)$ and $E(G)$. Let $n = |V(G)|$ and $m = |E(G)|$. For a subset $U \subseteq V(G)$, we denote by $G[U]$ the subgraph of G induced by U, and by $G - U$ the subgraph $G[V(G) \setminus U]$, which is shortened to $G - v$ when $U = \{v\}$. The *neighborhood* of a vertex v, denoted by $N_G(v)$, comprises vertices adjacent to v, i.e., $N_G(v) = \{u \mid uv \in E(G)\}$, and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. The *closed neighborhood* and the *neighborhood* of a set $X \subseteq V(G)$ of vertices are defined as $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$, respectively. We may drop the subscript if the graph is clear from the context. We write $N(u, v)$ and $N[u, v]$ instead of $N(\{u, v\})$ and $N[\{u, v\}]$; i.e., we drop the braces when writing the neighborhood of a vertex set. Two vertex sets X and Y are *complete* (resp., *nonadjacent*) *to* each other if all (resp., no) edges between X and Y are present.

For positive ℓ, we use C_ℓ ($\ell \geq 3$), P_ℓ, and K_ℓ to denote the cycle, path, and complete graph, respectively, on ℓ vertices. When $\ell \geq 4$, an induced C_ℓ is called an *ℓ-hole.* A complete bipartite graph with p and q vertices in the two parts are denoted as $K_{p,q}$.

The *disjoint union* of two graphs G_1 and G_2 is denoted by $G_1 + G_2$. The *complement graph* $\overline{G}$ of a graph G is defined on the same vertex set $V(G)$, where a pair of distinct vertices u and v is adjacent in $\overline{G}$ if and only if $uv \notin E(G)$. By $\mathcal{G}^c$, we denote the set of graphs not in $\mathcal{G}$. The switching of a vertex subset A of a graph G is denoted by $S(G, A)$. It has the same vertex set as G and its edge set is $E(G[A]) \cup E(G - A) \cup \{uv \mid u \in A, v \in V(G) \setminus A, uv \notin E(G)\}$. The following observations are immediate from the definition. The *symmetric difference* of two sets is defined as $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

▶ **Proposition 5** (folklore). *Let G be a graph, and $A, B \subseteq V(G)$.*

- $S(G, A) = S(G, (V(G) \setminus A))$.
- $S(S(G, A), A) = G$.
- $S(S(G, A), B) = S(S(G, B), A) = S(G, A \Delta B)$.
- $\overline{S(G, A)} = S(\overline{G}, A)$.

Two graphs G and G' are called *switching equivalent* if $S(G, A) = G'$ for some $A \subseteq V(G)$. By Proposition 5, switching is an equivalence relation. For example, the eleven graphs of order 4 can be partitioned into the following three sets

$$\{C_4, \overline{K_3 + K_1}, 4K_1\}, \{2K_2, K_3 + K_1, K_4\}, \{P_4, K_2 + 2K_1, \overline{K_2 + 2K_1}, P_3 + K_1, \overline{P_3 + K_1}\}.$$

Note that $\overline{K_3 + K_1}$ is the claw, $\overline{P_3 + K_1}$ is the paw, and $\overline{K_2 + 2K_1}$ is the diamond; see Figure 1 and 2a. For a graph G, we use $\mathcal{S}(G)$ to denote the set of non-isomorphic graphs that can be obtained from G by switching. Figure 2 illustrates $\mathcal{S}(C_4)$ and $\mathcal{S}(C_5)$. For a set $\mathcal{G}$ of graphs, by $\mathcal{S}(\mathcal{G})$ we denote the union of $\mathcal{S}(G)$ for $G \in \mathcal{G}$.

A graph G is a *split graph* if the vertex set of G can be partitioned in such a way that one is a clique and the other is an independent set. *Split partitions* of a split graph refer to such (clique, independent set) partitions. An *edgeless graph* is a graph without any edges.

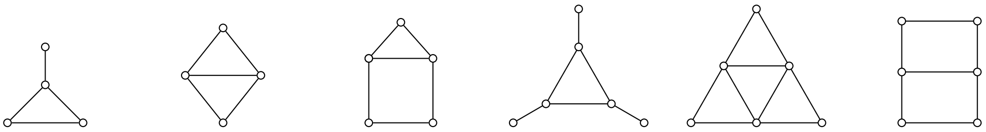
In general, for two sets $\mathcal{G}$ and $\mathcal{H}$ of graphs, we say that $\mathcal{G}$ is $\mathcal{H}$-free if G is H-free for every $G \in \mathcal{G}$ and for every $H \in \mathcal{H}$. By $\mathcal{F}(\mathcal{H})$, we denote the class of $\mathcal{H}$-free graphs. Note that $\mathcal{F}(\mathcal{H} \cup \mathcal{H}') = \mathcal{F}(\mathcal{H}) \cap \mathcal{F}(\mathcal{H}')$.

For a graph property $\mathcal{G}$, the *lower $\mathcal{G}$ switching class*, denoted by $\mathcal{L}(\mathcal{G})$, consists of all graphs G with $\mathcal{S}(G) \subseteq \mathcal{G}$. Note that every graph in $\mathcal{L}(\mathcal{G})$ is also in $\mathcal{G}$. Thus, $\mathcal{L}(\mathcal{G})$ is the maximal subset $\mathcal{G}'$ of $\mathcal{G}$ such that $\mathcal{S}(\mathcal{G}') = \mathcal{G}'$. The *upper $\mathcal{G}$ switching class*, denoted by $\mathcal{U}(\mathcal{G})$, consists of all graphs G with $\mathcal{S}(G) \cap \mathcal{G} \neq \emptyset$. Clearly, every graph in $\mathcal{G}$ is in $\mathcal{U}(\mathcal{G})$. Therefore, $\mathcal{U}(\mathcal{G})$ is the minimal superset $\mathcal{G}'$ of $\mathcal{G}$ such that $\mathcal{S}(\mathcal{G}') = \mathcal{G}'$. We note that $\mathcal{U}(\mathcal{G}) = \mathcal{S}(\mathcal{G})$. The following proposition is immediate from the definitions and Proposition 5.

▶ **Proposition 6.** *Let $\mathcal{G}$ and $\mathcal{G}'$ be graph classes. Then the following hold true.*

1. $(\mathcal{L}(\mathcal{G}))^c = \mathcal{U}(\mathcal{G}^c)$.
2. *If $\mathcal{G}' \subseteq \mathcal{G}$, then $\mathcal{L}(\mathcal{G}') \subseteq \mathcal{L}(\mathcal{G})$ and $\mathcal{U}(\mathcal{G}') \subseteq \mathcal{U}(\mathcal{G})$.*
3. $\mathcal{L}(\mathcal{G}) \cap \mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G} \cap \mathcal{G}')$.

▶ **Proposition 7.** *For a set $\mathcal{H}$ of graphs, $\mathcal{L}(\mathcal{F}(\mathcal{H})) = \mathcal{F}(\mathcal{U}(\mathcal{H}))$.*

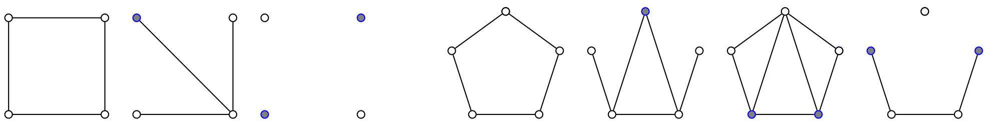


(a) paw **(b)** diamond **(c)** house **(d)** net **(e)** sun **(f)** domino

Figure 1 Small graphs.

3 Lower switching classes

Every (odd) hole of length at least seven contains an induced $P_4 + K_1$, and its complement contains an induced gem. Both $P_4 + K_1$ and the gem are in $\mathcal{S}(C_5)$; see Figure 2b. Thus, all the forbidden induced subgraphs of perfect graphs, namely, odd holes and their complements, boil down to $\mathcal{S}(C_5)$, and the lower perfect switching class is equivalent to the lower C_5-free



(a) $\mathcal{S}(C_4) = \{C_4, \text{claw}, 4K_1\}$ **(b)** $\mathcal{S}(C_5) = \{C_5, \text{bull}, \text{gem}, P_4 + K_1\}$

Figure 2 Switching equivalent graphs of C_4 and C_5. Switching the solid nodes (or the rest) results in the first graph in the list.

switching class [18]. In the same spirit, we characterized the lower $\mathcal{G}$ switching classes of a number of important graph classes listed in Figure 3. The results are listed in Table 1. Since all these lower switching classes have finite characterizations, they can be recognized in polynomial time. For the class of chordal graphs and several of its subclasses, we show a stronger structural characterization of their lower switching classes. They have to be proper interval graphs with a very special structure. The following lemma, a consequence of Proposition 6(2), is crucial for our arguments.

▶ **Lemma 8.** *Let $\mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$ be three classes of graphs such that $\mathcal{G}_1 \subseteq \mathcal{G}_2 \subseteq \mathcal{G}_3$. If $\mathcal{L}(\mathcal{G}_3) = \mathcal{L}(\mathcal{G}_1)$, then $\mathcal{L}(\mathcal{G}_2) = \mathcal{L}(\mathcal{G}_1)$. In particular, the following is true. Let $\mathcal{H}_1, \mathcal{H}_2$, and $\mathcal{H}_3$ be three sets of graphs such that $\mathcal{H}_3 \subseteq \mathcal{H}_2 \subseteq \mathcal{H}_1$. If $\mathcal{L}(\mathcal{F}(\mathcal{H}_3)) = \mathcal{L}(\mathcal{F}(\mathcal{H}_1))$, then $\mathcal{L}(\mathcal{F}(\mathcal{H}_2)) = \mathcal{L}(\mathcal{F}(\mathcal{H}_1))$.*

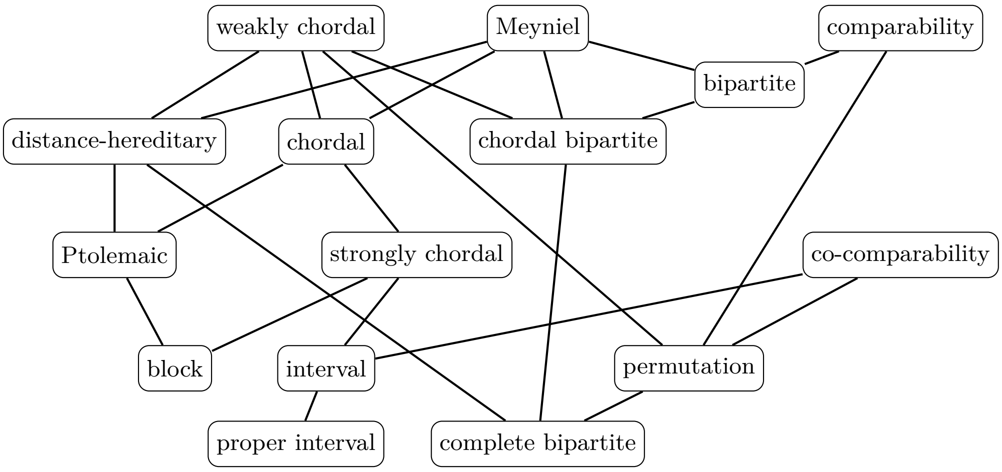


Figure 3 The Hasse diagram of graph classes studied in Section 3.

To see a simple application of Lemma 8, let $\mathcal{G}$ be the class of complete bipartite graphs and $\mathcal{G}'$ be the class of bipartite graphs. Since K_3 and $K_2 + K_1$ are switching equivalents, and bipartite graphs are K_3-free, we obtain that lower bipartite switching class is $\{K_3, K_2 + K_1\}$-free. Recall that $\{K_3, K_2 + K_1\}$-free graphs are exactly the class of complete bipartite graphs. Further, switching a complete bipartite graph results in a complete bipartite graph. Therefore, lower $\mathcal{G}''$ switching class is equivalent to the class of complete bipartite graphs, where $\mathcal{G}''$ is a subclass of bipartite graphs and a superclass of complete bipartite graphs, such as bipartite graphs, complete bipartite graphs, and chordal bipartite graphs (bipartite graphs in which every cycle longer than 4 has a chord).

▶ **Lemma 9.** *Let $\mathcal{G}$ be any subclass of bipartite graphs and any superclass of complete bipartite graphs. Then $\mathcal{L}(\mathcal{G})$ is the class of complete bipartite graphs.*

Table 1 Lower switching classes of various graph classes.

$\mathcal{G}$	$\mathcal{L}(\mathcal{G})$	By
weakly chordal, permutation	$\{C_5, C_6, \overline{C_6}\}$-free	Corollary 11
distance-hereditary	$\{\text{domino}, \text{house}, C_5, C_6\}$-free	
comparability	$\{C_5, \overline{C_6}\}$-free	
co-comparability	$\{C_5, C_6\}$-free	
Meyniel graphs	$\{C_5, \text{house}\}$-free	
complete bipartite, chordal bipartite, bipartite	complete bipartite	Lemma 9
chordal, strongly chordal, interval, proper interval, Ptolemaic	$\mathcal{C}_0$	Corollary 13
block	$(+)$, $(+, 0, +)$, $(1, 1, 1)$, and $(1, 0, 1, 0, 1)$	Lemma 14

Let $\mathcal{H}$ be the set of all graphs having an induced subgraph isomorphic to at least one graph in $\mathcal{S}(C_5)$. A *building* is obtained from a hole by adding an edge connecting two vertices of distance two; e.g., the house, see Figure 1. An *odd building* is a building with odd number of vertices.

▶ **Observation 10.** *$\mathcal{H}$ contains C_5, holes of length at least seven, complements of holes of length at least seven, and buildings of at least six vertices.*

Lemma 8 and Observation 10 lead us to Corollary 11.

▶ **Corollary 11.** *The forbidden induced subgraphs of the lower $\mathcal{G}$ switching class of $\mathcal{G}$ being weakly chordal, distance-hereditary, comparability, co-comparability, permutation, and Meyniel graphs are* $\{C_5, C_6, \overline{C_6}\}$, $\{\text{domino}, \text{house}, C_5, C_6\}$, $\{C_5, \overline{C_6}\}$, $\{C_5, C_6\}$, $\{C_5, C_6, \overline{C_6}\}$, $\{C_5, \text{house}\}$, *respectively.*

Next we deal with the class of chordal graphs and its subclasses. We start with showing that the lower $\{C_4, C_5, C_6\}$-free switching class is a subclass of proper interval graphs and has very simple structures. Let $a_1, \ldots, a_p$ be p nonnegative integers. For $1 \leq i \leq p$, we substitute the ith vertex of a path on p vertices with a clique of a_i vertices. We denote the resulting graph as $(a_1, a_2, \ldots, a_p)$. For example, the paw and the diamond are $(1, 1, 2)$ and $(1, 2, 1)$, respectively, while the complement of the diamond can be represented as $(2, 0, 1, 0, 1)$. We use "+" to denote an unspecified positive integer, and hence $(+)$ stands for all complete graphs.

The forbidden induced subgraphs of proper interval graphs are holes, sun, net, and claw. Note that a sun and a net (see Figure 1) contains an induced bull ($\in \mathcal{S}(C_5)$), while any cycle on at least seven vertices contains an induced $P_4 + K_1 \in \mathcal{S}(C_5)$. A claw is in $\mathcal{S}(C_4)$. Therefore, lower $\{C_4, C_5, C_6\}$-free switching class is a subclass of proper interval graphs. A careful analysis shows that the structure is much simpler.

▶ **Lemma 12.** *The lower $\{C_4, C_5, C_6\}$-free switching class consists of graphs* $(+)$, $(+, +, 1)$, $(+, 1, +)$, $(+, 0, +)$, $(+, +, 1, 0, +)$, $(+, 0, +, 0, 1)$, $(+, +, 1, +)$, *and* $(+, +, 1, +, +)$.

Let $\mathcal{C}_0$ denote the lower $\{C_4, C_5, C_6\}$-free switching class. Since chordal graphs are $\{C_4, C_5, C_6\}$-free, lower chordal switching class is a subclass of $\mathcal{C}_0$. By Lemma 12, $\mathcal{C}_0$ is a subclass of lower chordal switching class. Therefore, they are equivalent. This same observation applies to subclasses of chordal graphs that contain all the graphs in $\mathcal{C}_0$ and by Lemma 8 to superclasses of chordal graphs which are $\{C_4, C_5, C_6\}$-free.

▶ **Corollary 13.** *The following switching classes are all equivalent to $\mathcal{C}_0$: lower chordal switching class, lower strongly chordal switching class, lower interval switching class, lower proper interval switching class, and lower Ptolemaic switching class.*

Proof. Since chordal graphs, strongly chordal graphs, interval graphs, and proper interval graphs are all hole-free, all the lower switching classes are subclasses of $\mathcal{C}_0$ by Proposition 6. On the other hand, by Lemma 12, all the graphs in $\mathcal{C}_0$ are proper interval graphs. Thus, $\mathcal{C}_0$ is a subclass of proper interval switching graphs, hence also a subclass of the first three switching classes. Ptolemaic graphs are gem-free chordal graphs. Since gem is in $\mathcal{S}(C_5)$, the lower Ptolemaic switching class is also $\mathcal{C}_0$. Thus, they are all equal. ◀

The class of line graphs has nine forbidden induced subgraphs [3], two of which are switching equivalent to C_6, and one C_4. Although C_5 is not forbidden, we show that a graph in the lower line switching class contains an induced C_5 if and only if it is a C_5. Thus, this switching class consists of $\mathcal{S}(C_5)$ and a subclass of $\mathcal{C}_0$.

▶ **Lemma 14.** *The lower block switching class consists of graphs* $(+)$, $(+,0,+)$, $(1,1,1)$, *and* $(1,0,1,0,1)$. *The lower line switching class comprises of (+), (1,1,1), (2,1,1), (1,2,1), (2,1,2), (+,0,+), (1,1,1,0,1), (2,1,1,0,1), (1,0,1,0,1), (2,0,1,0,1), (2,0,2,0,1), (1,1,1,1), (1,2,1,1), (1,1,1,1,1), and $\mathcal{S}(C_5)$.*

A graph F is a *minor* of a graph G if F can be obtained from a subgraph of G by contracting edges (identifying the two ends of the edge and keeping one edge between the resulting vertex and each of the neighbors of the end points of the edge). For example, any cycle contains all shorter cycles as minors. A graph class $\mathcal{G}$ is *minor-closed* if every minor of a graph in $\mathcal{G}$ also belongs to $\mathcal{G}$. In other words, there is a set $\mathcal{M}$ of *forbidden minors* such that a graph belongs to $\mathcal{G}$ if and only if it does not contain as a minor any graph in $\mathcal{M}$. Since an induced subgraph of a graph G is a minor of G, a minor-closed graph class is hereditary. We say that a graph class is *nontrivial* if there is at least one graph not in the class.

Kostochka [21, 22] and Thomason [33] proved that, there exists an absolute constant $c > 0$ such that every graph G with at least $c \cdot |V(G)| \cdot p\sqrt{p}$ edges has K_p as a minor. See [34] for an overview. This helps us to prove Theorem 2:

▶ **Theorem 2.** *Let $\mathcal{G}$ be a nontrivial minor-closed graph class, and let p be the smallest order of a forbidden minor of $\mathcal{G}$. Then $|V(G)| = O(p\sqrt{p})$, for graphs G in lower $\mathcal{G}$ switching class.*

Proof. Let $G \in \mathcal{L}(\mathcal{G})$ be a graph with n vertices. It is straight-forward to verify that there exists a constant $c' > 0$ such that either G or $S(G, A)$ has $c' \cdot n^2$ edges, where A is any subset of $V(G)$ with cardinality $\lfloor n/2 \rfloor$. If $c' \cdot n^2 \geq c \cdot n \cdot p\sqrt{p}$, then G has a K_p-minor. Therefore, $n = O(p\sqrt{p})$. ◀

We have found that, for the class of outerplanar graphs, planar graphs, and series-parallel graphs, the maximum orders of graphs in the lower switching classes are five, seven, and at most 12, respectively.

Let us mention that there are classes $\mathcal{G}$ such that the lower $\mathcal{G}$ switching class has an infinite number of forbidden induced subgraphs.

▶ **Lemma 15.** *For any infinite set $I \subseteq \{9, 10, \ldots\}$, the forbidden induced subgraphs of the lower $\{C_\ell, \ell \in I\}$-free switching class are $\bigcup_{\ell \in I} \mathcal{S}(C_\ell)$.*

4 Upper switching classes: algorithms

For the recognition of the upper $\mathcal{G}$ switching class, the input is a graph G, and the solution is a vertex subset $A \subseteq V(G)$ such that $S(G,A) \in \mathcal{G}$.

We start with split graphs. If the input graph G is a split graph, then we have nothing to do. Suppose that G is in the upper split switching class. Let A be a solution, and $K \uplus I$ a split partition of $S(G,A)$. Note that if $A \in \{K, I\}$, then G is a split graph. We may assume that A intersects both K and I: if A is a proper subset of K or I, we replace A with $V(G) \setminus A$. We can guess a pair of vertices $u \in A \cap K$ and $v \in A \cap I$. The vertex set $V(G) \setminus \{u, v\}$ can be partitioned into four parts, namely, $N(u) \setminus N[v]$, $N(v) \setminus N[u]$, $N(u) \cap N(v)$, and $V(G) \setminus N[u,v]$. It is easy to see that the first is a subset of A while the second is disjoint from A. The subgraphs $G[N(u) \cap N(v)]$ and $G - N[u,v]$ must be split graphs, and each admits a special split partition with respect to A. The algorithm is described in Figure 4. We can modify the algorithm so that it enumerates all solutions.

▶ **Theorem 16.** *Let G be a graph. There are a polynomial number of subsets A of $V(G)$ such that $S(G,A)$ is a split graph, and they can be enumerated in polynomial time.*

```
1.        if $G$ is a split graph then return “yes”;
2.        for each pair of vertices $u, v \in V(G)$ do
2.1.          if $G[N(u) \cap N(v)]$ is not a split graph then continue;
2.2.          if $G - N[u,v]$ is not a split graph then continue;
2.3.          for each split partition $K_1 \uplus I_1$ of $G[N(u) \cap N(v)]$ do
2.3.1.            for each split partition $K_2 \uplus I_2$ of $G - N[u,v]$ do
2.3.1.1.              if $S(G, \{u,v\} \cup (N(u) \setminus N[v]) \cup K_1 \cup I_2)$ is a split graph
                      then return “yes”;
3.        return “no.”
```

Figure 4 The algorithm for split graphs.

A *pseudo-split graph* is either a split graph, or a graph whose vertex set can be partitioned into a clique K, an independent set I, and a set H that (1) induces a C_5; (2) is complete to K; and (3) is nonadjacent to I. We say that $K \uplus I \uplus H$ is a *pseudo-split partition* of the graph, where H may or may not be empty. If H is empty, then $K \uplus I$ is a split partition of the graph. When H is nonempty, the pseudo-split partition is unique.

For pseudo-split graphs, we start with checking whether the input graph can be switched to a split graph. We are done if the answer is “yes.” Henceforth, we are looking for a resulting graph that contains a hole C_5. Suppose that G is in the upper pseudo-split switching class. Let A be a solution, and $K \uplus I \uplus H$ is a *pseudo-split partition* of $S(G,A)$. We may assume that $|A \cap H| \geq 3$: otherwise, we replace A with $V(G) \setminus A$. The subgraph $G[H]$ must be one of Figure 2b, and $A \cap H$ are precisely the vertices represented as empty nodes. We can guess the vertex set H as well as its partition with respect to A, and then all the other vertices are fixed by the following observation:

- K is complete to $H \cap A$ and nonadjacent to $H \setminus A$, and
- I is complete to $H \setminus A$ and nonadjacent to $H \cap A$.

The algorithm is described in Figure 5. We can modify the algorithm so that it enumerates all solutions.

```
1.        if $G$ can be switched to a split graph then return “yes”;
2.        for each vertex set $H$ such that $G[H] \in \mathcal{S}(C_5)$ do
2.0.          $H_1 \leftarrow$ the empty nodes of $G[H]$ as in Figure 2b; $H_2 \leftarrow H \setminus H_1$;
2.1.          for each vertex $x$ in $V(G) \setminus H$ do
2.1.1.            if $N(x) \cap H$ is neither $H_1$ nor $H_2$ then continue;
2.2.          if $N(H_1) \setminus H$ does not induce a split graph then continue;
2.3.          if $N(H_2) \setminus H$ does not induce a split graph then continue;
2.4.          for each split partition $K_1 \uplus I_1$ of the subgraph induced by $N(H_1) \setminus H$ do
2.4.1.            for each split partition $K_2 \uplus I_2$ of the subgraph induced by $N(H_2) \setminus H$ do
2.4.1.1.              if $S(G, H_1 \cup K_1 \cup I_2)$ is a pseudo-split graph then return “yes”;
3.        return “no”.
```

Figure 5 The algorithm for pseudo-split graphs.

▶ **Theorem 17.** *Let G be a graph. There are a polynomial number of subsets A of $V(G)$ such that $S(G, A)$ is a pseudo-split graph, and they can be enumerated in polynomial time.*

As a result, we have an algorithm for any hereditary subclass $\mathcal{G}$ of pseudo-split graphs that can be recognized in polynomial time. Since a graph has 2^n subsets, and the switching of only a polynomial number of them leads to a pseudo-split graph, every graph of sufficiently large order can be switched to a graph that is not a pseudo-split graph. Thus, the lower pseudo-split switching class is finite.

Next we give an algorithm for recognizing upper paw-free switching class. Since a paw contains an induced C_3 and an induced $\overline{P_3}$, both C_3-free graphs and $\overline{P_3}$-free graphs are paw-free. Olariu [27] showed that a connected paw-free graph is C_3-free or $\overline{P_3}$-free (i.e., complete multipartite). We start with checking whether G can be switched to a C_3-free graph [17] or a $\overline{P_3}$-free graph [24]. When the answers are both “no”, we look for a set $A \subseteq V(G)$ such that $S(G, A)$ is not connected and contains a triangle. It is quite simple when $S(G, A)$ has three or more components. We can always assume that A intersects two of them. We guess one vertex from each of these intersections, and an arbitrary vertex from another component (which can be in A or not). The three vertices are sufficient to determine A. It is more challenging when $S(G, A)$ comprises precisely two components. The crucial observation here is that one of the components is C_3-free and the other $\overline{P_3}$-free. We have assumed the graph contains a triangle. If both components contain triangles, hence $\overline{P_3}$-free, then $S(G, A)$ can be switched to a complete multipartite graph, contradicting the assumption above. We guess a triple of vertices that forms a triangle in $S(G, A)$, and they can determine A. The algorithm is described in Figure 6. A *co-component* of a graph G is a component of the complement of G. Indeed, a graph is complete multipartite if and only if every co-component is an independent set. With two tailored algorithms we prove that recognizing upper $\{K_{1,p}, \overline{K_{1,q}}\}$-free switching class and upper bipartite chain switching class can be solved in polynomial-time.

We end this section with the following remark. By Proposition 6(1), we know that recognizing $\mathcal{L}(\mathcal{G})$ is polynomially equivalent to recognizing $\mathcal{U}(\mathcal{G}^c)$. This implies polynomial-time algorithms for $\mathcal{U}(\mathcal{G}^c)$ for all the classes $\mathcal{G}$ for which we proved (in Section 3) the finiteness of $\mathcal{L}(\mathcal{G})$ or finiteness of the set of forbidden induced subgraphs of $\mathcal{L}(\mathcal{G})$. In particular, this implies that we have polynomial-time algorithms for recognizing upper non-planar switching class and upper non-chordal switching class.

```
1.        if $G$ can be switched to a $\overline{P_3}$- or $C_3$-free graph then return "yes";
2.        for each pair of nonadjacent vertices $u_1, u_2$ do // three or more components.
2.1.          for each $u_3 \in V(G) \setminus N[u_1, u_2]$ do
2.1.1.            $A \leftarrow \{x \in V(G) \mid |N[x] \cap \{u_1, u_2, u_3\} \leq 1\}$;
2.1.2.            if $S(G, A)$ is paw-free then return "yes";
2.2.          for each $u_3 \in N(u_1) \cap N(u_2)$ do
2.2.1.            $A \leftarrow (V(G) \setminus N[u_1, u_2]) \cup ((N[u_1] \Delta N[u_2]) \setminus N(u_3))$;
2.2.2.            if $S(G, A)$ is paw-free then return "yes";
3.        for each pair of adjacent vertices $u_1, u_2$ do // two components,
          one containing $C_3$.
3.1.          $p \leftarrow$ number of components of $G[N(u_1) \cap N(u_2)]$;
3.2.          $q \leftarrow$ number of components of $G - N[u_1, u_2]$;
3.3.          for each $I \subseteq \{1, \ldots, p\}$ and $J \subseteq \{1, \ldots, q\}$ with $|I|, |J| \leq 2$ do
3.3.1.            $X \leftarrow \bigcup_{i \notin I}$ $i$th co-component of $G[N(u_1) \cap N(u_2)]$;
3.3.2.            $Y \leftarrow \bigcup_{j \in J}$ $j$th co-component of $G - N[u_1, u_2]$;
3.3.3.            if $X \neq \emptyset$ then
3.3.3.1.              $u_3 \leftarrow$ an arbitrary vertex from $X$;
3.3.3.2.              $A \leftarrow X \cup Y \cup ((N(u_1) \Delta N(u_2)) \cap N(u_3))$;
3.3.4.            else
3.3.4.1.              $u_3 \leftarrow$ an arbitrary vertex from $V(G) \setminus (N[u_1, u_2] \cup Y)$;
3.3.4.2.              $A \leftarrow X \cup Y \cup ((N(u_1) \Delta N(u_2)) \setminus N(u_3))$;
3.3.5.            if $S(G, A)$ is paw-free then return "yes";
4.        return "no."
```

Figure 6 The algorithm for paw-free graphs.

5 Upper switching classes: hardness

In this section, we prove hardness results for recognition problems for $\mathcal{U}(\mathcal{G})$, for $\mathcal{G}$ being the class of P_{10}-free graphs or the class of C_7-free graphs. For convenience, we denote the recognition problem for $\mathcal{U}(\mathcal{G})$ as Switching-to-$\mathcal{G}$. We prove that Switching-to-$\mathcal{F}(P_{10})$ and Switching-to-$\mathcal{F}(C_7)$ are NP-complete and cannot be solved in time subexponential in the number of vertices, assuming the Exponential Time Hypothesis (ETH). We refer to the book [10] for an exposition to ETH and linear reductions which can be used to transfer complexity lower bounds.

Our reductions are from Monotone NAE k-SAT. A Monotone NAE k-SAT instance is a boolean formula Φ with n variables and m clauses where each clause contains exactly k positive literals (and no negative literals). The objective is to check whether there is a truth assignment to the variables so that there is at least one TRUE literal and at least one FALSE literal in each clause in Φ. It is folklore that the problem is NP-complete and cannot be solved in subexponential-time assuming ETH.

▶ **Proposition 18** (folklore). *For every $k \geq 3$, Monotone NAE k-SAT is NP-complete. Further, the problem cannot be solved in time $2^{o(n+m)}$, assuming ETH.*

We use the following construction for a reduction from Monotone NAE 5-SAT to Switching-to-$\mathcal{F}(P_{10})$.

▶ **Construction 1.** *Let Φ be a* Monotone NAE 5-SAT *formula with n variables $X_1, X_2, \cdots, X_n$, and m clauses $C_1, C_2, \cdots, C_m$. We construct a graph G_Φ as follows:*

- *For each variable X_i in Φ, introduce a variable vertex x_i. Let L be the set of all variable vertices, which forms an independent set of size n.*
- *For each clause C_i in Φ of the form $\{\ell_{i1}, \ell_{i2}, \ell_{i3}, \ell_{i4}, \ell_{i5}\}$, introduce a set of clause vertices, also named C_i, consisting of an independent set of size 5, denoted by I_i, and 5 disjoint P_9s each of which is denoted by B_{ij}, for $1 \leq j \leq 5$. Let $B_i = \bigcup_{j=1}^{5} B_{ij}$. The adjacency among the set B_{ij} and I_i, for $1 \leq j \leq 5$, is in such a way that the set of vertices in the P_9 induced by the B_{ij}, except one of the end vertex v_{ij}, is complete to I_i. Note that $C_i = B_i \cup I_i$. The set of union of all clause vertices is denoted by C. Let the 5 vertices introduced (in the previous step) for the variables $\ell_{i1}, \ell_{i2}, \ell_{i3}, \ell_{i4}, \ell_{i5}$ be denoted by $L_i = \{x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5}\}$. Make the adjacency between the vertices in L_i and the sets of P_9s in B_is in such a way that, taking one vertex from each set B_{ij} along with the variable vertices in L_i induces a P_{10}, where the vertices in L_i correspond to an independent set of size 5 in P_{10}. More precisely, x_{i1} is complete to B_{i1} and x_{ij} is complete to $B_{i(j-1)} \cup B_{ij}$, for $2 \leq j \leq 5$. Further, make the adjacency among the set I_i and L_i in such a way that, if exactly one of the set L_i or I_i is in the switching set A, then the vertices in $L_i \cup I_i$ together induce a P_{10} in $S(G_\Phi, A)$.*
- *For all $i \neq j$, C_i is complete to C_j.*

This completes the construction of the graph G_Φ (see Figure 7 for an example of the construction).

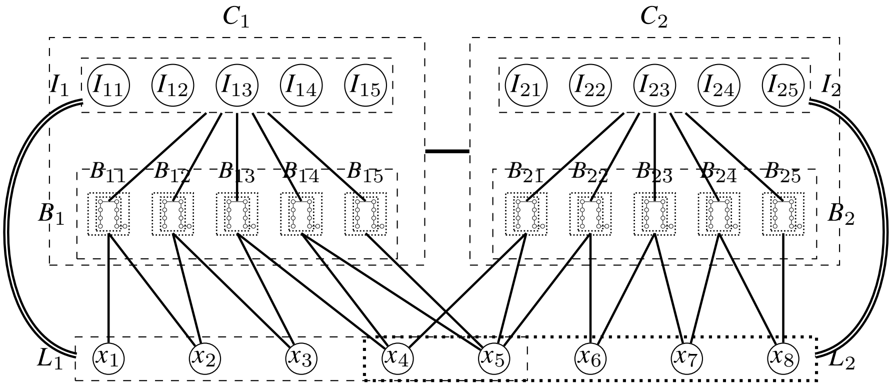


Figure 7 An example of Construction 1 with the formula $\Phi = C_1 \wedge C_2$, where $C_1 = \{x_1, x_2, x_3, x_4, x_5\}$ and $C_2 = \{x_4, x_5, x_6, x_7, x_8\}$. Single lines connecting two rectangles indicate that each vertex in one rectangle is adjacent to all vertices in the other rectangle. The double line connecting two rectangles indicates that each vertex in one rectangle is adjacent to the vertices in the other rectangle in such a way that if a switching set A contains all the vertices of one rectangle and no vertex of the other rectangle, then a P_{10} is induced by these two sets of vertices after switching.

We recall that the vertices in L_i and one vertex each from B_{ij}s ($1 \leq j \leq 5$) induce a P_{10}. If we have a truth assignment which satisfies Φ, then the vertices in L corresponding to the TRUE literals can be switched to obtain a P_{10}-free graph. The backward direction is easy and is proved in Lemma 19.

▶ **Lemma 19.** *Let Φ be an instance of* Monotone NAE 5-SAT. *If $S(G_\Phi, A)$ is P_{10}-free, for some $A \subseteq V(G_\Phi)$, then there exists a truth assignment satisfying Φ.*

Proof. We claim that assigning TRUE to the variables corresponding to the variable vertices in $A \cap L$ satisfies Φ. It is sufficient to prove that $A \cap L_i \neq \emptyset$ and $L_i \setminus A \neq \emptyset$, for every $1 \leq i \leq m$.

For a contradiction, assume that $A \cap L_i = \emptyset$, for some $1 \leq i \leq m$. Since L_i and one vertex each from B_{ij} induces a P_{10}, we obtain that $B_{ij} \subseteq A$, for some $1 \leq j \leq 5$. Then $I_i \subseteq A$ (otherwise, there is a P_{10} induced in $S(G_\Phi, A)$ by B_{ij} and a vertex in I_i not in A - recall that one end vertex v_{ij} of the P_9 formed by B_{ij} is not adjacent to I_i). Then at least one vertex from L_i is in A, otherwise there is a P_{10} induced in $S(G_\Phi, A)$ by $I_i \cup L_i$. This gives us a contradiction.

Next we show that L_i is not a subset of A. For a contradiction, assume that $L_i \setminus A = \emptyset$. Then at least one vertex $I_{i\ell} \in I_i$ (for some $1 \leq \ell \leq 5$) is in A - otherwise there is an P_{10} induced in $S(G_\Phi, A)$ by $L_i \cup I_i$. Then at least one vertex from each B_{ij} (for $1 \leq j \leq 5$) must be in A - otherwise there is a P_{10} induced in $S(G_\Phi, A)$ by $I_{i\ell}$ and B_{ij}, where $B_{ij} \cap A = \emptyset$. Then there is a P_{10} induced by L_i and one vertex, which is in A, from each B_{ij} (for $1 \leq j \leq 5$). This is a contradiction. ◀

With a similar reduction from MONOTONE NAE 3-SAT, we prove that SWITCHING-TO-$\mathcal{F}(C_7)$ is NP-complete and cannot be solved in subexponential-time.

6 Concluding remarks

There are many interesting questions one can ask about the characterization and computation of lower and upper switching classes of various graph classes. Here we list a few of them.

Since recognizing $\mathcal{U}(\mathcal{F}(P_{10}))$ and recognizing $\mathcal{U}(\mathcal{F}(C_7))$ are NP-complete, by Proposition 6(1), we obtain that recognizing $\mathcal{L}(\mathcal{G})$ is NP-complete, where $\mathcal{G}$ is the class of graphs containing an induced P_{10} or the class of graphs containing an induced C_7. Note that these classes are non-hereditary. For a hereditary graph class $\mathcal{G}$, is it true that whenever $\mathcal{G}$ is recognizable in polynomial-time, lower $\mathcal{G}$ switching class is also recognizable in polynomial-time? We know by Proposition 7 that this is true whenever $\mathcal{G}$ is characterized by a finite set of forbidden induced subgraphs.

Is it true that recognizing upper H-free switching class is polynomially equivalent to recognizing the upper H'-free switching class, where H and H' are switching equivalent? We know that the answer to the corresponding question for lower switching class is trivial, as both lower H-free and lower H'-free switching classes can be recognized in polynomial-time. In particular, can we recognize the upper H-free switching class in polynomial time when H is C_4, K_4, or diamond? For each of them, we know a switching equivalent H' such that the upper H'-free switching class can be recognized in polynomial time.

Let $\mathcal{G}$ be a graph class. Assume that, for any graph G, there are only polynomial number of ways to switch G to a graph in $\mathcal{G}$. Then every large enough graph G can be switched to a graph not in $\mathcal{G}$. Therefore, $\mathcal{L}(\mathcal{G})$ is finite. Is it true that whenever $\mathcal{L}(\mathcal{G})$ is finite, then $\mathcal{U}(\mathcal{G})$ can be recognized in polynomial-time?

What is the smallest integer ℓ such that the recognition of $\mathcal{U}(\mathcal{F}(P_\ell))$ is NP-complete? We know that $5 \leq \ell \leq 10$. Similarly, what is the smallest integer ℓ such that the recognition of $\mathcal{U}(\mathcal{F}(C_\ell))$ is NP-complete? We know that $4 \leq \ell \leq 7$.

References

1 Dhanyamol Antony, Yixin Cao, Sagartanu Pal, and R. B. Sandeep. Switching classes: Characterization and computation, 2024. arXiv:2403.04263.

2 Dhanyamol Antony, Jay Garchar, Sagartanu Pal, R. B. Sandeep, Sagnik Sen, and R. Subashini. On subgraph complementation to H-free graphs. *Algorithmica*, 84(10):2842–2870, 2022. doi:10.1007/s00453-022-00991-3.

3 Lowell W Beineke. Characterizations of derived graphs. *Journal of Combinatorial theory*, 9(2):129–135, 1970.

4 Hans L. Bodlaender and Jurriaan Hage. On switching classes, NLC-width, cliquewidth and treewidth. *Theor. Comput. Sci.*, 429:30–35, 2012. doi:10.1016/J.TCS.2011.12.021.

5 J. Adrian Bondy and F. Mercier. Switching reconstruction of digraphs. *J. Graph Theory*, 67(4):332–348, 2011. doi:10.1002/JGT.20535.

6 Peter J Cameron. Cohomological aspects of two-graphs. *Mathematische Zeitschrift*, 157:101–119, 1977.

7 Ying Cheng and Albert L. Wells Jr. Switching classes of directed graphs. *J. Comb. Theory, Ser. B*, 40(2):169–186, 1986. doi:10.1016/0095-8956(86)90075-4.

8 Václáv Chvátal. Set-packing and threshold graphs. *Res. Rep., Comput. Sci. Dept., Univ. Waterloo, 1973*, 1973.

9 Charles J. Colbourn and Derek G. Corneil. On deciding switching equivalence of graphs. *Discret. Appl. Math.*, 2(3):181–184, 1980. doi:10.1016/0166-218X(80)90038-4.

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 Andrzej Ehrenfeucht, Jurriaan Hage, Tero Harju, and Grzegorz Rozenberg. Complexity issues in switching of graphs. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Theory and Application of Graph Transformations, 6th International Workshop, TAGT'98, Paderborn, Germany, November 16-20, 1998, Selected Papers*, volume 1764 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 1998. doi:10.1007/978-3-540-46464-8_5.

12 Mark N. Ellingham and Gordon F. Royle. Vertex-switching reconstruction of subgraph numbers and triangle-free graphs. *J. Comb. Theory, Ser. B*, 54(2):167–177, 1992. doi:10.1016/0095-8956(92)90048-3.

13 Florent Foucaud, Hervé Hocquard, Dimitri Lajou, Valia Mitsou, and Théo Pierron. Graph modification for edge-coloured and signed graph homomorphism problems: Parameterized and classical complexity. *Algorithmica*, 84(5):1183–1212, 2022. doi:10.1007/S00453-021-00918-4.

14 Jurriaan Hage and Tero Harju. Acyclicity of switching classes. *Eur. J. Comb.*, 19(3):321–327, 1998. doi:10.1006/EUJC.1997.0191.

15 Jurriaan Hage and Tero Harju. A characterization of acyclic switching classes of graphs using forbidden subgraphs. *SIAM J. Discret. Math.*, 18(1):159–176, 2004. doi:10.1137/S0895480100381890.

16 Jurriaan Hage, Tero Harju, and Emo Welzl. Euler graphs, triangle-free graphs and bipartite graphs in switching classes. *Fundam. Informaticae*, 58(1):23–37, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi58-1-03.

17 Ryan B Hayward. Recognizing p3-structure: A switching approach. *journal of combinatorial theory, Series B*, 66(2):247–262, 1996.

18 Alain Hertz. On perfect switching classes. *Discret. Appl. Math.*, 94(1-3):3–7, 1999. doi:10.1016/S0166-218X(98)00153-X.

19 Eva Jelínková and Jan Kratochvíl. On switching to H-free graphs. *J. Graph Theory*, 75(4):387–405, 2014. doi:10.1002/jgt.21745.

20 Eva Jelínková, Ondrej Suchý, Petr Hlinený, and Jan Kratochvíl. Parameterized problems related to Seidel's switching. *Discret. Math. Theor. Comput. Sci.*, 13(2):19–44, 2011. doi:10.46298/DMTCS.542.

21 Alexandr V Kostochka. The minimum hadwiger number for graphs with a given mean degree of vertices. *Metody Diskret. Analiz.*, 38:37–58, 1982.

22 Alexandr V. Kostochka. Lower bound of the hadwiger number of graphs by their average degree. *Combinatorica*, 4(4):307–316, 1984.

23 Jan Kratochvíl. Complexity of hypergraph coloring and Seidel's switching. In Hans L. Bodlaender, editor, *Graph-Theoretic Concepts in Computer Science, 29th International Workshop,*

WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers, volume 2880 of *Lecture Notes in Computer Science*, pages 297–308. Springer, 2003. doi:10.1007/978-3-540-39890-5_26.

24 Jan Kratochvíl, Jaroslav Neŝetril, and Ondrej Zŷka. On the computational complexity of Seidel's switching. In *Annals of Discrete Mathematics*, volume 51, pages 161–166. Elsevier, 1992.

25 CL Mallows and NJA Sloane. Two-graphs, switching classes and euler graphs are equal in number. *SIAM Journal on Applied Mathematics*, 28(4):876–880, 1975.

26 Suho Oh, Hwanchul Yoo, and Taedong Yun. Rainbow graphs and switching classes. *SIAM J. Discret. Math.*, 27(2):1106–1111, 2013. doi:10.1137/110855089.

27 Stephan Olariu. Paw-fee graphs. *Inf. Process. Lett.*, 28(1):53–54, 1988. doi:10.1016/0020-0190(88)90143-3.

28 Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005. doi:10.1016/J.JCTB.2005.03.003.

29 Johan Jacob Seidel. Graphs and two-graphs. In *Proceedings 5th Southeastern Conference on Combinatorics, Graph Theory and Computing (Boca Raton FL, USA, 1974)*, pages 125–143, 1974.

30 Johan Jacob Seidel. A survey of two-graphs. In *Atti Convegno Internazionale Teorie Combinatorie (Rome, Italy, September 3-15, 1973), Tomo I.*, pages 481–511. Accademia Nazionale dei Lincei, 1976.

31 Johan Jacob Seidel and DE Taylor. Two-graphs, a second survey. In *Geometry and Combinatorics*, pages 231–254. Elsevier, 1991.

32 Richard P. Stanley. Reconstruction from vertex-switching. *J. Comb. Theory, Ser. B*, 38(2):132–138, 1985. doi:10.1016/0095-8956(85)90078-4.

33 Andrew Thomason. An extremal function for contractions of graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 95(2), pages 261–265. Cambridge University Press, 1984.

34 Andrew Thomason. The extremal function for complete minors. *J. Comb. Theory, Ser. B*, 81(2):318–338, 2001. doi:10.1006/JCTB.2000.2013.

35 Jacobus Hendricus van Lint and Johan Jacob Seidel. Equilateral point sets in elliptic geometry. *Indagationes Mathematicae, Series A: Mathematical Sciences*, 69:335–348, 1966. Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen.

Generalizing Roberts' Characterization of Unit Interval Graphs

Virginia Ardévol Martínez
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016 Paris, France

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

Abdallah Saffidine
University of New South Wales, Sydney, Australia

Florian Sikora
Université Paris-Dauphine, PSL University, CNRS, LAMSADE, 75016 Paris, France

Stéphane Vialette
LIGM, CNRS, Univ Gustave Eiffel, F77454 Marne-la-Vallée, France

Abstract

For any natural number d, a graph G is a (disjoint) d-interval graph if it is the intersection graph of (disjoint) d-intervals, the union of d (disjoint) intervals on the real line. Two important subclasses of d-interval graphs are unit and balanced d-interval graphs (where every interval has unit length or all the intervals associated to a same vertex have the same length, respectively). A celebrated result by Roberts gives a simple characterization of unit interval graphs being exactly claw-free interval graphs. Here, we study the generalization of this characterization for d-interval graphs. In particular, we prove that for any $d \geqslant 2$, if G is a $K_{1,2d+1}$-free interval graph, then G is a unit d-interval graph. However, somehow surprisingly, under the same assumptions, G is not always a *disjoint* unit d-interval graph. This implies that the class of disjoint unit d-interval graphs is strictly included in the class of unit d-interval graphs. Finally, we study the relationships between the classes obtained under disjoint and non-disjoint d-intervals in the balanced case and show that the classes of disjoint balanced 2-intervals and balanced 2-intervals coincide, but this is no longer true for $d > 2$.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Interval graphs, Multiple Interval Graphs, Unit Interval Graphs, Characterization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.12

Related Version *Full Version*: `https://arxiv.org/abs/2404.17872`

Supplementary Material
Software (Source Code): `https://github.com/AbdallahS/unit-graphs` [26]
archived at `swh:1:dir:3d7b6495d1f70618a537cd23c94530c23c030215`

Funding This work was partially supported by the ANR project ANR-21-CE48-0022 ("S-EX-AP-PE-AL").

Acknowledgements Part of this work was conducted when RR was an invited professor at Université Paris-Dauphine.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 12; pp. 12:1–12:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Interval graphs are the intersection graphs of intervals on the real line: every vertex represents an interval and there is an edge between two vertices if and only if their corresponding intervals intersect. The class of interval graphs is one of the most important classes of intersection graphs, mostly due to their numerous applications in scheduling or allocation problems and in bioinformatics, see for examples these monographs [11, 23, 25].

Already in the late 70s, situations arising naturally in scheduling and allocation motivated the generalization of interval graphs to *multiple interval graphs*, where every vertex is associated to the union of d intervals on the real line (called a d-interval), for some natural number d, instead of to a single interval. This allowed a more robust modeling of problems such as multi-task scheduling or allocation of multiple associated linear resources [15, 22, 29], and led to several interesting problems [10, 12, 17, 18, 19]. The applications of 2-interval graphs to bioinformatics also increased the interest on this class of graphs [20, 30].

These concrete applications of multiple interval graphs, specifically 2-interval graphs, suggested a focus on different restrictions, such as unit 2-interval graphs [2], or balanced 2-interval graphs [7]. For both interval and multiple interval graphs, we say that they are *unit* if all the intervals in the representation, i.e. the set of intervals associated to the graph, have unit length. For multiple interval graphs, we also define the subclass of *balanced* d-interval graphs, where all intervals forming the same d-interval have equal length, but intervals of different d-intervals can have different lengths. Finally, for both interval and multiple interval graphs, we say that they are *proper* if there exists an interval representation where no interval properly contains another one. The class of unit 2-interval graphs is known to be properly contained in the class of balanced 2-interval graphs [13].

Let us remark that in the literature, d-intervals have been defined both as the union of d *disjoint* intervals [2, 5, 31], as the union of d *not necessarily disjoint* intervals [29], and simply as the union of d intervals, without specifying whether they are disjoint or not [9, 27]. This ambiguity is not relevant in the general case, since both definitions lead to the same class of graphs. However, in this paper we focus on subclasses of multiple interval graphs, namely unit and balanced, for which this equivalence is not known to be true. Therefore, we will distinguish between the two possible definitions of d-intervals. The first definition is denoted as *disjoint d-intervals* while the second is simply denoted as *d-intervals* (further details are discussed in Section 2).

From an algorithmic perspective, another reason why interval graphs have been widely studied is because many problems that are NP-hard become solvable in polynomial time when restricted to this class of graphs. This is not the case for d-interval graphs [2, 5, 12]. The problem of recognizing d-interval graphs is no exception: it is NP-complete for every natural number $d \geqslant 2$ [31], even for unit 2-interval graphs [1] and balanced 2-interval graphs [13]. In sharp contrast, the recognition of interval graphs (both in the unit and unrestricted case) can be done in polynomial time [4, 16], and there exist multiple characterizations of them, including a characterization in terms of forbidden induced subgraphs [21, 24]. In particular, in 1969, Roberts proved that the class of proper interval graphs and the class of unit interval graphs coincide [24], and showed that unit interval graphs are exactly $K_{1,3}$-free interval graphs (i.e., interval graphs that do not contain the star with three leaves as an induced subgraph). To do so, he used the Scott-Suppes characterization of semiorders (see [3, 14] for short constructive proofs of this result). This is a remarkable result as it gives a simple characterization of unit interval graphs. It also implies that if $G = (V, E)$ is a unit interval graph, then there exists a semiorder $S(V, P)$ on the vertices of V such that $(u, w) \in P$ if and only if $(u,w) \notin E$, which justifies the original name of "indifference graphs" for unit interval graphs (as they can represent indifference relations by joining two elements by an edge if neither is preferred over the other one).

It is straight-forward to check that being $K_{1,3}$-free is a necessary condition for being a unit interval graph, as an interval of unit length cannot intersect three pairwise disjoint intervals of length one. The reader can observe that this necessary condition extends naturally to multiple interval graphs: a unit 2-interval graph cannot contain a $K_{1,5}$ as an induced subgraph; and more generally, a unit d-interval graph cannot contain a $K_{1,2d+1}$ as an induced subgraph. Thus the following natural question arises: can we generalize Roberts characterization of unit interval graphs to multiple interval graphs? Perhaps the most straight-forward generalization would be to characterize unit d-interval graphs as $K_{1,2d+1}$-free d-interval graphs, but this has already been proven false in [28]: there exists a graph which is 2-interval and $K_{1,5}$-free, but not unit 2-interval. But not all hope of generalizing Roberts characterization must be lost yet! What if we add some additional constraints?

Already in 2016, Durán et al. decided to focus on d-interval graphs which are also *interval* [8]. In a presentation at VII LAWCG, they claimed that if G is an interval graph, then G is a disjoint unit d-interval graph if and only if it is $K_{1,2d+1}$-free [1]. In this paper, we show that the aforementioned statement is actually false, and that, perhaps surprisingly, Roberts characterization can only be generalized depending on the chosen definition of d-interval graphs! (See Figure 1 for a summary of the main results).

We also study the subclasses obtained under the two definitions of d-intervals in the balanced case, expanding the knowledge of the relationships between the different subclasses of 2-interval graphs.

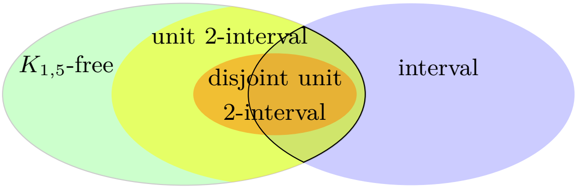


Figure 1 $K_{1,5}$-free interval graphs are not contained in the class of disjoint unit 2-interval graphs. The class of unit 2-interval graphs is a superclass of disjoint unit 2-interval graphs, and spans the whole intersection of $K_{1,5}$-free and interval graphs.

The structure of the paper is as follows: Section 2 briefly introduces the necessary definitions and discusses the definition of d-interval. In Section 3, we prove that if G is an interval graph, then it is unit d-interval if and only if it is $K_{1,2d+1}$-free. We then show that this result cannot be generalized for *disjoint* multiple intervals in Section 4, which implies that the class of disjoint unit d-interval graphs is actually properly contained in the class of unit d-interval graphs. Finally, we study the balanced case in Section 5, and show that the definition of d-interval also matters, as the classes of disjoint balanced 2-intervals and balanced 2-intervals coincide, but this is no longer true for $d > 2$. We conclude with some open questions in Section 6. Due to space constraints, some proofs, marked with a $(\star)$, are deferred to the full version of this paper.

[1] Note that they refer to disjoint unit d-intervals simply as unit d-intervals, but they are explicitly defined beforehand as the union of d disjoint intervals.

2 Preliminaries

In the following, $G = (V, E)$ will denote a simple undirected graph on the set of vertices V and with edges E, and an interval will be a set of real numbers of the form $[a, b] := \{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$.

A graph G is an *interval graph* if there exists a bijection from the vertices of G to a multiset of intervals, $f : V \to \mathcal{I}$, such that there exists an edge between two vertices if and only if their corresponding intervals intersect. The multiset $\mathcal{I}$ is called an *interval representation* of G.

For any natural number $d > 0$, a (disjoint) *d-interval* is the union of d (disjoint) intervals on the real line.

For any natural number $d > 0$, a graph G is a (disjoint) *d-interval graph* if there exists a bijection from the vertices of G to a multiset of (disjoint) d-intervals, $f : V \to \mathcal{I}$, such that there exists an edge between two vertices if and only if their corresponding d-intervals intersect. The multiset $\mathcal{I}$ of d-intervals is called a d-interval representation of G, and the family of all intervals that compose the d-intervals in $\mathcal{I}$ is called the *underlying family of intervals* of $\mathcal{I}$.

A (disjoint) d-interval graph is *unit* if there exists a (disjoint) d-interval representation where all the intervals of the underlying family have unit length, and it is *proper* if there exists a representation where no interval of the underlying family is properly contained in another one. A (disjoint) d-interval graph is *balanced* if there exists a (disjoint) d-interval representation where the d intervals of a same d-interval have the same length, but intervals of different d-intervals can differ in length.

The graph $K_{1,t}$ is the star with t leaves (also referred to as *t-claw* in the following). For any $t \geqslant 3$, if the set of vertices $\{v_0, v_1, \ldots, v_t\}$ induces a $K_{1,t}$ with center v_0, we will denote it by $[v_0; v_1, \ldots, v_t]$. We say that a graph is $K_{1,t}$-free if it does not contain any induced $K_{1,t}$'s. Furthermore, we say that an induced t-claw $K_{1,t}$ is *maximal* if it is not contained in an induced $K_{1,m}$ with $m > t$.

Discussion on the definition of d-intervals. As mentioned in the introduction, d-interval graphs have been defined in the literature both as the union of d disjoint intervals and as the union of d not necessarily disjoint intervals. This might be related to the fact that when there are no length restrictions on the intervals, both definitions lead to the same class of graphs (as one can simply stretch the intervals associated to a same vertex that intersect to make them disjoint without changing any of the other intersections).

▶ **Observation 1.** ($\star$) *The classes of disjoint d-interval and d-interval graphs are equivalent.*

However, if there are length restrictions, the previous observation does not hold. For unit intervals, one cannot replace two intersecting intervals $[a, b]$ and $[c, d]$, with $a < c < b < d$, by $[a, d]$, as the resulting interval would not be of unit length, and stretching it to make it unit might disrupt the rest of the intersections. Thus, in this case, it cannot be inferred that both definitions of multiple intervals lead to the same class of graphs. In fact, our results prove that they do not. Therefore, we study the generalization of Roberts characterization separately for both definitions of d-intervals.

3 Unit d-interval graphs

In this section, we generalize Roberts characterization of unit interval graphs for d-interval graphs. Recall that by d-interval graphs we refer to intersection graphs of d-intervals where the d intervals are not necessarily disjoint, or in other words, to the most general definition.

▶ **Theorem 2.** *Let G be an interval graph. Then, for any natural number $d \geqslant 2$, G is a unit d-interval graph if and only if G does not contain a copy of a $K_{1,2d+1}$ as an induced subgraph. Furthermore, given a $K_{1,2d+1}$-free interval graph, a unit d-interval representation can be constructed in $\mathcal{O}(n+m)$ time, where n and m are the number of vertices and edges of the graph, respectively.*

We present a polynomial-time algorithm that, given an arbitrary interval representation $\mathcal{I}$ of a $K_{1,2d+1}$-free graph, returns a d-interval representation $\mathcal{I}'$ of the graph where no interval of the underlying family of $\mathcal{I}'$ intersects three or more pairwise disjoint intervals. This ensures that the underlying family of intervals returned corresponds to an interval representation of a $K_{1,3}$-free graph, so we can use the algorithm described in [3] to turn it into a proper representation (and then to a unit one in linear time [14]). Note that if an interval representation of the graph is not given, we can always compute it in linear time [6].

Before presenting the algorithm formally, let us give the idea behind it. The algorithm constructs a family $\mathcal{I}'$ of d-intervals in the following way: for every interval $I \in \mathcal{I}$ that intersects m (and no more than m) pairwise disjoint intervals, we create a t-interval $I_1 \cup \ldots \cup I_t$, where $t = \lceil \frac{m}{2} \rceil$. Note that for every interval I that intersect only two disjoint intervals, we have $t = 1$, and the interval I_1 added to $\mathcal{I}'$ will be exactly I. We will refer to such intervals as *original intervals*, as they are equal to the ones in $\mathcal{I}$. After creating the t-intervals described above, to obtain a d-interval representation of the graph, it suffices to add $d-t$ "dummy" intervals for each vertex that is represented by $t < d$ intervals (where by "dummy" intervals we mean that they do not intersect any other interval from the representation). Each d-interval $I_1 \cup \ldots \cup I_d$ introduced will preserve the same intersections as the interval $I \in \mathcal{I}$, and each I_i will possess three key properties: it intersects at most two disjoint original intervals, it contains an original interval, and each of its endpoints coincides with an endpoint of an original interval. These properties ensure that the representation $\mathcal{I}'$ can be made unit.

Algorithm

Let the family of intervals $\mathcal{I}$ be an interval representation of G. For every interval $I \in \mathcal{I}$, let $l(I)$ and $r(I)$ stand for its left and right endpoint, respectively. Furthermore, define a partial order as follows: given two intervals $I, J \in \mathcal{I}$, let $I \prec J$ if and only if $r(I) < l(J)$ (i.e. interval J is fully to the right of interval I). Two intervals are incomparable if they intersect.

Step 1 Initialize a set of intervals $\mathcal{C}$ with all the intervals of $\mathcal{I}$, set $\mathcal{I}' := \emptyset$, and go to **Step 2**.

Step 2 Pick an interval I of $\mathcal{C}$, remove it from the set and define its neighborhood $\mathcal{N}(I) = \{J \in \mathcal{I} \colon J \cap I \neq \emptyset\}$. Let m be the maximum number of pairwise disjoint intervals that I intersects. If $m \leqslant 2$, go to **Step 3**; if $m = 3$, go to **Step 4**; and if $m > 3$, go to **Step 5**.

Step 3 If $m \leqslant 2$, add the interval $I_1 = I$ to the family $\mathcal{I}'$ and call I_1 an original interval. Then go to **Step 6**.

Step 4 If $m = 3$, define four auxiliary intervals:

$$A_1 = \operatorname*{arg\,min}_{J \in \mathcal{N}(I)} \{r(J)\} \qquad A_2 = \operatorname*{arg\,min}_{\{J \in \mathcal{N}(I) \colon A_1 \prec J\}} \{r(J)\}$$

$$A_4 = \operatorname*{arg\,max}_{J \in \mathcal{N}(I)} \{l(J)\} \qquad A_3 = \operatorname*{arg\,max}_{\{J \in \mathcal{N}(I) \colon J \prec A_4\}} \{l(J)\}$$

Then add to $\mathcal{I}'$ the 2-interval $I_1 \cup I_2$, with $I_1 = [l(I), r(A_2)]$ and $I_2 = [l(A_3), r(I)]$. Note that A_2 and A_3 necessarily intersect, as otherwise we would have $m \geqslant 4$, so $I_1 \cup I_2$ is not a disjoint 2-interval. After adding it to $\mathcal{I}'$, go to **Step 6**.

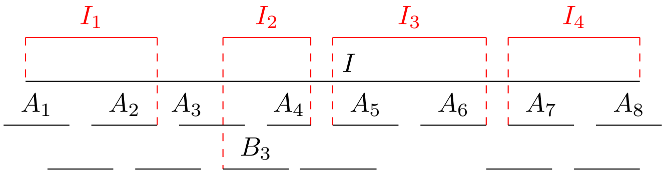


Figure 2 Interval I intersects 8 disjoint intervals. In red, the 4-interval returned by the algorithm. Note that if $l(I_2)$ were defined as $l(A_3)$ instead of $l(B_3)$, it would create a forbidden $K_{1,3}$.

Step 5 If $m > 3$, define two families of auxiliary intervals. The first family $\mathcal{A} := \{A_i \,|\, i \in \{1, \dots, m\}\}$ forms a maximum set of pairwise disjoint intervals intersecting I, and it will ensure that all the intersections are preserved. It is defined as follows:

$$A_1 = \underset{J \in \mathcal{N}(I)}{\arg\min}\{r(J)\} \qquad A_i = \underset{\{J \in \mathcal{N}(I) \colon A_{i-1} \prec J\}}{\arg\min}\{r(J)\}\ , \forall\ i \in \{2, \dots, m-2\}$$

$$A_m = \underset{J \in \mathcal{N}(I)}{\arg\max}\{l(J)\} \quad A_{m-1} = \underset{\{J \in \mathcal{N}(I) \colon J \prec A_m\}}{\arg\max}\{l(J)\}$$

The second family $\mathcal{B} := \{B_i \,|\, i \in \{1, \dots, m\}\}$ is a tool to ensure that each new interval I_i intersects only two disjoint intervals in $\mathcal{I}'$. Note that restricting each I_i to intersect only two disjoint intervals from the family $\mathcal{A}$ is not enough: for example, in Figure 2, if I_2 were defined as $[l(A_3), r(A_4)]$, then it would intersect three pairwise disjoint intervals in $\mathcal{I}'$ (as all the intervals except I are original intervals in this example), whereas if the left endpoint of I_2 were chosen as $r(A_{2i-1})$, then an original interval that was not the center of a claw in $\mathcal{I}$ might become the center of a new claw in $\mathcal{I}'$. Thus, for every $i \in \{1, \dots, m\}$, B_i is defined as follows:

$$B_i = \underset{J \in \mathcal{N}(A_i) \cup A_i}{\arg\max}\{l(J)\}$$

In other words, B_i is the interval in the closed neighborhood of A_i starting the latest. Note that if there does not exist any interval intersecting A_i which starts after A_i, then $B_i = A_i$ since we are considering the closed neighborhood. Now, add to $\mathcal{I}'$ the t-interval $I_1 \cup ... \cup I_t$, defined as follows. We distinguish two slightly different cases:

a. If m is even, i.e., $m = 2t$ for some $t > 1$, define $I_1 = [l(I), r(A_2)]$, $I_i = [l(B_{2i-1}), r(A_{2i})]$ for every $i \in \{2, \dots, t-1\}$, and $I_t = [l(A_{2t-1}, r(I)]$.

b. If m is odd, i.e., $m = 2t - 1$ for $t > 2$, define I_{t-1} and I_t differently, as $I_{t-1} = A_{2t-3}$ and $I_t = [l(A_{2t-2}, r(I)]$, and the rest of the intervals as before.

Notice that by definition, the intervals $I_1, ..., I_t$ are actually pairwise disjoint, so if $m > 3$, the t-interval added to $\mathcal{I}'$ is a disjoint d-interval. After adding the t-interval, go to **Step 6**.

Step 6 If $\mathcal{C} = \emptyset$, return $\mathcal{I}'$, else go to **Step 2**.

Figure 2 illustrates the algorithm on a concrete interval which intersects eight pairwise disjoint intervals. Before proceeding to the proof of correctness of the algorithm, we highlight the properties of the intervals constructed that will be useful to prove the next three claims. In the following, we say that an interval I of $\mathcal{I}$ has been *transformed* into a t-interval $I_1 \cup \cdots \cup I_t$ by the algorithm after it has been processed.

▶ **Observation 3.** *Let $I \in \mathcal{I}$ be an interval transformed into $I_1 \cup \cdots \cup I_t$ by the algorithm, for some $1 < t \leqslant d$. Then, for every $i \in \{1, \ldots, t\}$:*

1. *The left (resp., right) endpoint of every interval I_i coincides with the left (resp., right) endpoint of an original interval.*
2. *There is an original interval contained in I_i.*

Now, to prove the correctness of the algorithm, we need to show that for every interval $I \in \mathcal{I}$, the t-interval $I_i \cup ... \cup I_t \in \mathcal{I}'$ preserves the same intersections as I, and that no interval in the underlying family of $\mathcal{I}'$ intersects three pairwise disjoint intervals. In the next claim, we prove that intersections are preserved:

▷ Claim 4. Let I be an interval transformed into $I_1 \cup \cdots \cup I_t$ by the algorithm, for some $1 \leqslant t \leqslant d$. Then, the t-interval $I_1 \cup \ldots \cup I_t$ preserves the intersections of I.

Proof. It is clear that no new intersections are created as $I_1 \cup \ldots \cup I_t \subseteq I$. To see that no intersection is lost, suppose that there exists an interval L that intersects I in the original representation $\mathcal{I}$, and after the algorithm finishes, L is transformed into a t_0-interval $L_1 \cup \ldots \cup L_{t_0}$ (for some $1 \leqslant t_0 \leqslant d$, where if $t_0 = 1$, the interval remains as in the original representation) such that the t-interval $I_1 \cup \ldots \cup I_t$ does not intersect the t_0-interval $L_1 \cup \ldots \cup L_{t_0}$ in $\mathcal{I}'$. Since $l(I_1) = l(I)$ and $r(I_t) = r(I)$ (and the same holds for L), this means that there exists an L_j (with $1 \leqslant j \leqslant t_0$) such that $I_i \prec L_j \prec I_{i+1}$ for some $1 \leqslant i \leqslant t-1$.

For $1 \leqslant t \leqslant 2$, this cannot occur because $I \subseteq I_1 \cup \ldots \cup I_t$. For $t > 3$, since the set of intervals A_k used to defined the t-interval associated to I forms a maximal set of pairwise disjoint intervals intersecting I, we cannot have that $A_k \prec L_j \prec A_{k+1}$ for any $1 \leqslant k \leqslant 2t-1$. Indeed, this would contradict maximality, as L_j is either an original interval or it contains an original interval (by Observation 3). Thus, the only possible option is that there exists an i such that $A_{2i} \prec L_j \prec B_{2i+1}$ (where B_{2i+1} is different from A_{2i+1}). Then, since B_{2i+1} intersects A_{2i+1} and $L_j \prec B_{2i+1}$, we have that $r(L_j) < r(A_{2i+1})$. But this contradicts the choice of A_{2i+1}, which should have been L_j or the original interval contained in L_j, as $A_{2i} \prec L_j$. ◁

The next two claims are dedicated to proving that no interval in the underlying family of $\mathcal{I}'$ intersects three or more pairwise disjoint intervals. We distinguish the cases when the center of the claw is an original interval and when it is not.

▷ Claim 5. Let $I \in \mathcal{I}$ be an original interval (i.e., transformed to I_1 by the algorithm). Then, I_1 intersects at most two disjoint intervals in the underlying family of $\mathcal{I}'$.

Proof. Suppose, towards a contradiction, that there exists an original interval I_1 that intersects three pairwise disjoint intervals L_1, L_2 and L_3 in the underlying family of $\mathcal{I}'$, with $L_1 \prec L_2 \prec L_3$. By Observation 3, there exists an original interval L_1' with the same right endpoint as L_1, an original interval L_2' contained in L_2, and an original interval L_3' with the same left endpoint as L_3. Note that if any of the L_i are original, then $L_i' = L_i$. But then, $L_1' \prec L_2' \prec L_3'$ are three pairwise disjoint original intervals that intersect I_1, which contradicts the fact that it is an original interval. Indeed, this implies that the interval I intersects three pairwise disjoint intervals in $\mathcal{I}$, and so the algorithm would have transformed it into a t-interval with t strictly greater than 1. ◁

▷ Claim 6. Let $I \in \mathcal{I}$ be an interval transformed into the t-interval $I_1 \cup \cdots \cup I_t$ by the algorithm, for some $1 < t \leqslant d$. For every $1 \leqslant i \leqslant t$, I_i intersects at most two disjoint intervals of the underlying family of $\mathcal{I}'$.

Proof. We proceed by contradiction. Suppose that there exists an interval I_i, with $1 \leqslant i \leqslant t$ that intersects three pairwise disjoint intervals L_1, L_2, L_3, with $L_1 \prec L_2 \prec L_3$. By Observation 3, there exists an original interval L'_1 with the same right endpoint as L_1, an original interval L'_2 contained in L_2, and an original interval L'_3 with the same left endpoint as L_3.

Assume first that $t = 2$. Then, if $i = 1$, this contradicts the choice of the interval A_2 (resp. A_3 if $i = 2$), which should have been L'_2.

Let us now study the general case for $t > 2$. Suppose first that $1 < t < d$ and I_i is defined as $[B_{2i-1}, A_{2i}]$ with $B_{2i-1} \neq A_{2i-1}$. We distinguish two cases:

1. $r(L_1) > r(A_{2i-1})$. Then, since we are assuming that L_1 and L_2 are disjoint, $l(L_2) > r(A_{2i-1})$. Furthermore, as L_3 also intersects I_i, we need $r(L_2) < r(A_{2i})$. But this contradicts the choice of A_{2i}, which should have been L'_2.
2. $r(L_1) < r(A_{2i-1})$. If $l(L_2) > r(A_{2i-1})$, we are in the same case as before. Thus, L_2 and A_{2i-1} must intersect. However, we have $l(L_2) > l(B_{2i-1})$ (since otherwise I_i would not be able to intersect L_1 on its left extreme). This contradicts the choice of B_{2i-1} if L'_2 intersects A_{2i-1}, or the choice of A_{2i} otherwise.

On the other hand, if $B_{2i-1} = A_{2i-1}$, then by construction, since we take the two disjoint intervals that finish first, we cannot have three pairwise disjoint intervals intersecting I_i. This is also the case for I_1 and I_t (although in the latter case, we take the two disjoint intervals starting last). Finally, for odd claws, it is also clear that I_{t-1} intersects at most two disjoint intervals, as it is equal to an original interval. ◁

Combining Claims 4, 5 and 6, plus the fact that we can trivially transform a t-interval with $t < d$ into a d-interval, we obtain that the algorithm returns a d-interval representation of the input graph where no interval of the underlying family intersects more than two disjoint intervals, which as explained before can be converted into a unit representation. The last part of Theorem 2 follows because an efficient implementation of the algorithm described above requires $\mathcal{O}(1 + deg(v))$ operations for each vertex v (where $deg(v)$ denotes the degree of vertex v), as it suffices to iterate over the neighborhood of a given interval to transform it into the corresponding d-interval. Finally, the obtained representation can be converted to a unit representation in linear time, which yields the stated runtime $\mathcal{O}(n + m)$. This concludes the proof of Theorem 2.

We have proven that the algorithm constructs a unit d-interval representation, but it is not a disjoint one. Indeed, as mentioned before, in the case of maximal $K_{1,3}$'s, the constructed intervals I_1 and I_2 intersect each other. However, in the case of maximal $K_{1,m}$'s with $m > 3$, the t intervals of the t-interval created are actually pairwise disjoint. Thus, we obtain as a direct corollary that if G is a $K_{1,2d+1}$-free interval graph not containing any maximal $K_{1,3}$'s, then G is a disjoint unit d-interval graph. In fact, with a more careful analysis, we can infer an even stronger corollary, which instead of requiring the absence of maximal 3-claws altogether, only forbids a subset of them. We refer to these forbidden claws, which are exactly those maximal 3-claws contained in an induced E graph, as E-claws. Recall that an E graph (or star$_{1,2,2}$) is a graph on six vertices which has as edge set a path v_1, v_2, v_3, v_4, v_5 and an additional edge (v_3, v_6).

▶ **Corollary 7.** *Let G be a $K_{1,2d+1}$-free graph that does not contain any E-claws. Then, G is a disjoint unit d-interval graph.*

Proof. To prove the theorem, we modify **Step 4** of the previous algorithm so that it produces a disjoint 2-interval.

Step 4' Let I be an interval and let $m = 3$ be the maximum number of pairwise disjoint intervals that it intersects. By assumption, the vertex associated to I is a center of a maximal claw which is not an E-claw. We define

$$A_1 = \arg\min_{J \in \mathcal{N}(I)} \{r(J)\}$$

$$A_2 = \arg\min_{\{J \in \mathcal{N}(I) \colon A_1 \prec J\}} \{r(J)\}$$

$$A_4 = \arg\max_{J \in \mathcal{N}(I)} \{l(J)\}$$

$$A_3 = \arg\max_{\{J \in \mathcal{N}(I) \colon J \prec A_4\}} \{l(J)\}$$

Note that A_2 and A_3 necessarily intersect (or are the same interval). Now, since the vertex associated to I is a center of a claw that is not an E-claw, this means that at least one of A_1 or A_4 does not intersect an interval which is disjoint from I. Thus, we can modify the representation so that A_1 (resp. A_4) is properly contained in I without loosing any intersections, by simply stretching them. Then, if A_1 is properly contained in I, we define $I_1 = A_1$ and $I_2 = [l(A_3), r(I)]$. On the other hand, if A_4 is properly contained in I instead, we define $I_1 = [l(I), r(A_2)]$ and $I_2 = A_4$. If both of them are properly contained in I, we can define I_1 and I_2 either way.

Notice that the 2-intervals introduced in this step have the same properties as in Observation 3, so the proof of correctness of the previous algorithm can be directly adapted for this extension. ◀

4 Disjoint unit d-interval graphs

In this section, we prove that Theorem 2 cannot be generalized for disjoint unit d-interval graphs. Note that by Corollary 7, if we have a graph which does not contain any E-claws, then the generalization still holds for disjoint unit d-interval graphs, but this is not the case in general. Indeed, suppose there is an interval I that intersects exactly three pairwise disjoint intervals, A_1, A_2 and A_3, and both A_1 and A_3 intersect each an interval disjoint from I. Then, the algorithm presented in the previous section would return a 2-interval $I_1 \cup I_2$, where I_1 and I_2 are not disjoint. If we try to extend the algorithm in the most natural way, that is, stretching these two intervals until they are disjoint, we would still not succeed. This is because, since I_1 and I_2 cannot intersect, then either $r(I_1)$ will be to the left of the right endpoint returned by the algorithm, or $l(I_2)$ will be to the right of the endpoint returned by the algorithm. But then, one of I_1 or I_2 might not properly contain a complete interval from the original representation, which can cause I_1 or I_2 – the interval which does not properly contain a complete original interval – to be contained in an interval that intersects three pairwise disjoint intervals (see Figure 3).

In the following, we show that there is no way to extend the algorithm to make it work in the general case for disjoint unit d-interval graphs. In particular, we prove the following theorem.

▶ **Theorem 8.** *There exists a $K_{1,5}$-free interval graph that is not a disjoint unit 2-interval graph.*

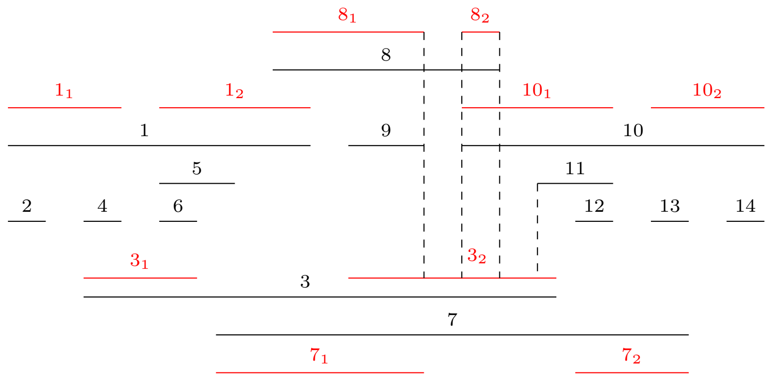


Figure 3 Interval representation of a $K_{1,5}$-free graph that cannot be turned into a disjoint unit 2-interval representation just by "cutting" intervals that intersect more than three pairwise disjoint intervals. In the figure, the intervals in red are all obtained using a natural extension of the algorithm. We can see that in this way, 3_2 intersects three disjoint intervals: $8_1, 8_2, 11$. The reader can check that no other way of stretching the intervals works if 8_1 and 8_2 are required to be disjoint.

To prove Theorem 8, we offer the graph G in Figure 4 as a certificate. The reader can check that G has no induced $K_{1,5}$, and an interval representation of G is provided in Figure 5. The proof that G is not a disjoint unit 2-interval graph is the challenging part. Indeed, checking whether a graph is disjoint unit 2-interval is a computationally expensive task, and even with the aid of computer search, a naive ILP implementation already takes too much time to consider an exhaustive search. Needless to say, checking manually by brute force leads to a very long branching process. The proof presented here is based on a careful analysis of the graph, and the technique employed (which uses the characterization of unit 2-interval graphs in [[1], Lemma 5]) may be applied to establish that other graphs are not disjoint unit 2-interval graphs. We also verify the proof computationally, using an encoding in answer set programming based on the semiorder characterization of unit interval graphs, which proves to be way more efficient than an ILP encoding. Our code and experimental setting can be found on our git repository [2]. Furthermore, there exist five other $K_{1,5}$-free interval graphs on the same number of vertices, and with a very similar structure, that are not disjoint unit 2-interval (see the full version of this paper). The proof that they are not disjoint unit 2-interval is omitted, but it is analogous to the one presented here. These six graphs are the only such graphs on 14 vertices, and there does not exist a graph satisfying the conditions of Theorem 8 with fewer vertices. These assertions were verified by computer search over all interval graphs of a given size without induced $K_{1,5}$'s [32].

Theorem 8 follows directly from the next lemma.

▶ **Lemma 9.** ($\star$) *The graph G in Figure 4 is not a disjoint unit 2-interval graph.*

We conclude this section showing that Theorem 8 can actually be generalized for disjoint unit d-interval graphs for any $d > 2$.

▶ **Corollary 10.** ($\star$) *There exists a $K_{1,2d+1}$-free interval graph that is not a disjoint unit d-interval graph.*

[2] https://github.com/AbdallahS/unit-graphs

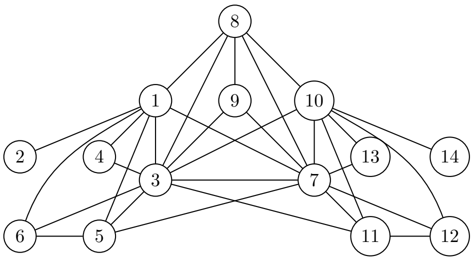


Figure 4 One of the 6 graphs with 14 vertices (the one with the fewest edges) which is an interval graph (see Figure 5) and $K_{1,5}$-free, but not disjoint unit 2-interval.

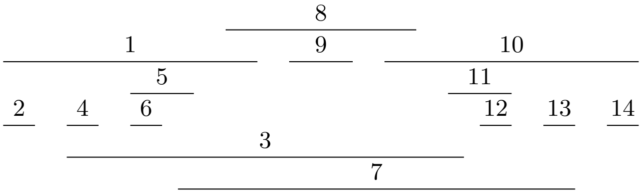


Figure 5 An interval representation of the graph in Figure 4.

5 Inclusions between the different subclasses of d-interval graphs

In this section, we analyze the relationships between different subclasses of multiple interval graphs. We have already seen that 2-interval graphs and disjoint 2-interval graphs are equivalent. Furthermore, the results from the previous two sections imply that the class of disjoint unit 2-interval graphs is properly contained in the class of unit 2-interval graphs. In the following, we summarize the containment relationships between unit 2-interval graphs, disjoint unit 2-interval graphs, balanced 2-interval graphs and disjoint balanced 2-interval graphs (see Figure 6 for a graphical illustration).

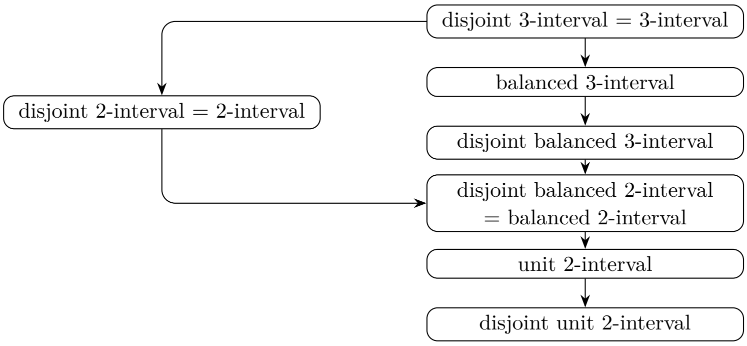

disjoint 3-interval = 3-interval
balanced 3-interval
disjoint 2-interval = 2-interval
disjoint balanced 3-interval
disjoint balanced 2-interval = balanced 2-interval
unit 2-interval
disjoint unit 2-interval

Figure 6 Landscape of graph subclasses of 3-interval graphs. An arrow from a graph class $\mathcal{C}$ to a $\mathcal{C}'$ indicates that $\mathcal{C}' \subset \mathcal{C}$. The relationships between the class of 2-interval graphs and the classes of balanced 3-interval graphs and disjoint balanced 3-interval graphs are not known.

▶ **Theorem 11.** ($\star$)

1. *The classes of* 2-interval *and* disjoint 2-interval *graphs are* equivalent*.*
2. *The classes of* balanced 2-interval *and* disjoint balanced 2-interval *graphs are* equivalent*.*
3. *The class of* unit 2-interval graphs *is* properly contained *in the class of* disjoint balanced 2-interval *graphs.*
4. *The class of* disjoint unit 2-interval *graphs is* properly contained *in the class of* unit 2-interval *graphs.*

We finish by showing that the previous theorem cannot be completely generalized for the subclasses of d-interval graphs, as the class of balanced d-interval graphs is not equivalent to the class of disjoint balanced d-interval graphs for $d > 2$. We first construct a graph that is balanced 3-interval but not disjoint balanced 3-interval and then show how to generalize this construction for every $d > 3$.

▶ **Theorem 12.** *The class of disjoint balanced 3-interval graphs is properly contained in the class of balanced 3-interval graphs.*

Proof. We construct a graph G which is balanced 3-interval but not disjoint balanced 3-interval. The high-level idea of the construction is that for a particular vertex, one of its intervals is forced to a given length, while the other two are forced to be placed somewhere where there is not enough space for both of them, and thus they cannot be disjoint (note that the difference with the case $d = 2$ is that now, if we stretch two of the intervals so that they do not intersect, we also have to modify the length of the third interval, and as we show here, this is not always possible). To enforce these constraints, we use the complete bipartite graph $K_{11,4}$ as a gadget and exploit the fact that any 3-interval representation of this gadget must be continuous (i.e., the union of the intervals in its underlying family is an interval) [31, Lemma 2] (see also [13, Fig. 3] for the idea of its representation).

We construct G as follows: we connect in a chain five $K_{11,4}$'s, to which we add six vertices v_1, v_2, v_3, v_4, v_5, v_6 (Figure 7 shows how to link v_1, v_2, v_3, v_4 to the chain, while vertices v_5 and v_6 mimic the behavior of v_3 and v_4 with a different set of neighbors, namely, v_5 is connected to the corresponding vertices of the first two $K_{11,4}$'s, and v_6 is connected to the corresponding vertices of the second and the third $K_{11,4}$'s). More precisely, let C_i, with $i \in \{1, \ldots, 5\}$, be the five $K_{11,4}$'s forming the chain, enumerated from left to right. Moreover, for every C_i, let f_i^j with $j \in \{1, \ldots, 11\}$ be the eleven vertices of one side of the bipartition, and t_i^k, with $k \in \{1, 2, 3, 4\}$, the four vertices of the other side of the bipartition. We assume that the chain is connected such that f_i^{11} is linked to f_{i+1}^1. Then, v_1 is connected to all the vertices of C_2 and C_4, and to f_3^{11} and f_5^1, plus another independent vertex. Similarly, v_2 is connected to all the vertices of C_2 and C_4, to v_1 and to f_1^{11} and f_3^1. On the other hand, v_3 is connected to f_3^{11}, t_3^4, t_4^1 and f_4^j for $j \in \{1, \ldots, 9\}$; while v_4 is connected to f_5^1, t_4^4, t_5^1 and f_4^j for $j \in \{3, \ldots, 11\}$. Finally, v_5 is connected to f_1^{11}, t_1^3, t_2^1 and f_2^j for $j \in \{1, \ldots, 7\}$, as well as f_4^8 and f_4^9, whereas v_6 is connected to f_3^1, t_2^4, t_3^1 and f_2^j for $j \in \{5, \ldots, 11\}$, as well as f_4^8 and f_4^9. The vertices v_1 and v_2 are both connected to v_3, v_4, v_5 and v_6.

Now, as any 3-interval representation of a $K_{11,4}$ is continuous, any realization of G groups the five $K_{11,4}$'s in a block [31]. For $j \in \{1, 2, 3\}$, let I_j be the intervals associated to v_1, J_j the intervals associated to v_2, and K_j the intervals associated to v_3. First, it is clear that we need three different intervals to cover the neighbors of v_1 (and these three intervals must be disjoint). Instead, the neighbors of v_2 could be covered only with two intervals. However, we will see that the two segments of the real line that need to be covered cannot have the same length (assuming that the 3-interval associated to v_1 is balanced). We will show that we need two intersecting intervals to cover the first segment.

Suppose that only two intervals are needed to represent the adjacencies of v_2, and let J be the interval displaying the edges between C_2 and v_2, and J_3 the interval displaying the edges between C_4 and v_2. Similarly, let I_1 be the interval associated to v_1 used to represent the edges with C_2, let I_2 the interval used to represent the edges with C_4, and I_3 the interval displaying the edge with the isolated vertex. One can easily see that J_3 is properly contained in I_2 (since I_2 must also intersect an interval associated to f_3^{11} on its left and an interval associated to f_5^1 on its right), while I_1 is properly contained in J (by an analogous argument). Thus, $len(J_3) < len(I_2) = len(I_1) < len(J)$. In order for the representation to be balanced,

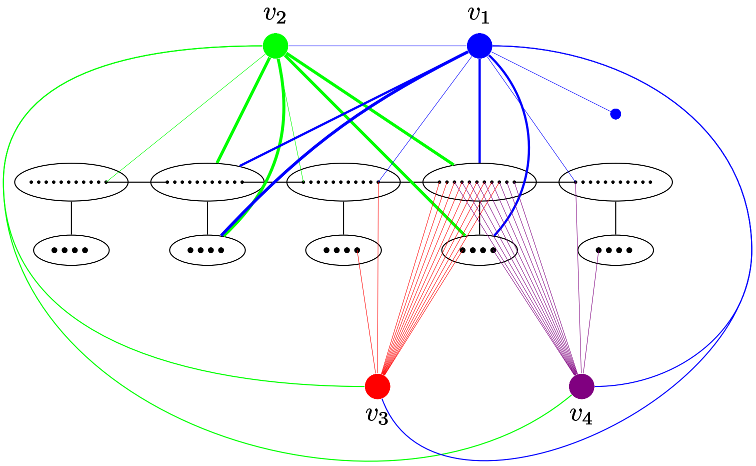


Figure 7 G is balanced 3-interval but not disjoint balanced 3-interval. $K_{11,4}$ graphs are drawn abstractly and are chained. A thick edge stopping at the border of the ellipse means that the vertex is connected to every vertex in the corresponding part of the $K_{11,4}$. Vertices v_5 and v_6 are omitted for readability purposes.

the segment of the real line covered by J needs to be covered by two different intervals, say J_1 and J_2. To prove that G is balanced 3-interval but not disjoint balanced 3-interval, we need to bound $len(J) - len(J_3)$. In particular, we need $len(J) - len(J_3) < len(J_3)$. Vertices v_3 and v_4 will allow us to find constants a and a' to bound $len(I_2) - len(J_3) \leqslant a + a'$, while vertices v_5 and v_6 will serve to find constants b and b' to bound $len(J) - len(I_1) \leqslant b + b'$. By showing that we can force the constants such that $a + a' + b + b' < len(J_3)$, we have the result. This will follow since we will have eight pairwise disjoint intervals properly contained in J_3: two of length a, two of length a', two of length b and two of length b'.

Indeed, let a and a' be the lengths of the intervals associated to v_3 and to v_4, respectively. The next claim implies that there are two disjoint intervals associated to v_3 properly contained in J_3, and another disjoint interval that properly contains the segment between $l(I_2)$ and $l(J_3)$, and so $l(J_3) - l(I_2) < a$.

▷ Claim 13. (⋆) Let G be a graph formed by the union of a $K_{11,4}$ and a vertex v which is adjacent to nine vertices in S_{11}, where S_{11} denotes the side of the bipartition with eleven vertices. Then, vertex v must be represented by three pairwise disjoint intervals, two of which are each properly contained in an interval representing a vertex of S_{11}.

Similarly, the segment between $r(I_2)$ and $r(J_3)$ is also contained in an interval associated to v_4, which has the same properties as v_3 and does not intersect any interval associated to v_3. This proves that there are two intervals of length a and two intervals of length a' (all pairwise disjoint) contained in J_3. Doing the same to bound $l(I_1) - l(J)$ and $r(J) - l(I_1)$, we get the result. Thus, to represent v_2, we need two intervals associated to v_2 to intersect. If we do not allow intersection, the length of these two intervals will be smaller than the length of the third interval associated to v_2, contradicting the fact that they are balanced. ◀

▶ **Corollary 14.** (⋆) *The class of disjoint balanced d-interval graphs is properly contained in the class of balanced d-interval graphs for every natural number* $d \geqslant 3$.

6 Concluding remarks

We have shown that the natural generalization of Roberts characterization for unit interval graphs remains valid for the most general definition of d-interval graphs that are interval graphs. However, quite surprisingly, if we require the d intervals to be disjoint, then the result does not hold anymore. It remains as an open question whether disjoint d-interval graphs that are also interval can be characterized in some other way, or simply if they can be recognized in polynomial time. Finally, we have obtained a relatively complete landscape of the containment relationships between different subclasses of 2-interval graphs, that cannot be fully generalized for $d > 2$. In particular, for $d > 2$, it is still unknown whether the class of unit d-interval graphs is contained in the class of disjoint balanced d-interval graphs.

References

1 Virginia Ardévol Martínez, Romeo Rizzi, Florian Sikora, and Stéphane Vialette. Recognizing unit multiple intervals is hard. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023*, volume 283 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.8.

2 Reuven Bar-Yehuda, Magnús M Halldórsson, Joseph Naor, Hadas Shachnai, and Irina Shapira. Scheduling split intervals. *SIAM J. Comput.*, 36(1):1–15, 2006.

3 Kenneth P. Bogart and Douglas B. West. A short proof that 'proper = unit'. *Discret. Math.*, 201(1-3):21–23, 1999. doi:10.1016/S0012-365X(98)00310-0.

4 Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-Tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

5 Ayelet Butman, Danny Hermelin, Moshe Lewenstein, and Dror Rawitz. Optimization problems in multiple-interval graphs. *ACM Trans. Algorithms*, 6(2):1–18, 2010.

6 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS structure and recognition of interval graphs. *SIAM J. Discret. Math.*, 23(4):1905–1953, 2009. doi:10.1137/S0895480100373455.

7 Maxime Crochemore, Danny Hermelin, Gad M. Landau, Dror Rawitz, and Stéphane Vialette. Approximating the 2-interval pattern problem. *Theor. Comput. Sci.*, 395(2-3):283–297, 2008. doi:10.1016/j.tcs.2008.01.007.

8 Guillermo Durán, Florencia Fernández Slezak, Luciano N. Grippo, Fabiano de S. Oliveira, and Jayme Luiz Szwarcfiter. On unit d–interval graphs. VII Latin American Workshop on Cliques in Graphs, 2016. URL: https://www.mate.unlp.edu.ar/~liliana/lawclique_2016/ffslezak.pdf,https://www.mate.unlp.edu.ar/~liliana/lawclique_2016/prolist.pdf.

9 Paul Erdös and Douglas B West. A note on the interval number of a graph. *Discret. Math.*, 55(2):129–133, 1985.

10 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

11 Peter C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets.* Wiley, 1985.

12 Mathew C. Francis, Daniel Gonçalves, and Pascal Ochem. The maximum clique problem in multiple interval graphs. *Algorithmica*, 71(4):812–836, 2015. doi:10.1007/s00453-013-9828-6.

13 Philippe Gambette and Stéphane Vialette. On restrictions of balanced 2-interval graphs. In Andreas Brandstädt, Dieter Kratsch, and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science, 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers*, volume 4769 of *LNCS*, pages 55–65. Springer, 2007. doi:10.1007/978-3-540-74839-7_6.

14 Frédéric Gardi. The Roberts characterization of proper and unit interval graphs. *Discret. Math.*, 307(22):2906–2908, 2007. doi:10.1016/j.disc.2006.04.043.

15 Jerrold R. Griggs and Douglas B. West. Extremal values of the interval number of a graph. *SIAM J. Algebraic Discret. Methods*, 1(1):1–7, 1980. doi:10.1137/0601001.

16 Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theor. Comput. Sci.*, 234(1-2):59–84, 2000. doi:10.1016/S0304-3975(97)00241-7.

17 Minghui Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theor. Comput. Sci.*, 411(49):4253–4262, 2010. doi:10.1016/j.tcs.2010.09.001.

18 Minghui Jiang. Recognizing d-interval graphs and d-track interval graphs. *Algorithmica*, 66(3):541–563, 2013. doi:10.1007/s00453-012-9651-5.

19 Minghui Jiang and Yong Zhang. Parameterized complexity in multiple-interval graphs: Domination, partition, separation, irredundancy. *Theor. Comput. Sci.*, 461:27–44, 2012. doi:10.1016/j.tcs.2012.01.025.

20 Deborah Joseph, Joao Meidanis, and Prasoon Tiwari. Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In *Scandinavian Workshop on Algorithm Theory*, pages 326–337. Springer, 1992.

21 C Lekkerkerker and Johan Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962.

22 Robert McGuigan. Presentation at NSF-CBMS Conference at Colby College, 1977.

23 Terry A McKee and Fred R McMorris. *Topics in intersection graph theory*. SIAM, 1999.

24 Fred S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, NY, 1969.

25 Fred S. Roberts. *Graph theory and its applications to problems of society*. SIAM, 1978.

26 Abdallah Saffidine. Unit 2-interval graph checker. Software, swhId: swh:1:dir:3d7b6495d1f70618a537cd23c94530c23c030215 (visited on 2024-08-06). URL: https://github.com/AbdallahS/unit-graphs.

27 Edward R Scheinerman and Douglas B West. The interval number of a planar graph: Three intervals suffice. *J. Comb. Theory, Ser. B*, 35(3):224–239, 1983.

28 Alexandre Simon. Algorithmic study of 2-interval graphs. Master's thesis, Delft University of Technology, 2021.

29 William T. Trotter and Frank Harary. On double and multiple interval graphs. *J. Graph Theory*, 3(3):205–211, 1979. doi:10.1002/jgt.3190030302.

30 Stéphane Vialette. On the computational complexity of 2-interval pattern matching problems. *Theor. Comput. Sci.*, 312(2-3):223–249, 2004. doi:10.1016/j.tcs.2003.08.010.

31 Douglas B. West and David B. Shmoys. Recognizing graphs with fixed interval number is NP-complete. *Discret. Appl. Math.*, 8(3):295–305, 1984. doi:10.1016/0166-218X(84)90127-6.

32 Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theor. Comput. Sci.*, 806:310–322, 2020. doi:10.1016/J.TCS.2019.04.017.

A Direct Translation from LTL with Past to Deterministic Rabin Automata

Shaun Azzopardi
University of Malta, Msida, Malta

David Lidell
University of Gothenburg, Sweden

Nir Piterman
University of Gothenburg, Sweden

Abstract

We present a translation from linear temporal logic *with past* to deterministic Rabin automata. The translation is direct in the sense that it does not rely on intermediate non-deterministic automata, and asymptotically optimal, resulting in Rabin automata of doubly exponential size. It is based on two main notions. One is that it is possible to encode the history contained in the prefix of a word, as relevant for the formula under consideration, by performing simple rewrites of the formula itself. As a consequence, a formula involving past operators can (through such rewrites, which involve alternating between weak and strong versions of past operators in the formula's syntax tree) be correctly evaluated at an arbitrary point in the future without requiring backtracking through the word. The other is that this allows us to generalize to linear temporal logic with past the result that the language of a pure-future formula can be decomposed into a Boolean combination of simpler languages, for which deterministic automata with simple acceptance conditions are easily constructed.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Modal and temporal logics

Keywords and phrases Linear temporal logic, ω-automata, determinization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.13

Related Version *Full Version*: https://arxiv.org/abs/2405.01178 [1]

Funding This research is supported by the Swedish research council (VR) project (No. 2020-04963) and the ERC Consolidator grant DSynMA (No. 772459).

Acknowledgements We are grateful to Gerardo Schneider and the reviewers for their very useful feedback.

1 Introduction

Finite-state automata over infinite words, commonly referred to as ω-automata, have been studied as models of computation since their introduction in the 1960s. Their introduction was followed by extensive investigations into their expressive power, closure properties, and the decidability and complexity of related decision problems, such as non-emptiness and language containment. In particular, it was soon established that determinization constructions of such automata are considerably more complex than they are for their finite word counterparts, for which a simple subset construction is sufficient.

In the 1970s, Pnueli [15] proposed linear temporal logic (*LTL*) as a specification language for the analysis and verification of programs, based on the idea that a program can be viewed in the context of a stream of interactions between it and its environment. It has since become ubiquitous in both academia and the industry due to the perceived balance of its expressive power and the computational complexity of its related decision procedures. The intimate semantic connection between *LTL* and ω-automata further motivated research into the properties of both, also from a practical point of view. In addition to being of theoretical interest, the fact that multiple automata-based applications of *LTL* as a specification language – such as reactive synthesis and probabilistic model checking – require deterministic automata, has drawn additional attention to the study of efficient translation procedures from *LTL* to deterministic ω-automata.

In the classic approach to determinization, the given *LTL* formula is first translated to a non-deterministic Büchi automaton. This automaton is subsequently determinized, for example using Safra's procedure [16], or the more modern Safra/Piterman variant [13]. Such constructions are both conceptually complex and difficult to implement. Moreover, information about the structure of the given formula is lost in the initial translation to Büchi automata; in particular, because of the generality of such determinization procedures, they cannot take advantage of the fact that *LTL* is less expressive than ω-automata.

In 2020, Esparza et al. [5] presented a novel translation from *LTL* to various automata, that is asymptotically optimal in both the deterministic and non-deterministic cases. The translation is direct in the sense that it avoids the intermediate steps of the classic approach, which involve employing a variety of separate translation procedures. In particular, for deterministic automata, it forgoes Safra-based constructions. Instead, the language of the formula under consideration is decomposed into a Boolean combination of simpler languages, for which deterministic automata with simple acceptance conditions can easily be constructed using what the authors have dubbed the "after-function"; that such a decomposition exists is a fundamental result named the "Master Theorem". These simpler automata are then combined into the desired final automaton using basic product or union operations according to the structure of the decomposition.

In this paper, we consider past linear temporal logic (*pLTL*); the extension of *LTL* that includes the past operators "Yesterday" and "Since", analogous to the standard operators "Next" and "Until", respectively. We adapt the Master Theorem and generalize the derived *LTL*-to-deterministic-Rabin-automata translation to *pLTL*, while maintaining its optimal asymptotic complexity. The merits of *LTL* extended with past operators were argued by Lichtenstein et al. [9], who also showed that their addition does not result in a more expressive logic with respect to initial equivalence of formulae. At the same time, the complexity of satisfiability/validity- and model checking remains PSPACE-complete for *pLTL* [17]. When it comes to determinization, as Esparza's approach [5] applies only to (future) *LTL*, the only option is the two-step approach: translate *pLTL* to nondeterministic Büchi automata [14] and convert to deterministic parity/Rabin automata [16, 13]. Our generalization to *pLTL* is of both theoretical and practical interest, for two main reasons. First, certain properties are more naturally and elegantly expressed with the help of past operators. Secondly, there exist formulae in *pLTL* such that all (initially) equivalent *LTL* formulae are exponentially larger [11]. Both of these properties can be exemplified by considering the natural language specification *"At any point in time, p should occur if and only if q and r have occurred at least once in the past"*. Expressing this in *LTL* requires explicitly describing the possible desired orders of occurrences of p, q, and r:

$$\big((\neg p \wedge \neg q)\,\mathbf{W}\,(r \wedge ((\neg p \wedge \neg q)\,\mathbf{W}\,(p \wedge q))) \vee (\neg p \wedge \neg r)\,\mathbf{W}\,(q \wedge ((\neg p \wedge \neg r)\,\mathbf{W}\,(p \wedge r)))\big) \\ \wedge \mathbf{G}(p \Rightarrow \mathbf{XG}p).$$

The same specification is very intuitively and succinctly expressed in *pLTL* by the formula $\mathbf{G}(p \Leftrightarrow \mathbf{O}q \wedge \mathbf{O}r)$.

The main contributions of this paper are an adaptation of the Master Theorem for *pLTL* and a utilization of this in the form of an asymptotically optimal direct translation from *pLTL* to deterministic Rabin automata. The paper is structured in the following manner: In Section 2, we define the syntax and semantics of linear temporal logic with past, infinite words and ω-automata, and the notion of propositional equivalence. As the automata we aim to construct are one-way, we need a way to encode information about the input history directly into the formula being translated; the machinery required to accomplish this is introduced in Section 3. The foundation of the translation from *pLTL* to Rabin automata is the after-function of Section 4. The subsequent Sections 5 and 6 are adaptations of the corresponding sections in Esparza et al. [5]. The decomposition of the language of the formula to be translated into a Boolean combination of simpler languages requires considering the limit-behavior of the formula. This notion is made precise in Section 5, which finishes with a presentation of the Master Theorem for *pLTL*. Section 6 describes how to create deterministic automata from the simpler languages of the decomposition, and how to combine them into a deterministic Rabin automaton. Finally, we conclude with a brief discussion in Section 7.

2 Preliminaries

2.1 Infinite Words and ω-automata

An infinite word w over a non-empty finite alphabet Σ is an infinite sequence $\sigma_0, \sigma_1, \ldots$ of letters from Σ. Given an infinite word w, we denote the finite infix $\sigma_t, \sigma_{t+1}, \ldots, \sigma_{t+s-1}$ of w by w_{ts}. If $t = s$, then w_{ts} is defined as representing the empty word ϵ. Note that no ambiguity will arise as we use parentheses whenever required; for example, $w_{(st)(en)}$ rather than w_{sten} We denote the infinite suffix $\sigma_t, \sigma_{t+1}, \ldots$ of w by w_t. We will also consider finite words, using the same infix- and suffix notation.

An *ω-automaton* over an alphabet Σ is a quadruple (Q, Q_0, δ, α), where Q is a finite set of states, $Q_0 \subseteq Q$ a non-empty set of initial states, $\delta \in Q \times \Sigma \to 2^Q$ a (partial) transition function, and α a set constituting its acceptance condition. In the case where $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and all $\sigma \in \Sigma$, the automaton is called *deterministic*. For deterministic automata we write $\delta : Q \times \Sigma \to Q$. Given an ω-automaton $\mathcal{A} = (Q, Q_0, \delta, \alpha)$ and an infinite word $w = \sigma_0, \sigma_1, \ldots$, both over the same alphabet, a *run* of $\mathcal{A}$ on w is a sequence of states $r = r_0, r_1, \ldots$ of Q such that $r_0 \in Q_0$ and $r_{i+1} \in \delta(r_i, \sigma_i)$ for all $i \geq 0$. Given such a run r, we write $\mathit{Inf}(r)$ to denote the set of states appearing infinitely often in r.

In this paper, we consider three particular classes of ω-automata, which differ only by their acceptance condition α. *Büchi-* and *co-Büchi* automata are ω-automata with α as a set $Q' \subseteq Q$. A Büchi automaton *accepts* the infinite word w iff there exists a run r on w such that $\mathit{Inf}(r) \cap Q' \neq \varnothing$, while a co-Büchi automaton accepts w iff there exists a run r on w such that $\mathit{Inf}(r) \cap Q' = \varnothing$. We will also consider Rabin automata. A Rabin automaton has a set of subsets $R \subseteq 2^Q \times 2^Q$ as acceptance condition, and accepts w iff there exists a run r on w and a pair $(A, B) \in R$ such that $\mathit{Inf}(r) \cap A = \varnothing$ and $\mathit{Inf}(r) \cap B \neq \varnothing$. We write *DBA*, *DCA*, and *DRA* to refer to deterministic Büchi-, co-Büchi-, and Rabin automata, respectively.

We use a specialized form of cascade composition of two automata: bed automaton and runner automaton. Bed automata are automata without an acceptance condition. Runner automata read input letters and (next) states of bed automata: given a bed automaton $\mathcal{A}$ with S as set of states, a runner automaton is $\mathcal{B} = (S, Q, Q_0, \delta, \alpha)$, where Q, Q_0, and α are as before and $\delta : Q \times S \times \Sigma \to 2^Q$ is the transition function. The composition $\mathcal{A} \ltimes \mathcal{B}$ is the automaton $(Q \times S, Q_0 \times S_0, \overline{\delta}, \overline{\alpha})$, where $\overline{\delta}(q, s, \sigma) = \{(q', s') \mid q' \in \delta_B(q, s', \sigma), s' \in \delta_{\mathcal{A}}(s, \sigma)\}$ and either

$\overline{\alpha} = \alpha \times S$ if $\mathcal{B}$ is a Büchi- or co-Büchi automaton, or $\overline{\alpha} = \bigcup\{(A \times S) \times (B \times S) \mid (A, B) \in \alpha\}$ if $\mathcal{B}$ is a Rabin automaton. Note that $\mathcal{A} \ltimes \mathcal{B}$ is an ω-automaton with $\mathcal{B}$'s acceptance, e.g., if $\mathcal{A}$ is deterministic and $\mathcal{B}$ is a deterministic co-Büchi runner automaton, then $\mathcal{A} \ltimes \mathcal{B}$ is a DCA.

2.2 Linear Temporal Logic with Past

Given a non-empty finite set of propositional variables AP, the well-formed formulae φ of *pLTL* are generated by the following grammar:

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \varphi \mid \mathbf{Y}\varphi \mid \varphi \mathbf{S} \varphi,$$

where $p \in AP$. Given a formula φ, we write $\mathit{Var}(\varphi)$ to denote the set of all atomic propositions appearing in φ. Given a formula φ, natural number t, and infinite word $w = \sigma_0, \sigma_1, \ldots$ over $2^{\mathit{Var}(\varphi)}$, we write $(w, t) \models \varphi$ to denote that w satisfies φ at index t. The meaning of this is made precise by the following inductive definition:

$$\begin{array}{ll}
(w,t) \models \top & (w,t) \not\models \bot \\
(w,t) \models p \text{ iff } p \in \sigma_t & (w,t) \models \neg\varphi \text{ iff } (w,t) \not\models \varphi \\
(w,t) \models \varphi \wedge \psi \text{ iff } (w,t) \models \varphi \text{ and } (w,t) \models \psi & (w,t) \models \varphi \vee \psi \text{ iff } (w,t) \models \varphi \text{ or } (w,t) \models \psi \\
(w,t) \models \mathbf{X}\varphi \text{ iff } (w,t+1) \models \varphi & (w,t) \models \mathbf{Y}\varphi \text{ iff } t > 0 \text{ and } (w,t-1) \models \varphi
\end{array}$$

$$(w,t) \models \varphi \mathbf{U} \psi \text{ iff } \exists r \geq t \,.\, ((w,r) \models \psi \text{ and } \forall s \in [t,r) \,.\, (w,s) \models \varphi)$$

$$(w,t) \models \varphi \mathbf{S} \psi \text{ iff } \exists r \leq t \,.\, ((w,r) \models \psi \text{ and } \forall s \in (r,t] \,.\, (w,s) \models \varphi) \,.$$

When $t = 0$ we omit the index and simply write $w \models \varphi$. Observe that $\mathbf{Y}$ is almost exactly the past analog of $\mathbf{X}$, with a similar relationship between $\mathbf{S}$ and $\mathbf{U}$. They differ in that the past is bounded; in particular, a formula $\mathbf{Y}\varphi$ is never satisfied at $t = 0$. The *language* of a formula φ, denoted $\mathcal{L}(\varphi)$, is the set of all infinite words w such that $w \models \varphi$. Two *pLTL* formulae φ and ψ are semantically equivalent, denoted $\varphi \equiv \psi$, iff $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$.

In addition to the above, we will also consider the following derived operators:

$$\begin{array}{ccc}
\mathbf{F}\varphi \coloneqq \top \mathbf{U} \varphi & \mathbf{O}\varphi \coloneqq \top \mathbf{S} \varphi & \mathbf{G}\varphi \coloneqq \neg\mathbf{F}\neg\varphi \\
\mathbf{H}\varphi \coloneqq \neg\mathbf{O}\neg\varphi & \varphi \mathbf{W} \psi \coloneqq \varphi \mathbf{U} \psi \vee \mathbf{G}\varphi & \varphi \tilde{\mathbf{S}} \psi \coloneqq \varphi \mathbf{S} \psi \vee \mathbf{H}\varphi \\
\varphi \mathbf{M} \psi \coloneqq \psi \mathbf{U} (\varphi \wedge \psi) & \varphi \mathbf{B} \psi \coloneqq \psi \mathbf{S} (\varphi \wedge \psi) & \varphi \mathbf{R} \psi \coloneqq \psi \mathbf{W} (\varphi \wedge \psi) \\
\varphi \tilde{\mathbf{B}} \psi \coloneqq \psi \tilde{\mathbf{S}} (\varphi \wedge \psi) & \tilde{\mathbf{Y}}\varphi \coloneqq \mathbf{Y}\varphi \vee \neg\mathbf{Y}\top. &
\end{array}$$

The past operators above are defined in analogy with their standard future counterparts. An important exception is the *weak yesterday* operator $\tilde{\mathbf{Y}}$, which is similar to $\mathbf{Y}$. However, a formula $\tilde{\mathbf{Y}}\varphi$ is always satisfied at $t = 0$.

A *pLTL* formula that is neither atomic nor whose syntax tree is rooted with a Boolean operator is called a *temporal* formula, and a *pLTL* formula whose syntax tree is rooted with an element of $\{\mathbf{Y}, \tilde{\mathbf{Y}}, \mathbf{S}, \tilde{\mathbf{S}}, \mathbf{B}, \tilde{\mathbf{B}}\}$ a *past* formula. Finally, a *pLTL* formula that is a proposition or the negation thereof is *propositional.* We write $\mathit{sf}(\varphi)$ to denote the set of propositional and temporal subformulae of φ, and $\mathit{psf}(\varphi)$ the set of past subformulae of φ. The *size* of a *pLTL* formula φ, denoted $|\varphi|$, is defined as the number of nodes of its syntax tree that are either temporal or propositional.

A *pLTL* formula where negations only appear before atomic propositions is in *negation normal form.* Observe that with the derived operators above, an arbitrary *pLTL* formula can be rewritten in negation normal form with a linear increase in size. For the remainder of the paper, when we write “formula” we implicitly refer to *pLTL* formulae in negation normal form, with no occurrences of $\mathbf{F}, \mathbf{G}, \mathbf{O}$ or $\mathbf{H}$. Subformulae rooted with either of these four operators can be replaced by equivalent formulae of the same size: every subformula of the form $\mathbf{F}\psi$ can be replaced with $\top \mathbf{U} \psi$ and every subformula of the form $\mathbf{G}\psi$ with $\psi \mathbf{W} \bot$, and analogously for $\mathbf{O}\psi$ and $\mathbf{H}\psi$. While these four derived operators are not part of the syntax under consideration – for the purpose of keeping the presentation more concise – we will occasionally use them as convenient shorthand. When we write "word" we implicitly refer to infinite words, unless otherwise stated.

We conclude this section by defining the notion of *propositional equivalence*, which is a stronger notion of equivalence than that given by the semantics of *pLTL*. It is relatively simple to determine whether two formulae are propositionally equivalent, which makes the notion useful in defining the state spaces of automata as equivalence classes of formulae. We also state a lemma that allows us to lift functions defined on formulae to the propositional equivalence classes they belong to. Both are due to Esparza et al. [5].

▶ **Definition 1** (Propositional Semantics of *pLTL*). *Let $\mathcal{I}$ be a set of formulae and φ a formula. The propositional satisfaction relation $\mathcal{I} \models_p \varphi$ is inductively defined as*

$$\begin{array}{ll} \mathcal{I} \models_p \top & \mathcal{I} \models_p \psi \wedge \xi \text{ iff } \mathcal{I} \models_p \psi \text{ and } \mathcal{I} \models_p \xi \\ \mathcal{I} \not\models_p \bot & \mathcal{I} \models_p \psi \vee \xi \text{ iff } \mathcal{I} \models_p \psi \text{ or } \mathcal{I} \models_p \xi, \end{array}$$

with $\mathcal{I} \models_p \varphi$ iff $\varphi \in \mathcal{I}$ for all other cases. Two formulae φ and ψ are propositionally equivalent, denoted $\varphi \sim \psi$, if $\mathcal{I} \models_p \varphi \Leftrightarrow \mathcal{I} \models_p \psi$ for all sets of formulae $\mathcal{I}$. The (propositional) equivalence class of a formula φ is denoted $[\varphi]_\sim$. The (propositional) quotient set of a set of formulae Ψ is denoted $\Psi_{/\sim}$.

▶ **Lemma 2.** *Let f be a function on formulae such that $f(\top) = \top$, $f(\bot) = \bot$, and for all formulae φ and ψ, $f(\varphi \wedge \psi) = f(\varphi) \wedge f(\psi)$ and $f(\varphi \vee \psi) = f(\varphi) \vee f(\psi)$. Then, for all pairs of formulae φ and ψ, if $\varphi \sim \psi$ then $f(\varphi) \sim f(\psi)$.*

3 Encoding the Past

Informally, given a formula φ and word w, our aim is to define a function that consumes a given finite prefix of w, of arbitrary length t, and produces a new formula φ', such that the suffix w_t satisfies φ' iff w satisfies φ. This function will serve as the foundation for defining the state spaces and transition relations of the automata that we are to construct. For standard *LTL*, defining such a function is straightforward using the local semantics of *LTL*. With the introduction of past operators the situation becomes more complicated. As a prefix of w is consumed we lose the information about the past therein, and must instead encode this information in the rewritten formula. The key insight is that this can be accomplished by rewriting strong past operators of φ – the operators $\mathbf{Y}$, $\mathbf{S}$, and $\mathbf{B}$ – into their weak counterparts $\tilde{\mathbf{Y}}$, $\tilde{\mathbf{S}}$, and $\tilde{\mathbf{B}}$, respectively, and vice versa, based on the consumed input. This section makes this idea precise.

▶ **Definition 3** (Weakening and strengthening formulae). *The weakening $\varphi_\mathcal{W}$ and strengthening $\varphi_\mathcal{S}$ of a formula φ is defined by case distinction on φ as*

$$\begin{array}{lll} (\mathbf{Y}\psi)_\mathcal{W} := \tilde{\mathbf{Y}}\psi & (\psi \,\mathbf{S}\, \xi)_\mathcal{W} := \psi \,\tilde{\mathbf{S}}\, \xi & (\psi \,\mathbf{B}\, \xi)_\mathcal{W} := \psi \,\tilde{\mathbf{B}}\, \xi \\ (\tilde{\mathbf{Y}}\psi)_\mathcal{S} := \mathbf{Y}\psi & (\psi \,\tilde{\mathbf{S}}\, \xi)_\mathcal{S} := \psi \,\mathbf{S}\, \xi & (\psi \,\tilde{\mathbf{B}}\, \xi)_\mathcal{S} := \psi \,\mathbf{B}\, \xi. \end{array}$$

For all other cases we have $\varphi_\mathcal{W} := \varphi$ and $\varphi_\mathcal{S} := \varphi$.

▶ **Definition 4** (Rewriting past operators under sets). *Given a formula φ and set of past formulae C, we write $\varphi\langle C\rangle$ to denote the result of weakening or strengthening the past operators in the syntax tree of φ according to C while otherwise maintaining its structure, as per the following inductive definition:*

$$\begin{aligned} a\langle C\rangle &\coloneqq a && (a \text{ atomic})\\ (op\,\psi)\langle C\rangle &\coloneqq \begin{cases}(op\,\psi\langle C\rangle)_{\mathcal{W}} & (op\,\psi \in C)\\ (op\,\psi\langle C\rangle)_{\mathcal{S}} & (\text{otherwise})\end{cases} && (op \text{ unary})\\ (\psi\ op\,\xi)\langle C\rangle &\coloneqq \begin{cases}(\psi\langle C\rangle\ op\,\xi\langle C\rangle)_{\mathcal{W}} & (\psi\ op\,\xi \in C)\\ (\psi\langle C\rangle\ op\,\xi\langle C\rangle)_{\mathcal{S}} & (\text{otherwise}).\end{cases} && (op \text{ binary}).\end{aligned}$$

We overload this definition to sets: given a set of formulae S, we define $S\langle C\rangle \coloneqq \{s\langle C\rangle \mid s \in S\}$.

▶ **Example 5.** Consider the formula $\varphi = \mathbf{Y}(p\,\tilde{\mathbf{S}}\,q)$ and set $C = \{\varphi\}$. We then have $\varphi\langle C\rangle = \tilde{\mathbf{Y}}(p\,\mathbf{S}\,q)$. For the same formula φ and set $C = \{p\,\tilde{\mathbf{S}}\,q)$, we instead have $\varphi\langle C\rangle = \varphi$.

▶ **Definition 6** (Weakening conditions). *Given a past formula φ, we define the weakening condition function $wc(\varphi)$:*

$$\begin{aligned} &wc(\mathbf{Y}\psi) \coloneqq \psi \quad && wc(\psi\,\mathbf{S}\,\xi) \coloneqq \xi && wc(\psi\,\mathbf{B}\,\xi) \coloneqq \psi \wedge \xi\\ &wc(\tilde{\mathbf{Y}}\psi) \coloneqq \psi && wc(\psi\,\tilde{\mathbf{S}}\,\xi) \coloneqq \psi \vee \xi \quad && wc(\psi\,\tilde{\mathbf{B}}\,\xi) \coloneqq \xi.\end{aligned}$$

The weakening condition for a past formula serves as a requirement that must hold immediately in order to justify weakening the formula for the next time step. More precisely, if $w \models wc(\varphi)$ then it is enough to check that $w_1 \models \varphi_{\mathcal{W}}$ to conclude that $(w, 1) \models \varphi$.

▶ **Example 7.** Let $\varphi = \mathbf{X}(p\,\mathbf{S}\,q)$ and w be a word such that $w_{02} = \{q\}\{p\}$. By establishing that $w \models wc(p\,\mathbf{S}\,q) = q$, we can "forget" about the initial letter $\{q\}$; it is enough to check that $w_1 \models p\,\tilde{\mathbf{S}}\,q$ to conclude that $(w, 1) \models p\,\mathbf{S}\,q$, and hence that $w \models \varphi$.

This suggests that there exists a set of past subformulae of φ that precisely captures the information contained in the initial letter of w required to evaluate all past subformulae of φ at every point in time. This is the main result of this section, which we now summarize.

▶ **Definition 8** (Sets of entailed subformulae). *Let φ be a formula, w a word, and $t \in \mathbb{N}$. The set of past subformulae of φ entailed by w at t is inductively defined as*

$$C^w_{\varphi,0} \coloneqq \{\psi \in psf(\varphi) \mid \psi = \psi_{\mathcal{W}}\} \qquad C^w_{\varphi,t} \coloneqq \{\psi \in psf(\varphi\langle C^w_{\varphi,t-1}\rangle) \mid w_t \models wc(\psi)\} \quad (t > 0).$$

When the word w above is clear from the context, we simply write $C_{\varphi,t}$. Given $t \in \mathbb{N}$ we denote the sequence $C_{\varphi,0}, \ldots, C_{\varphi,t}$ of length $t+1$ by $\vec{C}_{\varphi,t}$. Given a sequence $\vec{C} = C_0, C_1, \ldots, C_t$ of sets of past formulae, there exists a set $C \subseteq psf(\varphi)$ that has the same effect in a rewrite as the sequential application of rewrites of $\vec{C}$, i.e., such that $\varphi\langle C\rangle = \varphi\langle C_0\rangle\langle C_1\rangle \ldots \langle C_t\rangle$. This is the set $\{\psi \in psf(\varphi) \mid \psi\langle C_0\rangle\langle C_1\rangle \ldots \langle C_t\rangle = (\psi\langle C_0\rangle\langle C_1\rangle \ldots \langle C_t\rangle)_{\mathcal{W}}\}$, which we denote by $\circ\,\vec{C}$. In particular, there exists a set that captures the sequence of sets of entailed subformulae, denoted $\circ\,\vec{C}_{\varphi,t}$.

▶ **Lemma 9.** *Given a formula φ, word w, and $t \in \mathbb{N}$, we have $(w, t) \models \varphi$ iff $w_t \models \varphi\langle\circ\,\vec{C}_{\varphi,t}\rangle$.*

With these definitions in place, we are ready to define the after-function for *pLTL*.

4 The after-function for *pLTL*

The after-function is the foundation for defining the states and transition relations of all automata used in our *pLTL*-to-DRA translation. We begin by defining the *local* after-function:

▶ **Definition 10** (The local after-function). *Given a formula φ, a letter $\sigma \in 2^{Var(\varphi)}$, and a set of past formulae C, we inductively define the local after-function af_ℓ mutually with the local past update-function pu_ℓ as follows:*

$$\begin{aligned}
af_\ell(\top, \sigma, C) &:= \top \\
af_\ell(\bot, \sigma, C) &:= \bot \\
af_\ell(p, \sigma, C) &:= \mathit{if}\ (p \in \sigma)\ \mathit{then}\ \top\ \mathit{else}\ \bot \\
af_\ell(\neg p, \sigma, C) &:= \mathit{if}\ (p \in \sigma)\ \mathit{then}\ \bot\ \mathit{else}\ \top \\
af_\ell(\mathbf{X}\psi, \sigma, C) &:= pu_\ell(\psi, \sigma, C) \\
af_\ell(\mathbf{Y}\psi, \sigma, C) &:= \bot \\
af_\ell(\tilde{\mathbf{Y}}\psi, \sigma, C) &:= \top \\
af_\ell(\psi\ op\ \xi, \sigma, C) &:= af_\ell(\psi, \sigma, C)\ op\ af_\ell(\xi, \sigma, C) && (op \in \{\wedge, \vee\}) \\
af_\ell(\psi\ op\ \xi, \sigma, C) &:= af_\ell(\xi, \sigma, C) \vee af_\ell(\psi, \sigma, C) \wedge pu_\ell(\psi\ op\ \xi, \sigma, C) && (op \in \{\mathbf{U}, \mathbf{W}\}) \\
af_\ell(\psi\ op\ \xi, \sigma, C) &:= af_\ell(\xi, \sigma, C) \wedge (af_\ell(\psi, \sigma, C) \vee pu_\ell(\psi\ op\ \xi, \sigma, C)) && (op \in \{\mathbf{R}, \mathbf{M}\}) \\
af_\ell(\psi\ op\ \xi, \sigma, C) &:= af_\ell(wc(\psi\ op\ \xi), \sigma, C)) && (op \in \{\mathbf{S}, \tilde{\mathbf{S}}, \mathbf{B}, \tilde{\mathbf{B}}\}),
\end{aligned}$$

where

$$pu_\ell(\varphi, \sigma, C) := \varphi\langle C\rangle \wedge \bigwedge_{\psi \in psf(\varphi) \cap C} af_\ell(wc(\psi), \sigma, C).$$

Intuitively, in the context of reading the initial letter σ of the word w, the local after-function decomposes φ into parts that can be fully evaluated using σ and immediately be replaced with $\top$ or $\bot$, and parts that can only be partially evaluated using σ. The resulting formula is then left to be further evaluated in the future; in the automaton, it corresponds to the state reached upon reading σ from the state corresponding to φ. Crucially, the past subformulae of the partially evaluated part are updated by pu_ℓ, using the information in C. Here, C is to be thought of as a guess of the past subformulae of φ whose weakening conditions hold upon reading σ. Finally, the weakening conditions that must hold to justify the guess are added conjunctively. Observe that, as these weakening conditions may contain subformulae referring to the future, it may not be possible to fully evaluate them immediately; this motivates the recursive application of af_ℓ in pu_ℓ.

▶ **Definition 11** (The extended local after-function). *Given a (possibly empty) finite word w of length n and sequence of sets of past formulae $\vec{C}$ of length $n+1$, we extend af_ℓ to w and $\vec{C}$ as follows:*

$$\begin{aligned}
af_\ell(\varphi, \epsilon, \vec{C}_{01}) &:= \varphi && (n = 0) \\
af_\ell(\varphi, w_{0n}, \vec{C}_{0(n+1)}) &:= af_\ell\left(\varphi_{n-1}, w_{(n-1)n}, \vec{C}_{n(n+1)}\right) && (n > 0),
\end{aligned}$$

where $\varphi_n = af_\ell(\varphi, w_{0n}, \vec{C}_{0(n+1)})$.

Observe that the initial set of $\vec{C}$ in the above definition is discarded; this is to match the sequence of Definition 8. For formulae where no future operators are nested inside past operators, the set of entailed subformulae is completely determined by the prefix w_{0n}. This is not the case in general, however, and so the (global) after-function is defined to consider all possible subsets of past subformulae of φ as a disjunction.

▶ **Definition 12** (The after-function). *Let φ be a formula and σ a letter. The after-function af is defined as:*

$$af(\varphi,\sigma) := \bigvee_{C \in 2^{psf(\varphi)}} af_\ell(\varphi,\sigma,C).$$

The extension of af to finite words is done in the natural way: given a formula φ and word w of length n, we define

$$\begin{aligned} af(\varphi,\epsilon) &:= \varphi && (n=0)\\ af(\varphi,w_{0n}) &:= af(af(\varphi,w_{0(n-1)}),w_{(n-1)n}) && (n>0). \end{aligned}$$

▶ **Example 13.** Let $\varphi = \mathbf{X}(p\,\mathbf{S}\,\mathbf{X}q)$. Observe that $\varphi \equiv \mathbf{X}(p \wedge q \vee \mathbf{X}q)$. Upon reading a letter σ we can guess that the "since" started holding at the current point, corresponding to the set $C = \{p\,\mathbf{S}\,\mathbf{X}q\}$ and formula $p\,\tilde{\mathbf{S}}\,\mathbf{X}q \wedge q$. Alternatively, we may guess that the "since" did *not* start holding upon reading σ, corresponding to the set $C = \varnothing$ and formula $p\,\mathbf{S}\,\mathbf{X}q$. Hence $af(\varphi,\sigma) = p\,\tilde{\mathbf{S}}\,\mathbf{X}q \wedge q \vee p\,\mathbf{S}\,\mathbf{X}q$. This is equivalent (at $t = 0$) to $p \wedge q \vee \mathbf{X}q$, which is what must be satisfied by w_1, as desired.

The correctness of af is established by the following theorem:

▶ **Theorem 14.** *For every formula φ, word w, and $t \in \mathbb{N}$ we have $w \models \varphi$ iff $w_t \models af(\varphi, w_{0t})$.*

5 Stability and the Master Theorem

We consider two fragments of *pLTL*: *μ-pLTL*, the set of formulae whose future operators are members of $\{\mathbf{X},\mathbf{U},\mathbf{M}\}$, and *$\nu$-pLTL*, the set of formulae whose future operators are members of $\{\mathbf{X},\mathbf{W},\mathbf{R}\}$. Given a formula φ, we define the set $\mu(\varphi)$ of subformulae of φ whose syntax trees are rooted with $\mathbf{U}$ or $\mathbf{M}$. Similarly, we define the set $\nu(\varphi)$ of subformulae of φ whose syntax trees are rooted with $\mathbf{W}$ or $\mathbf{R}$.

The Master Theorem for *pLTL* establishes that the language of a *pLTL* formula can be decomposed into a Boolean combination of simple languages. It is motivated by two ideas:

i) Assume that φ is a formula and w is a word such that all subformulae in $\mu(\varphi)$ that are eventually satisfied by w are infinitely often satisfied by w, and all subformulae in $\nu(\varphi)$ that are almost always satisfied by w never fail to be satisfied by w. In this case, we say that w is a *stable* word of φ, as will be properly defined shortly. Under these circumstances, a subformula of φ of the form $\psi\,\mathbf{U}\,\xi$ is satisfied by w iff both $\psi\,\mathbf{W}\,\xi$ and $\mathbf{GF}(\psi\,\mathbf{U}\,\xi)$ are. Dually, a subformula of φ of the form $\psi\,\mathbf{W}\,\xi$ is satisfied by w iff either $\psi\,\mathbf{U}\,\xi$ or $\mathbf{FG}(\psi\,\mathbf{W}\,\xi)$ are. Hence, we can partition all words over $\mathrm{Var}(\varphi)$ into partitions of the form $P_{M,N}$, where $w \in P_{M,N}$ iff M is the set of *μ-pLTL*-subformulae of φ satisfied infinitely often by w, and N the set *ν-pLTL*-subformulae of φ that are almost always satisfied by w. Given two such sets M and N, the above implies that φ can be rewritten into a formula that belongs to either the fragment *μ-pLTL* or *ν-pLTL*, as desired.

ii) Given a formula φ and word w, there exists a point in the future from which the above holds. Indeed, if we look far ahead into the future, all subformulae of φ that are satisfied by w only finitely often will have been satisfied for the last time. In particular, this is true for the μ-*pLTL*-subformulae of φ. Similarly, there exists a point in the future at which all subformulae of φ that are almost always satisfied by w will have failed to be satisfied by w for the last time. In particular, this is true for the ν-*pLTL*-subformulae of φ.

These two notions suggest that, given a formula φ and word $w \in P_{M,N}$, it is possible to transform φ using the after-function until w becomes stable, and then rewrite it according to either M or N. Since these sets are unknown, we need to consider all possible combinations of such subsets, which ultimately manifests as a number of Rabin pairs exponential in the size of the formula. For more details and further examples we refer the reader to Section 5 of Esparza et al. [5]. The exposition of this section follows the similar exposition therein. However, the Master Theorem and the lemmata that imply it require considerable "pastification".

We now make precise the idea expressed in ii). Given a formula φ, word w, and $t \in \mathbb{N}$, we define the set of subformulae in $\mu(\varphi)$ that are satisfied by w at least once at t and the set of subformulae in $\mu(\varphi)$ that are satisfied by w infinitely often at t. Similarly, we define the set of subformulae in $\nu(\varphi)$ that are always satisfied by w at t and the set of subformulae in $\nu(\varphi)$ that are almost always satisfied by w at t:

$$\mathcal{F}^{\varphi}_{w,t} \coloneqq \{\psi \in \mu(\varphi) \mid (w,t) \models \mathbf{F}\psi\} \quad \mathcal{GF}^{\varphi}_{w,t} \coloneqq \{\psi \in \mu(\varphi) \mid (w,t) \models \mathbf{GF}\psi\}$$
$$\mathcal{G}^{\varphi}_{w,t} \coloneqq \{\psi \in \nu(\varphi) \mid (w,t) \models \mathbf{G}\psi\} \quad \mathcal{FG}^{\varphi}_{w,t} \coloneqq \{\psi \in \nu(\varphi) \mid (w,t) \models \mathbf{FG}\psi\}.$$

As mentioned, we are in particular interested in the point at which the two sets in each row coincide. We express this as the word being *stable* at that point.

▶ **Definition 15** (Stable words). *A word w is μ-stable (ν-stable) with respect to a formula φ at index t if $\mathcal{F}^{\varphi}_{w,t} = \mathcal{GF}^{\varphi}_{w,t}$ ($\mathcal{G}^{\varphi}_{w,t} = \mathcal{FG}^{\varphi}_{w,t}$). If w is both μ-stable and ν-stable with respect to φ at index t, then it is stable with respect to φ at index t.*

▶ **Lemma 16.** *Let φ be a formula and w a word. Then there exists an index $r \in \mathbb{N}$ such that w is stable with respect to φ at all indices $t \geq r$.*

The following two definitions specify how to rewrite a formula according to sets M and N, as indicated in i):

▶ **Definition 17.** *Let φ be a formula and M a set of μ-pLTL-formulae. The formula $\varphi[M]_\nu$ is inductively defined as*

$$\begin{aligned}
a[M]_\nu &\coloneqq a && (a \text{ atomic})\\
(op\,\psi)[M]_\nu &\coloneqq op\,(\psi[M]_\nu) && (op \text{ unary})\\
(\psi\;op\;\xi)[M]_\nu &\coloneqq (\psi[M]_\nu)\;op\;(\xi[M]_\nu) && (op \in \{\mathbf{W},\mathbf{R},\mathbf{S},\tilde{\mathbf{S}},\mathbf{B},\tilde{\mathbf{B}}\})\\
(\psi\,\mathbf{U}\,\xi)[M]_\nu &\coloneqq \begin{cases}(\psi[M]_\nu)\,\mathbf{W}\,(\xi[M]_\nu) & (\psi\,\mathbf{U}\,\xi \in M)\\ \bot & (\text{otherwise})\end{cases}\\
(\psi\,\mathbf{M}\,\xi)[M]_\nu &\coloneqq \begin{cases}(\psi[M]_\nu)\,\mathbf{R}\,(\xi[M]_\nu) & (\psi\,\mathbf{M}\,\xi \in M)\\ \bot & (\text{otherwise}),\end{cases}
\end{aligned}$$

▶ **Definition 18.** *Let φ be a formula and N a set of ν-pLTL-formulae. The formula $\varphi[N]_\mu$ is inductively defined as*

$$\begin{aligned}
(\psi\,\mathbf{W}\,\xi)[N]_\mu &\coloneqq \begin{cases}\top & (\psi\,\mathbf{W}\,\xi \in N)\\ (\psi[N]_\mu)\,\mathbf{U}\,(\xi[N]_\mu) & (\text{otherwise})\end{cases}\\
(\psi\,\mathbf{R}\,\xi)[N]_\mu &\coloneqq \begin{cases}\top & (\psi\,\mathbf{R}\,\xi \in N)\\ (\psi[N]_\mu)\,\mathbf{M}\,(\xi[N]_\mu) & (\text{otherwise}).\end{cases}
\end{aligned}$$

The other cases are defined by recursive descent similarly to Definition 17.

Notice that once a formula has been rewritten by M, it becomes a ν-formula. Dually, once a formula has been rewritten by N, it becomes a μ-formula. In terms of the hierarchy of Manna and Pnueli [10], these are safety and guarantee formulae, respectively. It is relatively simple to construct deterministic automata for such formulae as they do not require complicated acceptance conditions.

▶ **Theorem 19** (The Master Theorem for $pLTL$). *Let φ be a formula and w a word stable with respect to φ at index r. Then $w \models \varphi$ iff there exist $M \subseteq \mu(\varphi)$ and $N \subseteq \nu(\varphi)$ such that*

1) $w_r \models af_\ell(\varphi, w_{0r}, \vec{C}_{\varphi,r})[M\langle \circ\, \vec{C}_{\varphi,r}\rangle]_\nu$.

2) $\forall \psi \in M \,.\, \forall s \,.\, \exists t \geq s \,.\, w_t \models \mathbf{F}(\psi\langle \circ\, \vec{C}_{\varphi,t}\rangle[N\langle \circ\, \vec{C}_{\varphi,t}\rangle]_\mu)$.

3) $\forall \psi \in N \,.\, \exists t \geq 0 \,.\, w_t \models \mathbf{G}(\psi\langle \circ\, \vec{C}_{\varphi,t}\rangle[M\langle \circ\, \vec{C}_{\varphi,t}\rangle]_\nu)$.

The statements of the Master Theorem in the only if-direction can be significantly strengthened. The existential quantification over t in premise 2 can be made universal. The statement $w_t \models \ldots$ in premise 3 holds for all $t' \geq r$. Given the semantics of $\mathbf{F}$ and $\mathbf{G}$, this is not a surprise. However, the ability to do the rewrites at every given moment is technically involved due to the incorporation of the past. In the next section we show how to use the Master Theorem in the construction of a DRA.

6 From $pLTL$ to DRA

We are now ready, based on the Master Theorem, to construct a DRA for the language of a formula φ. We decompose the language of φ into a Boolean combination of languages, each of which is recognized by a deterministic automaton with the relatively simple acceptance condition of Büchi or co-Büchi. For every possible pair of sets M and N of μ- and ν-subformulae we try to establish the premises of the Master Theorem. Premise 1 can be checked by trying to identify a stability point r from which the safety automaton for $af_\ell(\varphi, w_{0r}, \vec{C}_{\varphi,r})[M\langle \circ\, \vec{C}_{\varphi,t}\rangle]_\nu$ continues forever. Whenever this safety check fails, simply try again. Overall, this corresponds to a co-Büchi condition and a DCA. Premise 2 can be checked for every $\psi \in M$ by identifying infinitely many points from which the guarantee automaton for $\mathbf{F}(\psi\langle \circ\, \vec{C}_{\varphi,t}\rangle[N\langle \circ\, \vec{C}_{\varphi,t}\rangle]_\mu)$ finishes its check. Overall, this corresponds to a Büchi condition and a DBA. Premise 3 is dual to premise 2 and leads to a DCA. The three together are combined to a DRA with one pair. Overall, we get a DRA with exponentially many Rabin pairs; one for each choice of M and N. We will as shorthand make use of the operators $\mathbf{F}$ and $\mathbf{G}$ in the construction of these automata, as described in Section 2.2.

The major difficulty of incorporating the past into this part, is that the rewriting using the set M and N needs to be done with the past subformulae correctly weakened. To facilitate this, we begin by defining an auxiliary automaton in Section 6.1 that serves to track the weakening conditions that must hold in order to justify rewrites by $\cdot\langle\cdot\rangle$.

The state spaces of the automata we construct are defined in terms of the following notions. Given a formula φ we denote by $\mathbb{B}(\varphi)$ the set of formulae ψ satisfying $sf(\psi) \subseteq sf(\varphi) \cup \{\top, \bot\}$. Similarly, consider a formula that is a disjunction of formulae with the property that for each such disjunct ψ there exists a $C \in 2^{psf(\varphi)}$ such that $\psi \in \mathbb{B}(\varphi\langle C\rangle)$. We denote by $\mathbb{B}^\vee(\varphi)$ the set of all such formulae. A key observation is that the sizes of the quotient sets $\mathbb{B}(\varphi)_{/\sim}$ and $\mathbb{B}^\vee(\varphi)_{/\sim}$ are doubly exponential in the size of φ.

For the remainder of this section we consider a fixed formula φ that is to be translated into a Rabin automaton and a fixed ordering $C_1, C_2, \ldots, C_k$ of the elements of $2^{psf(\varphi)}$. For simplicity, we assume $C_1 = \{\psi \in psf(\varphi) \mid \psi = \psi_{\mathcal{W}}\} = C_{\varphi,0}$. We freely make use of af, $\cdot[\cdot]_\nu$, and $\cdot[\cdot]_\mu$ lifted to equivalence classes of formulae. This is justified by Lemma 2.

6.1 The Weakening Conditions Automaton

The weakening conditions automaton (WC automaton) is a bed automaton that tracks the development of weakening conditions under rewrites of the local after-function. Its states are k-tuples of formulae, each of which describes the requirements that remain to be verified in order to justify a sequence of rewrites. To facilitate the construction of the WC automaton, we define the following function:

▶ **Definition 20.** *Given a k-tuple of formulae $\psi^\times = \langle \psi_1, \psi_2, \ldots, \psi_k \rangle$ and a letter σ, the rewrite condition function is defined as $rc(\psi^\times, \sigma) := \langle \psi'_1, \psi'_2, \ldots, \psi'_k \rangle$, where*

$$\psi'_i = \bigvee_{j \in J_i} \Big(af_\ell(\psi_j, \sigma, C_i\langle C_j\rangle) \wedge \bigwedge_{\xi \in C_i} af_\ell(wc(\xi\langle C_j\rangle), \sigma, C_i\langle C_j\rangle) \Big),$$

and where $J_i := \big\{ j \in [1..k] \mid \forall \xi, \xi' \in psf(\varphi) \,.\, \xi\langle C_j\rangle = \xi'\langle C_j\rangle \Rightarrow \xi\langle C_i\rangle = \xi'\langle C_i\rangle \big\}$.

The definition of J_i ensures that $\psi'_i \in \mathbb{B}(\varphi\langle C_i\rangle)$. The rewrite condition function takes a k-tuple of formulae and a letter, and returns an updated k-tuple. In the resulting tuple, the i^{th} item ψ'_i is a formula that encodes the updated requirements for further applying the rewrite $\cdot\langle C_i\rangle$. We remark that, for every $t > 0$, there exist indices $i \leq k$ and $j \in J_i$ such that $C_i = \circ\vec{C}_{\varphi,t}$, $C_j = \circ\vec{C}_{\varphi,t-1}$, and $C_i\langle C_j\rangle = C_{\varphi,t}$.

We now define the WC automaton $\mathcal{H}_\varphi := (S, S_0, \delta_\mathcal{H})$ over $2^{Var(\varphi)}$. Its set of states S is $\Pi_{i=1}^k \mathbb{B}(\varphi\langle C_i\rangle)_{/\sim}$. The initial state S_0 is the k-tuple $\langle [\top]_\sim, [\bot]_\sim, \ldots, [\bot]_\sim \rangle$, which represents that the set used to rewrite φ into its initial form is known to be C_1. Finally, the transition relation δ is defined by $\delta_\mathcal{H}(\psi^\times, \sigma) = rc(\psi^\times, \sigma)$, with rc lifted to propositional equivalence classes of formulae in the natural way.

6.2 Verifying the Premises of the Master Theorem

We now describe the automata that are capable of verifying the premises of the Master Theorem. They are all runner automata with bed automaton $\mathcal{H}_\varphi := (S, S_0, \delta_\mathcal{H})$.

Given a formula $\psi \in \mu(\varphi)$ and set $N \subseteq \nu(\varphi)$ we define the runner automaton $\mathcal{B}^\psi_{2,N} := (S, Q, Q_0, \delta, \alpha)$ over $2^{Var(\varphi)}$ with set of states $Q := \mathbb{B}^\vee(\varphi[N]_\mu \wedge \mathbf{F}(\psi[N]_\mu))_{/\sim}$, initial state $Q_0 := [\mathbf{F}(\psi[N]_\mu)]_\sim$, and Büchi acceptance condition $\alpha := [\top]_\sim$. Finally, its transition relation is defined as

$$\delta(\zeta, \xi^\times, \sigma) := \begin{cases} \displaystyle\bigvee_{i \in [1..k]} \mathbf{F}(\psi\langle C_i\rangle[N\langle C_i\rangle]_\mu) \wedge \xi_i[N\langle C_i\rangle]_\mu & (\zeta \sim \top) \\ af(\zeta, \sigma) & (\text{otherwise}), \end{cases}$$

where $\xi^\times = \langle \xi_1, \xi_2, \ldots, \xi_k \rangle$. For readability, we express δ in terms of formulae, but ask the reader to note that they represent their corresponding propositional equivalence classes.

Informally, the automaton $\mathcal{B}^\psi_{2,N}$ begins by checking that the input word w satisfies the formula $\mathbf{F}(\psi[N]_\mu)$. Because this formula is in the fragment *μ-pLTL*, it is satisfied by w iff the after-function eventually rewrites it into a propositionally true formula. At this point, the automaton restarts, checking the formula again. Because the subset of $psf(\varphi)$ that puts $\psi[N]_\mu$ in the correct form at this point – with respect to its past subformulae – is unknown, all possible such sets are considered in the form of a disjunction. To each disjunct the corresponding weakening conditions, as tracked by the WC automaton, are added. It follows that $\mathcal{H}_\varphi \ltimes \mathcal{B}^\psi_{2,N}$ is able to verify premise 2) of the Master Theorem for the considered formula ψ and sets M and N.

Given a formula $\psi \in \nu(\varphi)$ and set $M \subseteq \mu(\varphi)$ we define the runner automaton $\mathcal{C}^{\psi}_{3,M} \coloneqq (S, Q, Q_0, \delta, \alpha)$ over $2^{\mathit{Var}(\varphi)}$ with set of states $Q \coloneqq \mathbb{B}^{\vee}(\varphi[M]_{\nu} \wedge \mathbf{G}(\psi[M]_{\nu}))_{/\sim}$, initial state $Q_0 \coloneqq [\mathbf{G}(\psi[M]_{\nu})]_{\sim}$, and co-Büchi acceptance condition $\alpha \coloneqq [\bot]_{\sim}$. Finally, its transition relation is defined as

$$\delta(\zeta, \xi^{\times}, \sigma) \coloneqq \begin{cases} \bigvee_{i \in [1..k]} \mathbf{G}(\psi\langle C_i\rangle[M\langle C_i\rangle]_{\nu}) \wedge \xi_i[M\langle C_i\rangle]_{\nu} & (\zeta \sim \bot) \\ \mathit{af}(\zeta, \sigma) & \text{(otherwise)}, \end{cases}$$

where $\xi^{\times} = \langle \xi_1, \xi_2, \ldots, \xi_k\rangle$. As before, the formulae of δ represent their corresponding equivalence classes. By a similar argument as before, the automaton $\mathcal{H} \ltimes \mathcal{C}^{\psi}_{3,M}$ is able to verify premise 3) of the Master Theorem for the formula ψ and sets M and N.

We now turn to constructing automata for verifying the first premise of the Master Theorem. Given a set $M \subseteq \mu(\varphi)$, we define the runner automaton $\mathcal{C}^{1}_{\varphi,M} \coloneqq (S, Q, Q_0, \delta, \alpha)$ over $2^{\mathit{Var}(\varphi)}$ with set of states $Q \coloneqq \mathbb{B}^{\vee}(\varphi)_{/\sim} \times \mathbb{B}^{\vee}(\varphi[M]_{\nu})_{/\sim}$, initial state $Q_0 \coloneqq \langle [\varphi]_{\sim}, [\varphi[M]_{\nu}]_{\sim}\rangle$, and co-Büchi acceptance condition $\alpha \coloneqq \mathbb{B}^{\vee}(\varphi)_{/\sim} \times \{[\bot]_{\sim}\}$. Its transition relation is defined as

$$\delta(\langle \psi, \zeta\rangle, \xi^{\times}, \sigma) \coloneqq \begin{cases} \Big\langle \mathit{af}(\psi, \sigma), \bigvee_{i \in [1..k]} \mathit{af}(\psi, \sigma)[M\langle C_i\rangle]_{\nu} \wedge \xi_i[M\langle C_i\rangle]_{\nu} \Big\rangle & (\zeta \sim \bot) \\ \Big\langle \mathit{af}(\psi, \sigma), \mathit{af}(\zeta, \sigma) \Big\rangle & \text{(otherwise)}, \end{cases}$$

where $\xi^{\times} = \langle \xi_1, \xi_2, \ldots, \xi_k\rangle$. Again, we remind the reader that the formulae in the above definition represent equivalence classes of formulae. The purpose of the above automaton is to "guess" an index at which w is stable with respect to φ, starting with the guess that it is initially stable. Both φ and $\varphi[M]$ are evaluated in tandem. If the current guess of point of stability is incorrect, the second component will eventually collapse into a formula propositionally equivalent to $\bot$. At this point, the automaton proceeds with a new guess by reapplying $\cdot[M]_{\nu}$ to φ as it has currently been transformed by the after-function. As with the other automata, the formulae that make up the states of the WC automaton are used to justify the rewrite by each set C_i. The automaton $\mathcal{H}_{\varphi} \ltimes \mathcal{C}^{1}_{\varphi,M}$ is able to verify premise 1) of the Master Theorem for the formula φ and the set M.

6.3 The Rabin Automaton

We now construct the final deterministic Rabin automaton. The automaton is the disjunction of up to 2^n simpler Rabin automata; one for every possible choice of $M \subseteq \mu(\varphi)$ and $N \subseteq \mu(\varphi)$. Taking the disjunction of deterministic Rabin automata is possible by running automata for all disjuncts in parallel (via a product construction) and taking the acceptance condition that checks that at least one of them is accepting. Each simpler Rabin automaton checks the three premises of the Master Theorem for its specific M and N: (a) $M \subseteq \mathcal{GF}^{\varphi}_{w,0}$, (b) $N \subseteq \mathcal{FG}^{\varphi}_{w,0}$, and (c) w satisfies a version of φ simplified by M. Each of the three is checked by a Büchi or co-Büchi automaton. In order to check all three we have to consider their conjunction. For a Rabin automaton (with one pair) it is possible to check the conjunction of Büchi and co-Büchi by running automata for all conjuncts in parallel (product construction) and using the Rabin acceptance condition (one pair) to ensure that all co-Büchi automata and all Büchi automata are accepting.

▶ **Theorem 21.** *Let φ be a formula. For each $M \subseteq \mu(\varphi)$ and $N \subseteq \nu(\varphi)$, define*

$$\mathcal{B}^2_{M,N} \coloneqq \bigcap_{\psi\in M} \mathcal{H}_\varphi \ltimes \mathcal{B}^\psi_{2,N} \qquad \mathcal{C}^3_{M,N} \coloneqq \bigcap_{\psi\in N} \mathcal{H}_\varphi \ltimes \mathcal{C}^\psi_{3,M}$$
$$\mathcal{R}_{\varphi,M,N} \coloneqq (\mathcal{H}_\varphi \ltimes \mathcal{C}^1_{\varphi,M}) \cap \mathcal{B}^2_{M,N} \cap \mathcal{C}^3_{M,N},$$

where $\mathcal{R}_{\varphi,M,N}$ has one Rabin pair. Then the following DRA over $2^{Var(\varphi)}$ recognizes φ:

$$\mathcal{A}_{DRA}(\varphi) \coloneqq \bigcup_{\substack{M\subseteq\mu(\varphi)\\ N\subseteq\nu(\varphi)}} \mathcal{R}_{\varphi,M,N}.$$

▶ **Corollary 22.** *Let φ be a formula of size $n + m$, where n is the number of future- and propositional subformulae of φ, and m is the number of past subformulae of φ. There exists a deterministic Rabin automaton recognizing φ with doubly exponentially many states in the size of the formula and at most 2^n Rabin pairs.*

7 Discussion

We presented a direct translation from *pLTL* to deterministic Rabin automata. Starting from a formula with n future subformulae and m past subformulae, we produce an automaton with an optimal $2^{2^{O(n+m)}}$ states and $2^{O(n)}$ acceptance pairs. Our translation relies on extending the classical "after"-function of *LTL* to *pLTL* by encoding memory about the past through the weakening and strengthening of embedded past operators. We extended the Master Theorem about decomposition of languages expressed for *LTL* to *pLTL*.

The only applicable approach (prior to our work) to obtain deterministic automata from *pLTL* formulae was to convert the formula to a nondeterministic automaton [14] and then determinize this automaton [16, 13]. The first can be done either directly [9, 14] or through two-way very-weak alternating automata [7]. In any case, the first translates a formula with n future operators and m past operators to an automaton with $2^{O(n+m)}$ states and the second translates an automaton with k states to a parity automaton with $O(k!^2)$ states and $O(k)$ priorities. It follows that the overall complexity of this construction is $2^{2^{O((n+m)\cdot\log(n+m))}}$ states and $2^{O(n+m)}$ priorities. Our approach improves this upper bound to $2^{2^{O(m+n)}}$. It is well known that *pLTL* does not extend the expressive power of *LTL*. However, conversion from *pLTL* to *LTL* is not viable algorithmically. The best known translation is worst-case non-elementary [6], and the conversion is provably exponential [11]. So using a conversion to *LTL* as a preliminary step to determinization could result in a triple exponential construction. We note that in the case where there are no future operators nested within past operators, it is possible to convert the past subformulae directly to deterministic automata. Then, the remaining future can be determinized independently. This approach has been advocated for usage of the past in reactive synthesis [3, 4] and implemented recently in a bespoke tool [2].

As future work, we note that the approach of Esparza et al. [5] additionally led to translations from *LTL* to nondeterministic automata, limit-deterministic automata, and deterministic automata. The same should be done for *pLTL*. Their work also led to a normal form for *LTL* formulae, which we believe could be generalized to work for *pLTL*. The latter could have interesting relations to the temporal hierarchy of Manna and Pnueli [10]. Such a normal form could also be related to more efficient translations from *pLTL* to *LTL*. Finally, this approach has led to a competitive implementation of determinization [8] and reactive synthesis [12]. Extending these implementations to handle past is of high interest.

References

1 Shaun Azzopardi, David Lidell, and Nir Piterman. A Direct Translation from LTL with Past to Deterministic Rabin Automata, 2024. arXiv:2405.01178.

2 Shaun Azzopardi, David Lidell, Nir Piterman, and Gerardo Schneider. ppLTLTT : Temporal Testing for Pure-Past Linear Temporal Logic Formulae. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part II*, volume 14216 of *Lecture Notes in Computer Science*, pages 276–287. Springer, 2023. doi:10.1007/978-3-031-45332-8_15.

3 Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of Reactive(1) Designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012. doi:10.1016/J.JCSS.2011.08.007.

4 Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. Reactive Synthesis from Extended Bounded Response LTL Specifications. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 83–92. IEEE, 2020. doi:10.34727/2020/ISBN.978-3-85448-042-6_15.

5 Javier Esparza, Jan Křetínský, and Salomon Sickert. A Unified Translation of Linear Temporal Logic to ω-Automata. *J. ACM*, 67(6), October 2020. doi:10.1145/3417995.

6 Dov M. Gabbay. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In Behnam Banieqbal, Howard Barringer, and Amir Pnueli, editors, *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 409–448, New York, NY, USA, 1987. Springer. doi:10.1007/3-540-51803-7_36.

7 Paul Gastin and Denis Oddoux. LTL with Past and Two-Way Very-Weak Alternating Automata. In Branislav Rovan and Peter Vojtás, editors, *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003, Bratislava, Slovakia, August 25-29, 2003, Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 439–448, New York, NY, USA, 2003. Springer. doi:10.1007/978-3-540-45138-9_38.

8 Jan Kretínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A Library for ω-Words, Automata, and LTL. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550. Springer, 2018. doi:10.1007/978-3-030-01090-4_34.

9 Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The Glory of the Past. In Rohit Parikh, editor, *Logics of Programs*, pages 196–218, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

10 Zohar Manna and Amir Pnueli. A Hierarchy of Temporal Properties. In Cynthia Dwork, editor, *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 377–410. ACM, 1990. doi:10.1145/93385.93442.

11 Nicolas Markey. Temporal Logic with Past is Exponentially More Succinct. *Bulletin- European Association for Theoretical Computer Science*, 79:122–128, 2003. URL: https://hal.science/hal-01194627.

12 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018. doi:10.1007/978-3-319-96145-3_31.

13 Nir Piterman. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, LICS '06, pages 255–264, USA, 2006. IEEE Computer Society. doi:10.1109/LICS.2006.28.

14 Nir Piterman and Amir Pnueli. Temporal Logic and Fair Discrete Systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 27–73. Springer, New York, NY, USA, 2018. doi:10.1007/978-3-319-10575-8_2.

15 Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, USA, 1977. IEEE Computer Society. doi:10.1109/SFCS.1977.32.

16 S. Safra. On the Complexity of Omega-Automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 319–327, USA, 1988. IEEE Computer Society. doi:10.1109/SFCS.1988.21948.

17 A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, July 1985. doi:10.1145/3828.3837.

Logical Characterizations of Weighted Complexity Classes

Guillermo Badia
The University of Queensland, Brisbane, Australia

Manfred Droste
Leipzig University, Germany

Carles Noguera
University of Siena, Italy

Erik Paul
Leipzig University, Germany

Abstract

Fagin's seminal result characterizing NP in terms of existential second-order logic started the fruitful field of descriptive complexity theory. In recent years, there has been much interest in the investigation of quantitative (weighted) models of computations. In this paper, we start the study of descriptive complexity based on weighted Turing machines over arbitrary semirings. We provide machine-independent characterizations (over ordered structures) of the weighted complexity classes $\mathsf{NP}[\mathcal{S}], \mathsf{FP}[\mathcal{S}]$, $\mathsf{FPLOG}[\mathcal{S}]$, $\mathsf{FPSPACE}[\mathcal{S}]$, and $\mathsf{FPSPACE}_{poly}[\mathcal{S}]$ in terms of definability in suitable weighted logics for an arbitrary semiring $\mathcal{S}$. In particular, we prove weighted versions of Fagin's theorem (even for arbitrary structures, not necessarily ordered, provided that the semiring is idempotent and commutative), the Immerman–Vardi's theorem (originally for P) and the Abiteboul–Vianu–Vardi's theorem (originally for PSPACE). We also discuss a recent open problem proposed by Eiter and Kiesel.

Recently, the above mentioned weighted complexity classes have been investigated in connection to classical counting complexity classes. Furthermore, several classical counting complexity classes have been characterized in terms of particular weighted logics over the semiring $\mathbb{N}$ of natural numbers. In this work, we cover several of these classes and obtain new results for others such as NPMV, $\oplus\mathsf{P}$, or the collection of real-valued languages realized by polynomial-time real-valued nondeterministic Turing machines. Furthermore, our results apply to classes based on many other important semirings, such as the max-plus and the min-plus semirings over the natural numbers which correspond to the classical classes $\mathsf{MaxP}[O(\log n)]$ and $\mathsf{MinP}[O(\log n)]$, respectively.

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

Keywords and phrases Descriptive complexity, Weighted Turing machines, Weighted logics, Semirings

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.14

Funding *Guillermo Badia*: Australian Research Council grant DE220100544
Carles Noguera: European Union's Marie Sklodowska–Curie grant no. 101007627 (MOSAIC project). GNSAGA, gruppo nazionale di ricerca INdAM

Acknowledgements We are grateful to three anonymous referees for their numerous useful remarks that contributed to improving the presentation of the paper.

1 Introduction

Descriptive complexity is a branch of computational complexity, as well as finite model theory, where the difficulty in solving a problem by a Turing machine is characterized not by the amount of resources required (such as time, space and so on) but rather in terms of the complexity of describing the problem in some logical formalism. This field was

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 14; pp. 14:1–14:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

initially started in 1974 by Ronald Fagin with the celebrated result in [19] (coined by Neil Immerman as "Fagin's theorem") which stated that the class of NP languages coincides with the class of languages definable in existential second-order logic. Many further surprising results followed this development, particularly the Immerman–Vardi's theorem characterizing P over ordered structures using fixed-point logic [31, 48] and the Abiteboul–Vianu–Vardi characterization of PSPACE in terms of partial fixed-point logic [1, 48]. Today there are several textbooks that cover the fundamentals of the area as a line of research within finite model theory [15, 38, 25, 32]. In this paper, we propose to study quantitative versions of some of these key results in this important field in connection with weighted computation. We work over finite structures that come with a linear ordering, which is a standard restriction in descriptive complexity.

Weighted automata are nondeterministic finite automata augmented with values from a semiring as weights on the transitions [45]. These weights may model, e.g. the cost involved when executing a transition, the amount of resources or time needed for this, or the probability or reliability of its successful execution. The theory of weighted automata and weighted context-free grammars was essential for the solution of such classical automata-theoretic problems as the decidability of the equivalence of unambiguous context-free languages and regular languages [43] (in fact, the only known proofs of this involve weighted automata), the decidability of two given deterministic multitape automata [30], and the decidability of two given deterministic pushdown automata [39, 46]. This led to quick development of this field, described in the books [6, 12, 16, 36, 42, 43]. Furthermore, weighted automata and weighted context-free grammars have been used as basic concepts in natural language processing and speech recognition, as well as in algorithms for digital image compression [2]. Weighted logic [11], with weights in an arbitrary semiring, was developed originally to obtain a weighted version of the Büchi–Elgot–Trakhtenbrot theorem, showing that a certain weighted monadic second-order logic has the same expressive power on words as weighted automata. Consequently, this weighted logic over suitable semirings like fields has similar decidability properties on words as unweighted monadic second-order logic. It is worth remarking that the classical Büchi–Elgot–Trakhtenbrot theorem is usually regarded as part of the "prehistory" of descriptive complexity [25, p. 145].

Weighted Turing machines extend the concept of weighted automata as natural quantitative counterparts of classical Turing machines. They were first introduced under the name "algebraic Turing machines" in [10, 9] and they have attracted further attention in [34]. Instances of this concept include the so called "fuzzy Turing machines" [49, 4]. Recently, the articles [18, 17] have introduced a related notion of "semiring Turing machine" and explicitly asked for the development of descriptive complexity in such framework as an open problem, focusing specifically on Fagin's theorem in connection to weighted logic [18, p. 255]. We will address this problem at the end of Section 5.

Our contribution. The present paper develops a theory of weighted descriptive complexity and establishes quantitative versions of some celebrated classical theorems. The novel contributions of this work can be summarized in the following characterizations (for an arbitrary semiring $\mathcal{S}$):

- The weighted complexity class $\mathsf{NP}[\mathcal{S}]$ coincides with the queries definable by weighted existential second-order logic on ordered structures, with weights in $\mathcal{S}$, respectively for all structures if $\mathcal{S}$ is idempotent and commutative (Theorem 22).
- The weighted complexity class $\mathsf{FP}[\mathcal{S}]$ coincides with the queries definable by weighted inflationary fixed-point logic, with weights in $\mathcal{S}$ (Theorem 26).

- The weighted complexity class $\mathsf{FPSPACE}[\mathcal{S}]$ coincides with the queries definable by weighted partial fixed-point logic with the addition of second-order multiplicative and additive quantifiers, with weights in $\mathcal{S}$ (Theorem 29).
- The weighted complexity class $\mathsf{FPSPACE}_{poly}[\mathcal{S}]$ coincides with the queries definable by weighted partial fixed-point logic, with weights in $\mathcal{S}$ (Theorem 31).
- The weighted complexity class $\mathsf{FPLOG}[\mathcal{S}]$ coincides with the queries definable by weighted deterministic transitive closure logic (Theorem 33).

Related work. Despite the fact that some characterizations of counting complexity classes using Boolean logics were known [33, 44, 14], observe that the article [3] (following up on the work of [44]) already proposes the idea of using certain weighted logics (with weights in the semiring $\mathbb{N}$ of natural numbers or, in a couple of cases, $\mathbb{Z}$) to characterize well-known counting complexity classes. The authors obtain several interesting results that are also covered by our more encompassing work here (that is, they provide logical characterizations of #P, FP, FPSPACE, FPSPACE($poly$), GapP, and MaxP). There is, however, some orthogonality as they cover some classical complexity classes that we do not and, similarly, we cover some that they do not, as we do not restrict our semiring to being $\mathbb{N}$ or $\mathbb{Z}$. Moreover, the investigation in [3], by contrast to ours, concentrates on the study of classical counting classes for ordered structures, while we consider both ordered and arbitrary structures (provided, in the latter case, that the semiring is idempotent and commutative; examples include e.g. the max-plus- and min-plus-semirings). In the present article, the central aim is rather starting the study of weighted complexity classes via logic, and the corollaries characterizing classical complexity classes are obtained as interesting byproducts of the work. In this way, we are also meeting the challenge posed in [34, p.3] of developing "quantitative descriptive complexity theory based on weighted logics [...] over some fairly general class of semirings". Further work on the model theory of weighted logics includes a Feferman–Vaught result [13], but the area remains largely unexplored despite being one of the open problems suggested in [11]. Finally, an approach related to the weighted logics discussed here has been recently proposed in [26, 27] motivated by problems in database theory [23]. The idea there is that the atomic facts of a model are annotated by values from a semiring whereas in the present paper this aspect is fully classical.

2 Weighted Turing machines

In order to introduce the notion of a weighted Turing machine, first we need to define the kind of algebraic structures that will provide the weights, that is, semirings.

▶ **Definition 1** (Semirings). *A* semiring *is a tuple $\mathcal{S} = \langle S, +, \cdot, \mathbb{0}, \mathbb{1}\rangle$, with operations addition $+$ and multiplication $\cdot$ and constants $\mathbb{0}$ and $\mathbb{1}$ such that*

- *$\langle S, +, \mathbb{0}\rangle$ is a commutative monoid and $\langle S, \cdot, \mathbb{1}\rangle$ is a monoid,*
- *multiplication distributes over addition, and*
- *$s \cdot \mathbb{0} = \mathbb{0} \cdot s = \mathbb{0}$ for every $s \in S$.*

We say that $\mathcal{S}$ is commutative *if the monoid $\langle S, \cdot, \mathbb{1}\rangle$ is commutative, and we say that $\mathcal{S}$ is* idempotent *if the monoid $\langle S, +, \mathbb{0}\rangle$ is idempotent (that is, $s + s = s$ for each $s \in S$).*

Some examples of semirings, including those that we will use in this paper, are the following:

- the *Boolean semiring* $\mathbb{B} = \langle\{0, 1\}, \min, \max, 0, 1\rangle$,
- any bounded distributive lattice $\langle L, \vee, \wedge, 0, 1\rangle$,
- the semiring of natural numbers $\langle \mathbb{N}, +, \cdot, 0, 1\rangle$,

- the semiring of extended natural numbers $\langle \mathbb{N} \cup \{+\infty\}, +, \cdot, 0, 1\rangle$ where $0 \cdot (+\infty) = 0$,
- the ring of integers, $\langle \mathbb{Z}, +, \cdot, 0, 1\rangle$,
- the ring of integers modulo n, $\langle \mathbb{Z}_n, +_n, \cdot_n, \overline{0}, \overline{1}\rangle$, for each $n \in \mathbb{N}$,
- the field of rational numbers $\langle \mathbb{Q}, +, \cdot, 0, 1\rangle$,
- the *max-plus* or *arctic semiring* $\mathrm{Arct} = \langle \mathbb{R}_+ \cup \{-\infty\}, \max, +, -\infty, 0\rangle$, where $\mathbb{R}_+$ denotes the set of non-negative real numbers,
- the restriction of the arctic semiring to the natural numbers $\mathbb{N}_{\max} = \langle \mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0\rangle$,
- the *min-plus* or *tropical semiring* $\mathrm{Trop} = \langle \mathbb{R}_+ \cup \{+\infty\}, \min, +, +\infty, 0\rangle$,
- the restriction of the tropical semiring to the natural numbers $\mathbb{N}_{\min} = \langle \mathbb{N} \cup \{+\infty\}, \min, +, +\infty, 0\rangle$,
- the semiring $\mathcal{F}_* = \langle [0,1], \max, *, 0, 1\rangle$ given by a t-norm $*$ [49],
- the semiring of finite languages $2^{\Sigma^*}_{\text{fin}} = \langle 2^{\Sigma^*}_{\text{fin}}, \cup, \cdot, \emptyset, \{\varepsilon\}\rangle$, for an alphabet Σ,
- the semiring $\mathcal{S}_{\max} = \langle \{0,1\}^* \cup \{-\infty\}, \max, \cdot, -\infty, \varepsilon\rangle$ of binary words in which max is computed according to the *radix order* (for $x, y \in \{0,1\}^*$, $x \preceq y$ iff $|x| < |y|$ or $|x| = |y|$ and x is smaller than or equal to y in the lexicographic order) and $\max(x, -\infty) = \max(-\infty, x) = x$ for each x, $\cdot$ is the concatenation operation, and $x \cdot (-\infty) = (-\infty) \cdot x = -\infty$ for each x,
- the semiring $\mathcal{S}_{\min} = \langle \{0,1\}^* \cup \{+\infty\}, \min, \cdot, +\infty, \varepsilon\rangle$ analogous to the previous one.

▶ **Definition 2** (Weighted Turing Machines). *Let $\mathcal{S}$ be a semiring and Σ an alphabet. A* weighted (or algebraic) Turing machine *over $\mathcal{S}$ and input alphabet Σ is a septuple $\mathcal{M} = \langle Q, \Gamma, \Delta, \nu, q_0, F, \square\rangle$, where*

- *Q is a nonempty finite set whose elements are called* states*,*
- *$\Gamma \supseteq \Sigma$ is an alphabet (*working alphabet*),*
- *$\Delta \subseteq (Q \setminus F) \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$ and its elements are called* transitions*,*
- *$\nu \colon \Delta \longrightarrow S$ is called a* transition weighting function*, $q_0 \in Q$ is called the* initial state*, $F \subseteq Q$ and its elements are called* accepting states*, and $\square \in \Gamma \setminus \Sigma$ is the blank symbol.*

We call $\mathcal{M}$ a Turing machine *if $\mathcal{S}$ is the Boolean semiring $\mathbb{B}$. We call $\mathcal{M}$* deterministic *if for every pair $\langle p, a\rangle \in Q \times \Gamma$, there is at most one transition $\langle p, a, q, b, d\rangle \in \Delta$.*

A *configuration* of $\mathcal{M}$ is a unique description of the machine's state, contents of the working tape, and the position of the machine's head. If $e = \langle p, c, q, d, t\rangle \in \Delta$ is a transition and C_1, C_2 are configurations of $\mathcal{M}$, then we write $C_1 \longrightarrow_e C_2$ if C_1 is a configuration with state p and the head reading c, while C_2 is obtained from C_1 by changing state to q, rewriting the originally read symbol c to d, and moving the head as prescribed by t. We write $C_1 \longrightarrow C_2$ if $C_1 \longrightarrow_e C_2$ for some $e \in \Delta$.

A *computation* of $\mathcal{M}$ is a word $\gamma = C_1 e_1 C_2 e_2 C_3 \dots C_n e_n C_{n+1}$ such that $C_1, \dots, C_{n+1}$ are configurations of $\mathcal{M}$, $e_1, \dots, e_n \in \Delta$, $C_k \longrightarrow_{e_k} C_{k+1}$ for each $k \in \{1, \dots, n\}$, and C_1 is a configuration with state q_0 and the head at the leftmost non-blank cell (if there is some). The *weight* of γ is defined as $\nu(\gamma) := \nu(e_1)\nu(e_2)\dots\nu(e_n)$. γ is called an *accepting* computation if C_{n+1} has an accepting state. We say that γ is a computation on w in Σ^*, and write $\Sigma(\gamma) = w$ if C_1 is a configuration with w on the working tape. We denote the set of all computations of $\mathcal{M}$ by $C(\mathcal{M})$ and the set of all accepting computations by $A(\mathcal{M})$.

▶ **Convention 3.** *From now on we will assume that every Turing machine $\mathcal{M}$ is* finitely terminating*, that is, the set $C_w(\mathcal{M}) = \{\gamma \in C(\mathcal{M}) \mid \Sigma(\gamma) = w\}$ is finite for each $w \in \Sigma^*$. In particular, the set $A_w(\mathcal{M}) = \{\gamma \in A(\mathcal{M}) \mid \Sigma(\gamma) = w\}$ is finite.*

By a *series* σ we mean a mapping $\sigma\colon \Sigma^* \longrightarrow S$ where Σ is an alphabet, Σ^* the corresponding language and $\mathcal{S}$ is a semiring. Thanks to the convention, we can introduce the following notion:

▶ **Definition 4** (Behavior of a weighted Turing machine). *Let $\mathcal{M}$ be a weighted Turing machine. The* behavior *of $\mathcal{M}$ as the mapping $\|\mathcal{M}\|\colon \Sigma^* \longrightarrow S$ defined as*

$$\|\mathcal{M}\|(w) := \sum_{\gamma \in A_w(\mathcal{M})} \nu(\gamma).$$

We say that a series $\sigma\colon \Sigma^* \longrightarrow S$ is recognized *by a weighted Turing machine $\mathcal{M}$ if $\|\mathcal{M}\| = \sigma$.*

The definition of weighted Turing machine we have used here is exactly the same as that of algebraic Turing machines [10, Def. 5.1] (see also [34]). Similarly, the notion of the behavior of the machine coincides. The semiring Turing machines of [17, 18], by contrast, differ in that they impose some conditions on the allowed transitions [18, cf. Def. 12]. Given distributivity of multiplication over addition, the notion of a semiring Turing machine function in [18, Def. 13] coincides with that of the behavior we use here. Semiring Turing Machines allow semiring values on the tape in somewhat of a black-box manner. Intuitively, one can transition with the weight of the value on the tape, but cannot differentiate the values on the tape or modify them. If semiring values are not allowed in the input string then the definition of weighted and semiring Turing Machines are equivalent (in the sense that one can be transformed into the other without a significant change of execution time). All these definitions generalize the corresponding notions for weighted automata.

3 Some weighted complexity classes

Let $\mathcal{M} = \langle Q, \Gamma, \Delta, \nu, q_0, F, \square\rangle$ be a weighted Turing machine over $\mathcal{S}$ and Σ. For $w \in \Sigma^*$, we denote by $\mathsf{TIME}(\mathcal{M}, w)$ the maximal length of a computation of $\mathcal{M}$ on w, and define, for $n \in \mathbb{N}$, $\mathsf{TIME}(\mathcal{M}, n) := \max\{\mathsf{TIME}(\mathcal{M}, w) : w \in \Sigma^*, |w| \leq n\}$.

For a function $f\colon \mathbb{N} \longrightarrow \mathbb{N}$, we denote by $\mathsf{SERIES}[S, \Sigma](f)$ the set of all series σ such that $\sigma = \|\mathcal{M}\|$ for some weighted Turing machine $\mathcal{M}$ over $\mathcal{S}$ and Σ with $\mathsf{TIME}(\mathcal{M}, n) = O(f(n))$. Now we can define the complexity classes:

$\mathsf{SERIES}[\mathcal{S}](f(n)) := \bigcup\{\mathsf{SERIES}[S, \Sigma](f(n)) : \Sigma \text{ is an alphabet}\}$.

▶ **Definition 5.** *Let $\mathcal{S}$ be a semiring. We define the following weighted complexity class*

$$\mathsf{NP}[\mathcal{S}] := \bigcup\{\mathsf{SERIES}[\mathcal{S}](n^k) : k \in \mathbb{N}\}.$$

$\mathsf{NP}[\mathcal{S}]$ (cf. [34, Def. 4.1]) coincides with the definition of the class $\mathcal{S}$-#P in [10, Def. 5.2]. Furthermore, it is contained as a subclass in the similarly defined class NP[$\mathcal{R}$] from [18, Def. 14] when $\mathcal{R}$ is a commutative semiring. Below (Proposition 35), we will actually show that this containment is proper, in the sense that NP[$\mathcal{R}$] will contain some series that are not in $\mathsf{NP}[\mathcal{S}]$.

▶ **Example 6.** Following [10, Prop. 5.3] and [34, Examples 4.2–4.6], we can list some prominent instances of $\mathsf{NP}[\mathcal{S}]$:

- the usual complexity class NP, obtained when $\mathcal{S} = \mathbb{B}$ is the two-element Boolean semiring and each transition is weighted by 1 (this is the standard way of representing a classical machine model in the weighted context),
- the counting class #P [47], obtained when $\mathcal{S} = \langle \mathbb{N}, +, \cdot, 0, 1\rangle$ is the semiring of natural numbers and each transition is weighted by 1,

- the complexity class $\bigoplus$P [40], obtained when $\mathcal{S} = \langle \mathbb{Z}_2, +_2, \cdot_2, \overline{0}, \overline{1} \rangle$ is the finite field of two elements and each transition is weighted by 1,
- the class GapP, closure of #P under subtraction [20, 28], obtained when $\mathcal{S} = \langle \mathbb{Z}, +, \cdot, \overline{0}, \overline{1} \rangle$ is the ring of integers and transitions are weighted by 1 and -1,
- the class $\mathsf{MOD}_q - \mathsf{P}$ (for $q \geq 2$) [8], defined similarly to #P but with respect to counting modulo q, obtained when $\mathcal{S} = \langle \mathbb{Z}_q, +_q, \cdot_q, \overline{0}, \overline{1} \rangle$ and transitions are weighted by 1.

▶ **Example 7.** Some further instances of NP[$\mathcal{S}$], this time following [34, Examples 4.7–4.11], are:

- the class NP[F_*] of all fuzzy languages realizable by fuzzy Turing machines [49] with t-norm $*$ in polynomial time, obtained when the semiring is $\mathcal{F}_* = \langle [0,1], \max, *, 0, 1 \rangle$ and the weights correspond to degrees of membership in the fuzzy language,
- the class NPMV of all multivalued functions realized by nondeterministic polynomial-time transducer machines [7], obtained when, given alphabets Σ_1 and Σ_2, the semiring is $\langle 2^{\Sigma_2^*}_{\text{fin}}, \cup, \cdot, \emptyset, \{\varepsilon\} \rangle$ and weighted Turing machines have input alphabet Σ_1,
- the class of all multiset-valued functions computed by nondeterministic polynomial-time transducer machines with counting, obtained as in the previous example but using the free semiring $\langle \mathbb{N}\langle \Sigma_2^* \rangle, +, \cdot, 0, 1 \rangle$ instead,
- the class MaxP $\subseteq$ OptP of problems in which the objective is to compute the value of a solution to an optimization problem in NPO [35], obtained when the semiring is $\mathcal{S}_{\max}$, and the class MinP $\subseteq$ OptP, obtained when the semiring is $\mathcal{S}_{\min}$,
- the class MaxP$[[O(\log n)]] \subseteq$ OptP$[O(\log n)]$ of problems in which the objective is to compute the value of a solution to an optimization problem in NPO PB [35], obtained when the semiring is $\mathbb{N}_{\max}$, and MinP$[[O(\log n)]] \subseteq$ OptP$[O(\log n)]$, , obtained when the semiring is $\mathbb{N}_{\min}$.

A notion from universal algebra (cf. [5]) that we will make use of in defining some of the complexity classes below (e.g. FP[$\mathcal{S}$], FPSPACE[$\mathcal{S}$] and FPLOG[$\mathcal{S}$]) is the following:

▶ **Definition 8** (Term algebra). *Consider a semiring $\mathcal{S} = \langle S, +, \cdot, \mathbb{0}, \mathbb{1} \rangle$ and a subset $X \subseteq S$. The* set of terms *$T(X)$ is the collection of all well-formed strings that can be constructed using the symbols in X and $+', \cdot', \mathbb{0}, \mathbb{1}$ (in particular, $\mathbb{0}, \mathbb{1} \in T(X)$), that is, the smallest set such that: (1) $X \subseteq T(X)$ and (2) $(t_1 +' t_2) \in T(X)$ and $(t_1 \cdot' t_2) \in T(X)$ for every two terms $t_1, t_2 \in T(X)$; we abuse notation and omit parentheses whenever associativity permits. The* term algebra *$\mathcal{T}(X)$ is the structure with universe $T(X)$ and operations $+', \cdot'$ defined in the obvious way.*

Recall that classically FP is the set of function problems that can be solved by a deterministic Turing machine in polynomial time.

▶ **Definition 9.** *We define the complexity class* FP[$\mathcal{S}$] *as*

$$\mathsf{FP}[\mathcal{S}] := \bigcup_{\substack{\{0,1\} \subseteq G \subseteq_{\text{fin}} S \\ \Sigma \textit{ is a finite alphabet}}} \mathsf{FP}[G, \Sigma]$$

where FP[G, Σ] *is the set of all series $\sigma \colon \Sigma^* \longrightarrow \langle G \rangle$ (where $\langle G \rangle$ is the subsemiring of $\mathcal{S}$ generated by G) such that there is a constant $k \in \mathbb{N}$ and a deterministic polynomial-time Turing machine which outputs for every word $w \in \Sigma^*$ a word of the form $\sum_{i_1=1}^{m_1} \prod_{j_1=1}^{n_1} \cdots \sum_{i_k=1}^{m_k} \prod_{j_k=1}^{n_k} s_{i_1 j_1 \cdots i_k j_k}$ in the algebra of terms $T(G)$ in S with value $\sigma(w)$ in $\mathcal{S}$.*

In Definition 9, we employ a classical deterministic Turing machine which outputs, in each transition, symbols from $G \cup \{(,), +', \cdot', \mathbb{0}, \mathbb{1}\}$ or a blank. Thus, for our outputs we could obtain arbitrarily complex expressions. Therefore, the constant k limiting the number of alternations of sums and products is a proper restriction. Hence this definition of $\mathsf{FP}[\mathcal{S}]$ differs from the one of [34]. Later on, we will model logical formulas with alternating sum and product quantifiers using Turing machines which compute functions in $\mathsf{FP}[\mathcal{S}]$, hence $k = 1$ would be insufficient to model these alternations. For the converse, in order to model these Turing machines by formulas, the number of alternations of sums and products in each such Turing machine needs to be bounded to obtain a formula with nested quantifiers.

▶ **Example 10.** If $\mathcal{S} = \mathbb{B}$ is the two-element Boolean semiring, then $\mathsf{FP}[\mathbb{B}]$ is just P [34, Example 5.4]. Observe that the terms output by the machine in that example are already trivially of the form $\sum_{i=1}^{n} \prod_{j=1}^{m} s_{ij}$.

FP is to $\#\mathsf{P}$ what P is to NP. Thus, considering $\mathsf{NP}[\mathcal{S}]$ as a generalization of $\#\mathsf{P}$ (as it is done in [10]), the relationship between $\mathsf{FP}[\mathcal{S}]$ and $\mathsf{NP}[\mathcal{S}]$ is similar to that between P and NP.

▶ **Example 11.** If $\mathcal{S} = \mathbb{N}$ is the natural numbers semiring, then $\mathsf{FP}[\mathbb{N}]$ is just FP [34, Example 5.5]. As before, observe that the terms output by the machine in that example are already of the form $\sum_{i=1}^{n} \prod_{j=1}^{m} s_{ij}$.

▶ **Definition 12.** *The class* $\mathsf{FPLOG}[\mathcal{S}]$ *is defined as* $\mathsf{FP}[\mathcal{S}]$ *except that we allow the machine to have logarithmic space on the length of the input rather than polynomial time.*

▶ **Example 13.** If $\mathcal{S} = \mathbb{B}$, then $\mathsf{FPLOG}[\mathbb{B}]$ is just $\mathsf{DLOGSPACE}$.

▶ **Example 14.** If $\mathcal{S} = \mathbb{N}$, then $\mathsf{FPLOG}[\mathbb{N}]$ is just FPLOG, which is defined as FP but allowing the machine to use logarithmic space on the size of the input (cf. [22]).

▶ **Definition 15.** *The class* $\mathsf{FPSPACE}[\mathcal{S}]$ *is defined as* $\mathsf{FP}[\mathcal{S}]$ *except that we allow the machine to have polynomial space on the length of the input rather than polynomial time.*

▶ **Example 16.** If $\mathcal{S} = \mathbb{B}$, then $\mathsf{FPSPACE}[\mathbb{B}]$ is just PSPACE.

▶ **Example 17.** If $\mathcal{S} = \mathbb{N}$, then $\mathsf{FPSPACE}[\mathbb{N}]$ is just $\mathsf{FPSPACE}$ ([37]).

▶ **Definition 18.** *The class* $\mathsf{FPSPACE}_{poly}[\mathcal{S}]$ *is defined as* $\mathsf{FPSPACE}[\mathcal{S}]$ *except that we require the word* $\sum_{i=1}^{n} \prod_{j=1}^{m} s_{ij}$ *to have length bounded by a polynomial. Here, every semiring element is considered to have length* 1.

▶ **Example 19.** If $\mathcal{S} = \mathbb{B}$, then $\mathsf{FPSPACE}_{poly}[\mathbb{B}]$ is just PSPACE.

▶ **Example 20.** If $\mathcal{S} = \mathbb{N}$, then $\mathsf{FPSPACE}_{poly}[\mathbb{N}]$ is just $\mathsf{FPSPACE}_{poly}$ ([37]).

4 Weighted logics

A *vocabulary* (or *signature*) τ is a pair $\langle \mathrm{Rel}_\tau, \mathrm{ar}_\tau \rangle$ where Rel_τ is a set of relation symbols and $\mathrm{ar}_\tau \colon \mathrm{Rel}_\tau \longrightarrow \mathbb{N}_+$ is the arity function. A τ*-structure* $\mathfrak{A}$ is a pair $\langle A, \mathcal{I}_\mathfrak{A} \rangle$ where A is a set, called the *universe of* $\mathfrak{A}$, and $\mathcal{I}_\mathfrak{A}$ is an *interpretation*, which maps every symbol $R \in \mathrm{Rel}_\tau$ to a set $R^\mathfrak{A} \subseteq A^{\mathrm{ar}_\tau(R)}$. We assume that each structure is *finite*, that is, its universe is a finite set. A structure is called *ordered* if it is given for a vocabulary $\tau \cup \{<\}$ where $<$ is interpreted as a linear ordering with endpoints. By $\mathrm{Str}(\tau)_<$ we denote the class of all finite ordered τ-structures.

We provide a countable set $\mathcal{V}$ of first and second-order variables, where lower case letters like x and y denote first-order variables and capital letters like X and Y denote second-order variables. Each second-order variable X comes with an associated arity, denoted by $\mathrm{ar}(X)$. We define first-order formulas β over a signature τ and weighted first-order formulas φ over τ and a semiring $\mathcal{S}$, respectively, by the grammars

$$\beta ::= \texttt{false} \mid R(x_1, \dots, x_n) \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta$$
$$\varphi ::= \beta \mid s \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus x.\varphi \mid \bigotimes x.\varphi,$$

where $R \in \mathrm{Rel}_\tau$, $n = \mathrm{ar}_\tau(R)$, $x, x_1, \dots, x_n \in \mathcal{V}$ are first-order variables, and $s \in S$. Likewise, we define second-order formulas β over τ and weighted second-order formulas φ over τ and $\mathcal{S}$ through

$$\beta ::= \texttt{false} \mid R(x_1, \dots, x_n) \mid X(x_1, \dots, x_n) \mid \neg\beta \mid \beta \vee \beta \mid \exists x.\beta \mid \exists X.\beta$$
$$\varphi ::= \beta \mid s \mid \varphi \oplus \varphi \mid \varphi \otimes \varphi \mid \bigoplus x.\varphi \mid \bigotimes x.\varphi \mid \bigoplus X.\varphi \mid \bigotimes X.\varphi,$$

with $R \in \mathrm{Rel}_\tau$, $n = \mathrm{ar}_\tau(R) = \mathrm{ar}(X)$, $x, x_1, \dots, x_n \in \mathcal{V}$ first-order variables, $X \in \mathcal{V}$ a second-order variable, and $s \in S$. We also allow the usual abbreviations $\wedge$, $\forall$, $\rightarrow$, $\leftrightarrow$, and $\texttt{true}$. By $\mathrm{FO}(\tau)$ and $\mathrm{wFO}(\tau, S)$ we denote the sets of all first-order formulas over τ and all weighted first-order formulas over τ and $\mathcal{S}$, respectively, and by $\mathrm{SO}(\tau)$ and $\mathrm{wSO}(\tau, S)$ we denote the sets of all second-order formulas over τ and all weighted second-order formulas over τ and $\mathcal{S}$, respectively.

The notion of *free variables* is defined as usual, i.e., the operators $\exists, \forall, \bigoplus$, and $\bigotimes$ bind variables. We let $\mathrm{Free}(\varphi)$ be the set of all free variables of φ. A formula φ with $\mathrm{Free}(\varphi) = \emptyset$ is called a *sentence*. For a tuple $\bar{\varphi} = \langle \varphi_1, \dots, \varphi_n \rangle \in \mathrm{wSO}(\tau, S)^n$, we define $\mathrm{Free}(\bar{\varphi}) = \bigcup_{i=1}^n \mathrm{Free}(\varphi_i)$.

We define the semantics of SO and wSO as follows. Let τ be a signature, $\mathfrak{A} = \langle A, \mathcal{I}_\mathfrak{A} \rangle$ a τ-structure, and $\mathcal{V}$ a set of first and second-order variables. A $(\mathcal{V}, \mathfrak{A})$-assignment ρ is a function $\rho\colon \mathcal{V} \longrightarrow A \cup \mathcal{P}(A)$ such that, whenever $x \in \mathcal{V}$ is a first-order variable and $\rho(x)$ is defined, we have $\rho(x) \in A$, and whenever $X \in \mathcal{V}$ is a second-order variable and $\rho(X)$ is defined, we have $\rho(X) \subseteq A^{\mathrm{ar}(X)}$. For a first-order variable, this restriction may cause the variable to become undefined. Let $\mathrm{dom}(\rho)$ be the domain of ρ. For a first-order variable $x \in \mathcal{V}$ and an element $a \in A$, the *update* $\rho[x \to a]$ is defined through $\mathrm{dom}(\rho[x \to a]) = \mathrm{dom}(\rho) \cup \{x\}$, $\rho[x \to a](\mathcal{X}) = \rho(\mathcal{X})$ for all $\mathcal{X} \in \mathcal{V} \setminus \{x\}$, and $\rho[x \to a](x) = a$. For a second-order variable $X \in \mathcal{V}$ and a set $I \subseteq A^{\mathrm{ar}(X)}$, the update $\rho[X \to I]$ is defined in a similar fashion. By $\mathfrak{A}_\mathcal{V}$ we denote the set of all $(\mathcal{V}, \mathfrak{A})$-assignments.

For $\rho \in \mathfrak{A}_\mathcal{V}$ and a formula $\beta \in \mathrm{SO}(\tau)$ the relation "$\langle \mathfrak{A}, \rho \rangle$ satisfies β", denoted by $\langle \mathfrak{A}, \rho \rangle \models \beta$, is defined as

$$\begin{aligned}
&\langle \mathfrak{A}, \rho \rangle \models \texttt{false} && && \text{never holds}\\
&\langle \mathfrak{A}, \rho \rangle \models R(x_1, \dots, x_n) && \iff && x_1, \dots, x_n \in \mathrm{dom}(\rho) \text{ and } (\rho(x_1), \dots, \rho(x_n)) \in R^\mathfrak{A}\\
&\langle \mathfrak{A}, \rho \rangle \models X(x_1, ..., x_n) && \iff && x_1, ..., x_n, X \in \mathrm{dom}(\rho) \text{ and } \langle \rho(x_1), \dots, \rho(x_n) \rangle \in \rho(X)\\
&\langle \mathfrak{A}, \rho \rangle \models \neg\beta && \iff && \langle \mathfrak{A}, \rho \rangle \models \beta \text{ does not hold}\\
&\langle \mathfrak{A}, \rho \rangle \models \beta_1 \vee \beta_2 && \iff && \langle \mathfrak{A}, \rho \rangle \models \beta_1 \text{ or } \langle \mathfrak{A}, \rho \rangle \models \beta_2\\
&\langle \mathfrak{A}, \rho \rangle \models \exists x.\beta && \iff && \langle \mathfrak{A}, \rho[x \to a] \rangle \models \beta \text{ for some } a \in A\\
&\langle \mathfrak{A}, \rho \rangle \models \exists X.\beta && \iff && \langle \mathfrak{A}, \rho[X \to I] \rangle \models \beta \text{ for some } I \subseteq A.
\end{aligned}$$

Let $\varphi \in \mathrm{wSO}(\tau, S)$ and $\mathfrak{A} \in \mathrm{Str}(\tau)_<$, $a_1, \dots, a_k$ be an enumeration of the elements of $\mathfrak{A}$ according to the ordering that serves as the interpretation of $<$, and for every integer n, let $I_1^n, \dots, I_{l_n}^n$ be an enumeration of the subsets of A^n according to the lexicographic ordering induced by the interpretation of $<$. The *(weighted) semantics* of φ is a mapping $[\![\varphi]\!](\mathfrak{A}, \cdot)\colon \mathfrak{A}_{\mathcal{V}} \longrightarrow S$ inductively defined as

$$\begin{aligned}
[\![\beta]\!](\mathfrak{A},\rho) &= \begin{cases} \mathbb{1} & \text{if } \langle \mathfrak{A},\rho\rangle \models \beta \\ \mathbb{0} & \text{otherwise} \end{cases} \\
[\![s]\!](\mathfrak{A},\rho) &= s \\
[\![\varphi_1 \oplus \varphi_2]\!](\mathfrak{A},\rho) &= [\![\varphi_1]\!](\mathfrak{A},\rho) + [\![\varphi_2]\!](\mathfrak{A},\rho) \\
[\![\varphi_1 \otimes \varphi_2]\!](\mathfrak{A},\rho) &= [\![\varphi_1]\!](\mathfrak{A},\rho) \cdot [\![\varphi_2]\!](\mathfrak{A},\rho) \\
[\![\textstyle\bigoplus x.\varphi]\!](\mathfrak{A},\rho) &= \sum_{a \in A} [\![\varphi]\!](\mathfrak{A},\rho[x \to a]) \\
[\![\textstyle\bigotimes x.\varphi]\!](\mathfrak{A},\rho) &= \prod_{1 \leq i \leq k} [\![\varphi]\!](\mathfrak{A},\rho[x \to a_i]) \\
[\![\textstyle\bigoplus X.\varphi]\!](\mathfrak{A},\rho) &= \sum_{I \subseteq A^{\mathrm{ar}(X)}} [\![\varphi]\!](\mathfrak{A},\rho[X \to I]) \\
[\![\textstyle\bigotimes X.\varphi]\!](\mathfrak{A},\rho) &= \prod_{1 \leq i \leq l_{\mathrm{ar}(X)}} [\![\varphi]\!](\mathfrak{A},\rho[X \to I_i^{\mathrm{ar}(X)}]).
\end{aligned}$$

Thanks to the lexicografic ordering our product quantifiers have a well-defined semantics. Note that if the semiring is commutative, in the clauses of universal quantifiers, the semantics is defined by using any order for the factors in the products.

We will usually identify a pair $\langle \mathfrak{A}, \emptyset \rangle$ (where $\emptyset$ is the empty mapping) with $\mathfrak{A}$. We will also refer to the following expansions of FO:

- Transitive closure logic (TC) is obtained by adding the following rule for building formulas: if $\varphi(\overline{x}, \overline{y})$ is a formula with variables $\overline{x} = x_1, \dots, x_k$ and $\overline{y} = y_1, \dots, y_k$, and $\overline{u}, \overline{v}$ are k-tuples of terms, then $[\mathbf{tc}_{\overline{x},\overline{y}}\, \varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$ is also a formula, and its semantics is given as $\mathfrak{A} \models [\mathbf{tc}_{\overline{x},\overline{y}}\, \varphi(\overline{x},\overline{y})](\overline{a},\overline{b}) \iff$ there exist an $n \geq 1$ and $\overline{c_0}, \dots, \overline{c_n} \in A^k$ such that $\overline{c_0} = \overline{a}$, $\overline{c_n} = \overline{b}$, and $\mathfrak{A} \models \varphi(\overline{c_i}, \overline{c_{i+1}})$ for each $i \in \{0, \dots, n-1\}$.
- Deterministic transitive closure logic (DTC) is obtained by adding the following rule for building formulas: if $\varphi(\overline{x}, \overline{y})$ is a formula with variables $\overline{x} = x_1, \dots, x_k$ and $\overline{y} = y_1, \dots, y_k$, and $\overline{u}, \overline{v}$ are k-tuples of terms, then $[\mathbf{dtc}_{\overline{x},\overline{y}}\, \varphi(\overline{x},\overline{y})](\overline{u},\overline{v})$ is also a formula, and its semantics is defined by the equivalence $[\mathbf{dtc}_{\overline{x},\overline{y}}\, \varphi(\overline{x},\overline{y})](\overline{u},\overline{v}) \equiv [\mathbf{tc}_{\overline{x},\overline{y}}\, \varphi(\overline{x},\overline{y}) \wedge \forall z(\varphi(\overline{x},\overline{z}) \to \overline{y} = \overline{z})](\overline{u},\overline{v})$.
- Least fixed-point logic (LFP) is obtained by adding the following rules for building formulas: if $\varphi(R, \overline{x})$ is a formula of vocabulary $\tau \cup \{R\}$ with only positive occurrences of R, $\overline{x}$ is a tuple of variables, and $\overline{t}$ is a tuple of terms (both matching the arity of R), then $[\mathbf{lfp}\, R\overline{x}.\psi](\overline{t})$ and $[\mathbf{gfp}\, R\overline{x}.\psi](\overline{t})$ are also formulas. For their semantics, we need to define some auxiliary notions. The *update operator* $F_\psi\colon \mathcal{P}(A^k) \longrightarrow \mathcal{P}(A^k)$ is defined by $F_\psi(R) := \{\overline{a} \mid \langle \mathfrak{A}, R\rangle \models \psi(R, \overline{a})\}$ for any relation R, and it is monotone because R occurs only positively in ψ. A *fixed point* of F_ψ is a relation R such that $F_\psi(R) = R$. Since F_ψ is monotone, it has a least and a greatest fixed point (by Knaster–Tarski Theorem). The semantics is given by: $\mathfrak{A} \models [\mathbf{lfp}\, R\overline{x}.\psi](\overline{t})$ iff $\overline{t}^{\mathfrak{A}}$ is contained in the least fixed point of F_ψ (analogously for $[\mathbf{gfp}\, R\overline{x}.\psi](\overline{t})$ and the greatest fixed point).
- Partial fixed-point logic (PFP) is obtained by adding the following rule for building formulas: if $\varphi(R, \overline{x})$ is a formula of vocabulary $\tau \cup \{R\}$, $\overline{x}$ is a tuple of variables, and $\overline{t}$ is a tuple of terms (both matching the arity of R), then $[\mathbf{pfp}\, R\overline{x}.\psi](\overline{t})$ is also a formula. For

the semantics, we consider again the update operator (now not necessarily monotone) and the sequence of its finite stages: $R^0 := \emptyset$ and $R^{m+1} := F_\psi(R^m)$. In a finite structure $\mathfrak{A}$, the sequence either reaches a fixed point or it enters a cycle of period greater than one. We define the partial fixed point of F_ψ as the fixed point reached in the former case, or as the empty set in the latter case. Now, the semantics is given by: $\mathfrak{A} \models [\mathbf{pfp}\, R\overline{x}.\psi](\overline{t})$ iff $\overline{t}^{\mathfrak{A}}$ is contained in the partial fixed point of F_ψ.

- Inflationary fixed-point logic (IFP) is obtained by adding the following rules for building formulas: if $\varphi(R, \overline{x})$ is a formula of vocabulary $\tau \cup \{R\}$, $\overline{x}$ is a tuple of variables, and $\overline{t}$ is a tuple of terms (both matching the arity of R), then $[\mathbf{ifp}\, R\overline{x}.\psi](\overline{t})$ is also a formula. For its semantics, we need to define some auxiliary notions. An operator $G : \mathcal{P}(B) \longrightarrow \mathcal{P}(B)$ is said to be *inflationary* if $X \subseteq G(X)$ for all $X \in \mathcal{P}(B)$. With any operator $F : \mathcal{P}(B) \longrightarrow \mathcal{P}(B)$ one can associate an inflationary operator G by setting $G(X) := X \cup F(X)$. Iterating G gives a fixed point that we will called the *inflationary fixed point* of F. The semantics is given by: $\mathfrak{A} \models [\mathbf{ifp}\, R\overline{x}.\psi](\overline{t})$ iff $\overline{t}^{\mathfrak{A}}$ is contained in the inflationary fixed point of F_ψ.

The weighted version of each of these logics is defined analogously as in the case of FO and SO by expanding the logics TC, DTC, LFP, PFP, and IFP with the same weighted constructs as given for wFO and wSO. By a famous result of Gurevich and Shelah [29], on finite structures, LFP coincides with IFP and thus their weighted versions, wIFP and wLFP, as we have defined them here, will also coincide in expressive power.

5 Logical characterizations of complexity classes

We are finally ready to present and prove the main results of the paper: the quantitative versions of several logical characterizations of prominent complexity classes. We may assume that every $\mathfrak{A} \in \mathrm{Str}(\tau)_<$ is encoded by a string of 0s and 1s. For example, where $\mathfrak{A} = \langle A, R_1^{\mathfrak{A}}, \ldots, R_j^{\mathfrak{A}} \rangle$ with $|A| = n$ (and we may assume in fact that $A = \{0, \ldots, n-1\}$) we might let

$$\mathrm{enc}(\mathfrak{A}) = \mathrm{enc}(R_1^{\mathfrak{A}}) \cdot \cdots \cdot \mathrm{enc}(R_j^{\mathfrak{A}})$$

where if $R_i^{\mathfrak{A}}$ is an l-ary relation, then $\mathrm{enc}(R_i^{\mathfrak{A}})$ is a string of symbols of length n^l with a 1 in its mth position if the mth tuple of n^l is in $R_i^{\mathfrak{A}}$ and a 0 otherwise.

▶ **Definition 21.** *Consider a weighted logic* $\mathrm{L}[\mathcal{S}]$ *(with weights in a semiring* $\mathcal{S}$*) and a weighted complexity class* $\mathcal{C}$*, which is simply a collection of series. We say that* $\mathrm{L}[\mathcal{S}]$ captures $\mathcal{C}$ *over ordered structures in the vocabulary* $\tau = \{R_1, \ldots, R_j\}$ *if:*

(1) *For every* $\mathrm{L}[\mathcal{S}]$*-formula* ϕ*, there exists* $P \in \mathcal{C}$ *such that* $P(enc(\mathfrak{A})) = \|\phi\|(\mathfrak{A})$ *for every finite ordered* τ*-structure* $\mathfrak{A}$*, and*

(2) *For every* $P \in \mathcal{C}$*, there exists an* $\mathrm{L}[\mathcal{S}]$*-formula* ϕ *such that* $P(enc(\mathfrak{A})) = \|\phi\|(\mathfrak{A})$ *for every finite ordered* τ*-structure* $\mathfrak{A}$*.*

The seminal Fagin's Theorem characterizes NP for ordered structures by existential second-order logic. Our goal is to present a weighted version of this result with arbitrary semirings as weight structures. Whereas in the classical setting one obtains an equivalence between the existence of runs of a Turing machine vs. the satisfiability of an existential logical formula, in the weighted setting of general semirings we have to derive a one-to-one correspondence between the runs of a Turing machine and satisfying assignments for the formulas. Moreover, due to the absence of a natural negation function in the semiring, here, beyond the classical setting, we need conjunctions and universal quantifications. For weighted finite automata over words, in [11] weighted conjunction and universal quantification turned out to be too powerful in general and had to be restricted. Surprisingly, here we do not need these restrictions, but we can show the expressive equivalence between weighted polynomial-time Turing machines and the full weighted existential second-order logic. Moreover, we do not need commutativity of the multiplication of $\mathcal{S}$ (essential in [11]), but can develop our characterization for arbitrary, also non-commutative, semirings $\mathcal{S}$. This is due to new constructions, in this setting, for the involved weighted Turing machines. By wESO we mean the fragment of wSO where the only second-order quantifiers appear at the beginning of the formula and are additive existential.

▶ **Theorem 22** (Weighted Fagin's theorem). *Let $\mathcal{S}$ be a semiring and τ a vocabulary.*

(i) *The logic* wESO$[\mathcal{S}]$ *captures* $\mathsf{NP}[\mathcal{S}]$ *over* ordered *finite τ-structures.*

(ii) *Assume that $\mathcal{S}$ is idempotent and commutative. Then, the logic* wESO$[\mathcal{S}]$ *captures* $\mathsf{NP}[\mathcal{S}]$ *over* all *finite structures in the vocabulary τ.*

Let us indicate some ideas for the proof. For (i), first, for a given wESO-formula ϕ, we have to construct an NP Turing machine $\mathcal{M}$ with $\|\phi\| = \|\mathcal{M}\|$. For Boolean formulas β, we can follow the classical proof. Regarding weighted formulas ϕ, let us comment on the interesting cases. For weighted conjunctions and universal quantifications, we employ new constructions. Since we are dealing with Turing machines, we can execute weighted Turing machines for the components successively, by saving the word and using transitions of weight 1 in a deterministic way to restore the initial tape configuration. We can show, using the distributivity of the semiring, that the constructed nondeterministic machine $\mathcal{M}$ computes precisely the values prescribed by the semantics of the weighted conjunction or the weighted universal quantifications, respectively.

Second, given a weighted NP Turing machine $\mathcal{M}$, by the assumption on its polynomial time usage, we construct a second-order formula ψ reflecting the accepting computation paths of $\mathcal{M}$ and their employed transitions in a one-to-one correspondence; this enables us to incorporate the weights of the transitions by means of constants in the formula. The order is used for the construction of the formula such that the interpretation of weighted universal quantification reflects precisely the weights of the computation sequences of the given Turing machine.

For (ii), the order in universal quantifications now is taken care of by the commutativity of the multiplication, and the existence of an order is taken care of by an additional existential second-order quantification where idempotency of $\mathcal{S}$ implies that we obtain the same value.

From Theorem 22 and Examples 6 and 7, we immediately obtain the following corollary:

▶ **Corollary 23.** *For ordered structures in a finite vocabulary τ, we have that:*

(1) wESO$[\mathbb{B}]$ captures NP *(originally proved in [19]).*

(2) wESO$[\mathbb{N}]$ captures $\#\mathsf{P}$ *(originally proved in [3] and [44]).*

(3) wESO$[\mathbb{Z}]$ captures GapP *(originally proved in [3]).*

(4) wESO$[\mathcal{S}_{\max}]$ *(respectively,* wESO$[\mathcal{S}_{\min}]$*).* captures MaxP *(*MinP*) (originally proved in [3]).*

(5) wESO$[\mathbb{Z}_2]$ captures $\bigoplus\mathsf{P}$.

(6) wESO$[\mathbb{Z}_q]$ captures $\mathsf{MOD}_q - \mathsf{P}$.

(7) wESO$[\mathbb{N}_{\max}]$ *(respectively,* wESO$[\mathbb{N}_{\min}]$*)* captures $\mathsf{MaxP}[O(\log n)]$ *(*$\mathsf{MinP}[O(\log n)]$*).*

(8) wESO$[\mathcal{F}_*]$ captures *the class of all fuzzy languages realizable by fuzzy Turing machines with t-norm $*$ in polynomial time.*

(9) wESO$[2^{\Sigma_2^*}_{\text{fin}}]$ captures NPMV.

(10) wESO$[\mathbb{N}\langle\Sigma_2^*\rangle]$ captures *the class of all multiset-valued functions computed by non-deterministic polynomial-time transducer machines with counting.*

▶ Remark 24. It is worth observing that the proofs of (2)-(4) in [3] (Prop. 4.2, Cor. 4.8, and Thm. 4.10) are (as expected) different from ours. Our argument works in all those cases but neither of the three arguments given in [3] works for our more general setting.

Our next application of the weighted Fagin's theorem consist in providing a natural computational problem complete for the class $\mathsf{NP}[\mathcal{S}]$ for certain semirings $\mathcal{S}$. Given a semiring $\mathcal{S}$, alphabets Σ_1, Σ_2, and series $\sigma_1 : \Sigma_1^* \longrightarrow S$ and $\sigma_2 : \Sigma_2^* \longrightarrow S$, we say that σ_1 is *polynomially many-one reducible* to σ_2 ($\sigma_1 \leq_m \sigma_2$, in symbols) if there is an $f\colon \Sigma_1^* \longrightarrow \Sigma_2^*$ computable deterministically in polynomial time such that $\sigma_2(f(w)) = \sigma_1(w)$ for each $w \in \Sigma_1^*$. A series $\sigma : \Sigma^* \longrightarrow S$ is said to be $\mathsf{NP}[\mathcal{S}]$*-hard* if $\sigma' \leq_m \sigma$ for all σ' in $\mathsf{NP}[\mathcal{S}]$. If, moreover, σ belongs to $\mathsf{NP}[\mathcal{S}]$, then it is called $\mathsf{NP}[\mathcal{S}]$*-complete.*

Fix an infinite set X. The language of the weighted propositional logic over a finitely generated semiring $\mathcal{S}$ is built from X as propositional variables, elements of S as truth-constants, and logical connectives $\wedge, \vee, \neg$ (where negation is only applied to propositional variables). Let $\mathtt{Fmla}[\mathcal{S}]$ be the set of all formulas.

A *truth assignment* is a mapping $V\colon X \longrightarrow \{0,1\}$ extended to $\overline{V}$ for all formulas in the following way:

1. For each propositional variable $x \in X$, let $\overline{V}(x) := V(x)$ and $\overline{V}(\neg x) := 1$ iff $V(x) = 0$. Moreover, let $\overline{V}(a) := a$ for each $a \in S$.
2. $\overline{V}(\varphi \vee \psi) := \overline{V}(\varphi) + \overline{V}(\psi)$ and $\overline{V}(\varphi \wedge \psi) := \overline{V}(\varphi) \cdot \overline{V}(\psi)$.

For each formula $\varphi \in \mathtt{Fmla}[\mathcal{S}]$, let X_φ be the set of propositional variables that occur in φ. Clearly, $\overline{V}(\varphi)$ depends only the values of V on X_φ. The "problem" $\mathrm{SAT}[\mathcal{S}]$ is the series $\sigma\colon \mathtt{Fmla}[\mathcal{S}] \longrightarrow S$ defined as follows: $\mathrm{SAT}[\mathcal{S}](\varphi) = \sum_{V \in \{0,1\}^{X_\varphi}} \overline{V}(\varphi)$.

The following corollary of our weighted version of Fagin's theorem has also appeared as [34, Thm. 6.3] with a direct proof. Our proof generalizes the reasoning for the Boolean case in [25].

▶ **Corollary 25** (Weighted Cook–Levin's theorem). *Let $\mathcal{S}$ be a finitely generated semiring. Then,* $\mathsf{SAT}[\mathcal{S}]$ *is* $\mathsf{NP}[\mathcal{S}]$*-complete.*

Now it is natural to wonder what happens with other well-known descriptive complexity results. In the reminder of this section we will tackle a few more of these. We start with the Immerman–Vardi's theorem, a result that first appeared in the Boolean case in the papers [31, 48]. Our own approach is inspired by [3, Thm. 4.4] where a version of the result for the counting complexity class FP is provided using a weighted logic with the semiring $\mathbb{N}$. We must observe, however, that our proof is a generalization of that in [3] that works for all semirings and not only $\mathbb{N}$.

▶ **Theorem 26** (Weighted Immerman–Vardi's theorem). *The logic* $\mathrm{wLFP}[\mathcal{S}]$ *(with weights in a semiring $\mathcal{S}$)* captures $\mathsf{FP}[\mathcal{S}]$ *over ordered structures in the vocabulary τ.*

▶ **Corollary 27.** *For ordered structures in a finite vocabulary τ, we have that:*

(1) $\mathrm{wLFP}[\mathbb{B}]$ captures P *(originally proved in [31, 48]).*
(2) $\mathrm{wLFP}[\mathbb{N}]$ captures FP *(originally proved in [3]).*

▶ Remark 28. Observe that using second-order Horn logic (which is known to capture P [24]) instead of least fixed-point logic, would not work for us, as in the weighted version one can encode a #P-complete problem (namely #HORNSAT). This was already noted in [3].

In the next result, $\mathrm{wPFP}[\mathcal{S}] + \{\prod X, \sum X\}$ will denote the logic that is obtained from $\mathrm{wPFP}[\mathcal{S}]$ by the addition of the second-order quantitative quantifiers $\prod X$ and $\sum X$. Clearly, when $\mathcal{S} = \mathbb{B}$, this is the same as second-order logic with partial fixed points. The Boolean counterpart of Theorem 29, namely that second-order logic extended with partial fixed points characterizes PSPACE is folklore, but a proof can be found in [41, Thm. 4]. The classical argument also uses the result for partial fixed-point logic in [1, 48] stating that the logic characterizes PSPACE over ordered structures.

▶ **Theorem 29.** *The logic* $\mathrm{wPFP}[\mathcal{S}] + \{\prod X, \sum X\}$ *(with weights in a semiring* $\mathcal{S}$*)* captures $\mathsf{FPSPACE}[\mathcal{S}]$ *over ordered structures in the vocabulary* τ*.*

▶ **Corollary 30.** *For ordered structures in a finite vocabulary* τ*, we have that:*

(1) $\mathrm{wPFP}[\mathbb{B}] + \{\prod X, \sum X\}$ captures PSPACE *(folklore, cf. [41]).*

(2) $\mathrm{wPFP}[\mathbb{N}] + \{\prod X, \sum X\}$ captures FPSPACE *(originally proved in [3]).*

▶ **Theorem 31.** *The logic* $\mathrm{wPFP}[\mathcal{S}]$ *(with weights in a semiring* $\mathcal{S}$*)* captures $\mathsf{FPSPACE}_{poly}[\mathcal{S}]$ *over ordered structures in the vocabulary* τ*.*

▶ **Corollary 32.** *For ordered structures in a finite vocabulary* τ*, we have that:*

(1) $\mathrm{wPFP}[\mathbb{B}]$ captures PSPACE *(originally proved in [1, 48]).*

(2) $\mathrm{wPFP}[\mathbb{N}]$ captures $\mathsf{FPSPACE}_{poly}$ *(originally proved in [3]).*

▶ **Theorem 33.** *The logic* $\mathrm{wDTC}[\mathcal{S}]$ *(with weights in a semiring* $\mathcal{S}$*)* captures $\mathsf{FPLOG}[\mathcal{S}]$ *over ordered structures in the vocabulary* τ*.*

▶ **Corollary 34.** *For ordered structures in a finite vocabulary* τ*, we have that:*

(1) $\mathrm{wDTC}[\mathbb{B}]$ captures DLOGSPACE *(originally proved in [31]).*

(2) $\mathrm{wDTC}[\mathbb{N}]$ captures FPLOG.

To end the present section, we address the general and interesting open problem suggested in [18] regarding a Fagin theorem that characterizes the class $\mathrm{NP}[\mathcal{R}]$ from [18, Def. 14]. We begin by observing that for the machine model in [18, Def. 12], Fagin's theorem will fail if the logic considered is wESO. This is essentially due to the fact that semiring Turing machines allow for arbitrary semiring values on the tape and can transition with these values. However, such a large set of transitions, is only actually needed when there are infinitely many semiring values in the input words.

▶ **Proposition 35.** *Let* $\mathcal{R}$ *be a commutative semiring. There is a series* $P \in \mathrm{NP}[\mathcal{R}]$ *such that for no* $\varphi \in \mathrm{wESO}$*,* $||\varphi|| = P$*.*

Thus one might reasonably further ask what kind of logic would capture $\mathrm{NP}[\mathcal{R}]$. Observe that an obvious challenge here is that in the proof of Fagin's theorem at some point we need to encode in the logic by means of a sentence involving a long (but finite) disjunction what the legal transitions of our machine are. Consequently, in the presence of infinitely many transitions, it is not clear how to achieve a Fagin-style characterization in a finitary language as before. Furthermore, it appears that semiring Turing machines are more suitable for an analysis that involves semirings that are not finitely generated.

By contrast to the above situation, we might ask a more restricted question if what we are doing is trying to capture $\mathrm{NP}[\mathcal{R}]$ over the class of all finite ordered structures. Recall that we are considering finite structures to be given via their binary encodings and thus the relevant series in $\mathrm{NP}[\mathcal{R}]$ are those that take as input merely binary strings. These series are not computed by SRTMs that involve infinitely many transitions because the input words do not involve semiring values. So let us consider now the modification of [18, Def. 12] that only allows semiring Turing machines to come with a finite set of transitions. In this case we will easily see that their machine model coincides with ours.

▶ **Proposition 36.** *Let $\mathcal{R}$ be a commutative semiring and allow only finitely many transitions in a semiring Turing machine. Then* $\mathrm{NP}[\mathcal{R}] = \mathsf{NP}[\mathcal{R}]$*, i.e. the NP class in the sense of [18] coincides with the NP class in our sense.*

6 Conclusions and further work

In this paper, we have established a few central results in weighted descriptive complexity, providing quantitative versions of Fagin's theorem and the Immerman–Vardi's theorem, among other logical characterizations of complexity classes. We also plan to extend our weighted Fagin's theorem to the even larger class of valuation monoids containing all semirings and supporting average calculations by the theory developed in [21] for weighted finite automata over words and weighted EMSO logic.

Furthermore, in future work, we aim to characterize further weighted complexity classes. For example, in the definition of $\mathsf{NP}[\mathcal{S}]$, by changing the requirement about polynomial time to logarithmic space on the size of the input, we can obtain a weighted complexity class that generalizes the classical counting class $\#\mathsf{L}$. The latter has been characterized by means of a logic weighted on the semiring $\mathbb{N}$ in [3, Thm. 6.4]. We suspect that this work can be generalized.

References

1 Serge Abiteboul and Victor Vianu. Fixpoint extensions of first-order logic and datalog-like languages. In *Proceedings of Fourth Annual Symposium on Logic in Computer Science*, pages 71–79. IEEE Comput. Soc. Press, 1989.

2 Jürgen Albert and Jarkko Kari. Digital image compression. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science, pages 453–479, Berlin, Heidelberg, 2009. Springer-Verlag.

3 Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. *Logical Methods in Computer Science*, 16, 2020.

4 Benjamín Callejas Bedregal and Santiago Figueira. On the computing power of fuzzy turing machines. *Fuzzy Sets and Systems*, 159:1072–1083, 2008.

5 Clifford Bergman. *Universal Algebra: Fundamentals and Selected Topics*. Chapman and Hall/CRC, 2011.

6 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Cambridge University Press, 2009.

7 Ronald V. Book, Timothy J. Long, and Alan L. Selman. Qualitative relativizations of complexity classes. *Logical Methods in Computer Science*, 30:395–413, 1985.

8 Jin-Yi Cai and Lane A. Hemachandra. On the power of parity. In *Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*, pages 229–240, Berlin, 1989. Springer-Verlag.

9 Carsten Damm, Markus Holzer, and Pierre McKenzie. The complexity of tensor calculus. In *Proceedings of the 15th Annual Conference on Computational Complexity*, Lecture Notes in Computer Science, pages 70–86, Berlin, 2000. IEEE Comput. Soc. Press.

10 Carsten Damm, Markus Holzer, and Pierre McKenzie. The complexity of tensor calculus. *Computational Complexity*, 11:54–89, 2002.

11 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380:69–86, 2007.

12 Manfred Droste, Werner Kuich, and Heiko Vogler (editors). *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, 2009.

13 Manfred Droste and Erik Paul. A feferman–vaught decomposition theorem for weighted mso logic. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of*

Computer Science, volume 76 of *Lecture Notes in Computer Science*, pages 1–15, Berlin, 2018. IEEE Comput. Soc. Press.

14 Arnaud Durand, Anselm Haak, Juha Kontinen, and Heribert Vollmer. Descriptive complexity of #p functions: A new perspective. *Journal of Computer and System Sciences*, 116:40–54, 2021.

15 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, Heidelberg, 1995.

16 Samuel Eilenberg. *Automata, Languages, and Machines.* Academic Press. Academic Press, New York and London, 1974.

17 Thomas Eiter and Rafael Kiesel. On the complexity of sum-of-products problems over semirings. In *Proceedings of the AAAI Conference on Artificial Intelligence AAAI-21*, pages 6304–6311. IEEE Comput. Soc. Press, 2021.

18 Thomas Eiter and Rafael Kiesel. Semiring reasoning frameworks in ai and their computational complexity. *Journal of Artificial Intelligence Research*, 77:207–293, 2023.

19 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

20 Stephen A. Fenner, Lance J. Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

21 Paul Gastin and Benjamin Monmege. A unifying survey on weighted logics and weighted automata - core weighted logic: minimal and versatile specification of quantitative properties. *Soft Computing*, 22:1047–1065, 2018.

22 Christian Glasser. Space-efficient informational redundancy. *Journal of Computer and System Sciences*, 76:792–811, 2010.

23 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the TwentySixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 31–40, Beijing, China, 2007. IEEE Comput. Soc. Press.

24 Erich Grädel. Capturing complexity classes by fragments of second-order logic. *Theoretical Computer Science*, 101:35–57, 1992.

25 Erich Grädel, Phokion Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications.* Springer-Verlag, 2007.

26 Erich Grädel and Val Tannen. Semiring provenance for first-order model checking, 2017. `arXiv:1712.01980`.

27 Erich Grädel and Val Tannen. Provenance analysis for logic and games. *Moscow Journal of Combinatorics and Number Theory*, 9:203–228, 2020.

28 S. Gupta. Closure properties and witness reduction. *Journal of Computer and System Sciences*, 50:412–432, 1995.

29 Yuri Gurevich and Saharon Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

30 Tero Harju and Juhani Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78:347–355, 1991.

31 Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.

32 Neil Immerman. *Descriptive Complexity.* Graduate texts in computer science. Springer, 1999.

33 Juha Kontinen. A logical characterization of the counting hierarchy. *ACM Transactions on Computational Logic*, 10:1–21, 2009.

34 Peter Kostolányi. Weighted automata and logics meet computational complexity, 2023. `arXiv:2312.10810`.

35 Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

36 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages.* Monographs in Theoretical Computer Science. Springer Verlag, 1985.

37 Richard E. Ladner. Polynomial space counting problems. *SIAM Journal on Computing*, 18:1087–1097, 1989.

38 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. Springer, 2004.

39 V. Yu. Meitus. Decidability of the equivalence problem for deterministic pushdown automata. *Cybernetics and Systems Analysis*, 5:20–45, 1992.

40 Christos H. Papadimitriou and Stathis Zachos. Two remarks on the power of counting. In *Proceedings of 6th GI Conferences in Theoretical Computer Science*, pages 269–275, 1983.

41 David Richerby. Logical characterizations of pspace. In J. Marcinkowski and A. Tarlecki, editors, *Proceedings of Computer Science Logic CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 370–384, Berlin, Heidelberg, 2004. Springer.

42 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

43 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Monographs in Computer Science. Springer, 1978.

44 Sanjeev Saluja, K. V. Subrahmanyam, and Madhukar N. Thakur. Descriptive complexity of #p functions. *Journal of Computer and System Sciences*, 50:493–505, 1995.

45 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.

46 Géraud Senizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Proceedings of International Colloquium on Automata, Languages, and Programming ICALP 1997*, volume 1256 of *Lecture Notes in Computer Science*, pages 671–681, 1997.

47 Leslie Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

48 Moshe Vardi. The complexity of relational query languages (extended abstract). In *STOC 1982 Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.

49 Jiří Wiedermann. Characterizing the super-turing computing power and efficiency of classical fuzzy turing machines,. *Theoretical Computer Science*, 317:61–69, 2004.

Breaking the Barrier 2^k for Subset Feedback Vertex Set in Chordal Graphs

Tian Bai

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

Mingyu Xiao

School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China

Abstract

The Subset Feedback Vertex Set problem (SFVS) is to delete k vertices from a given graph such that in the remaining graph, any vertex in a subset T of vertices (called a terminal set) is not in a cycle. The famous Feedback Vertex Set problem is the special case of SFVS with T being the whole set of vertices. In this paper, we study exact algorithms for SFVS in Split Graphs (SFVS-S) and SFVS in Chordal Graphs (SFVS-C). SFVS-S generalizes the minimum vertex cover problem and the prize-collecting version of the maximum independent set problem in hypergraphs (PCMIS), and SFVS-C further generalizes SFVS-S. Both SFVS-S and SFVS-C are implicit 3-Hitting Set problems. However, it is not easy to solve them faster than 3-Hitting Set. In 2019, Philip, Rajan, Saurabh, and Tale (Algorithmica 2019) proved that SFVS-C can be solved in $\mathcal{O}^*(2^k)$ time, slightly improving the best result $\mathcal{O}^*(2.0755^k)$ for 3-Hitting Set. In this paper, we break the "2^k-barrier" for SFVS-S and SFVS-C by introducing an $\mathcal{O}^*(1.8192^k)$-time algorithm. This achievement also indicates that PCMIS can be solved in $\mathcal{O}^*(1.8192^n)$ time, marking the first exact algorithm for PCMIS that outperforms the trivial $\mathcal{O}^*(2^n)$ threshold. Our algorithm uses reduction and branching rules based on the Dulmage-Mendelsohn decomposition and a divide-and-conquer method.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Subset Feedback Vertex Set, Prize-Collecting Maximum Independent Set, Parameterized Algorithms, Split Graphs, Chordal Graphs, Dulmage-Mendelsohn Decomposition

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.15

Related Version *Full Version*: https://arxiv.org/abs/2212.04726

Funding *Mingyu Xiao:* Mingyu Xiao acknowledges the National Science Foundation of China under Grant No. 62372095.

1 Introduction

The Feedback Vertex Set problem (FVS), one of Karp's 21 NP-complete problems [32], is a fundamental problem in graph algorithms. Given a graph G with n vertices and a parameter k, FVS is to decide whether there is a subset of vertices of size at most k whose deletion makes the remaining graph acyclic. FVS arises in a variety of applications in various fields such as circuit testing, network communications, deadlock resolution, artificial intelligence, and computational biology [6, 11, 29]. Because of the importance of FVS, different variants and generalizations have been extensively studied in the literature. The Subset Feedback Vertex Set problem (SFVS), introduced by Even et al. [17] in 2000, is a famous case. In SFVS, we are further given a vertex subset $T \subseteq V$ called *terminal set*, and we are asked to determine whether there is a set of vertices of size at most k whose removal makes each terminal in T not contained in any cycle in the remaining graph. When the terminal set is the whole vertex set of the graph, SFVS becomes FVS. SFVS also generalizes another famous

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 15; pp. 15:1–15:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem, i.e., Node Multiway Cut. Whether SFVS is FPT had been once a well-known open problem [12]. Until 2013, Cygan et al. [11] proved the fixed-parameter tractability of SFVS by giving an algorithm with running time $\mathcal{O}^*(2^{\mathcal{O}(k\log k)})$. Recently, Iwata et al. [30, 31] showed the first single-exponential algorithm with running time $\mathcal{O}^*(4^k)$ for SFVS. In 2018, Hols and Kratsch showed that SFVS has a randomized polynomial kernelization with $\mathcal{O}(k^9)$ vertices [25]. Besides, FVS admits a quadratic kernel [28, 40], whereas whether there is a deterministic polynomial kernel for SFVS is still unknown.

SFVS has also been studied in several graph classes [35, 37, 2, 3, 4], such as interval graphs, permutation graphs, chordal graphs, and split graphs. SFVS remains NP-complete even in split graphs [18], while FVS in split and chordal graphs are polynomial-time solvable [44]. It turns out that both SFVS in Split Graphs (SFVS-S) and SFVS in Chordal Graphs (SFVS-C) can be regarded as implicit 3-Hitting Set. Its importance stems from the fact that 3-Hitting Set can be used to recast a wide range of problems, and now it can be solved in time $\mathcal{O}^*(2.0755^k)$ [42]. On the other hand, when we formulate SFVS-S or SFVS-C in terms of 3-Hitting Set, the structural properties of the input graph are lost. We believe these structural properties can potentially be exploited to obtain faster algorithms for the original problems. However, designing a faster algorithm for SFVS-S and SFVS-C seems challenging. Only recently did Philip et al. [37] improve the running bound to $\mathcal{O}^*(2^k)$, where they needed to consider many cases of the clique-tree structures of the chordal graphs. In some cases, they needed to branch into seven branches. Note that 2^k is another barrier frequently considered in algorithm design and analysis. Some preliminary brute force algorithms, dynamic programming, and advanced techniques, such as inclusion-exclusion, iterative compression, and subset convolution, always lead to the bound 2^k. Breaking the "2^k-barrier" becomes an interesting question for many problems.

We highlight that SFVS-S and SFVS-C are important since they generalize a natural variation of the maximum independent set problem called Prize-Collecting Maximum Independent Set in hypergraphs (PCMIS). In PCMIS, we are given a hypergraph H with n vertices. The object is to find a vertex subset S maximizing the size of S minus the number of hyperedges in H that contain at least two vertices from S. In other words, we may balance the size of the vertex subset against the number of hyperedges on which S violates the independent constraints. The prize-collecting version of many important fundamental problems has drawn certain attention recently, such as Prize-Collecting Steiner Tree [36], Prize-Collecting Network Activation [21], and Prize-Collecting Travelling Salesman Problem [5]. To the best of our knowledge, no exact algorithm for PCMIS faster than $\mathcal{O}^*(2^n)$ is known before.

Fomin et al. [19] showed that Cluster Vertex Deletion and Directed FVS in Tournaments admit subquadratic kernels with $\mathcal{O}(k^{5/3})$ vertices and $\mathcal{O}(k^{3/2})$ vertices, respectively; while the size of the best kernel for SFVS-C is still quadratic, which can be easily obtained from the kernelization for 3-Hitting Set [1]. As for parameterized algorithms, Dom et al. [14] first designed an $\mathcal{O}^*(2^k)$-time algorithm for Directed FVS in Tournaments, breaking the barrier of 3-Hitting Set, and the running time bound of which was later improved to $\mathcal{O}^*(1.6191^k)$ by Kumar and Lokshtanov [33]. For Cluster Vertex Deletion, in 2010, Hüffner et al. [27] first broke the barrier of 3-Hitting Set by obtaining an $\mathcal{O}^*(2^k)$-time algorithm. Now it can be solved in $\mathcal{O}^*(1.7549^k)$ time [41].

Contributions and Techniques

In this paper, we contribute to parameterized algorithms for SFVS-S and SFVS-C. Our main contributions are summarized as follows.

1. We firstly break the "2^k-barrier" for SFVS-S and SFVS-C by giving an $\mathcal{O}^*(1.8192^k)$-time algorithm, which significantly improves previous algorithms.
2. We show that an $\mathcal{O}^*(\alpha^k)$-time algorithm ($\alpha > 1$) for SFVS-S leads to an $\mathcal{O}^*(\alpha^n)$-algorithm for PCMIS. Thus, we can solve PCMIS in time $\mathcal{O}^*(1.8192^n)$, also breaking the "2^n-barrier" for this problem for the first time.
3. We make use of the Dulmage-Mendelsohn decomposition of bipartite graphs to catch structural properties, and then we are able to use a new measure μ to analyze the running time bound. This is the most crucial technique for us to obtain a significant improvement. Note that direct analysis based on the original measure k has encountered bottlenecks. Any tiny improvement may need complicated case-analysis.
4. The technique based on Dulmage-Mendelsohn decomposition can only solve SFVS-S. We also propose a divide-and-conquer method by dividing the instance of SFVS-C into instances of SFVS-S. We show that SFVS-C can be solved in time $\mathcal{O}^*(\alpha^k + 1.6191^k)$ if SFVS-S can be solved in time $\mathcal{O}^*(\alpha^k)$.

2 Preliminaries

2.1 Graphs

Let $G = (V, E)$ stand for an undirected graph with a set V of vertices and a set E of edges. We adopt the convention that $n = |V|$ and $m = |E|$. When a graph G' is mentioned without specifying its vertex and edge sets, we use $V(G')$ and $E(G')$ to denote these sets, respectively. For a subset $X \subseteq V$ of vertices, we define the following notations. The *neighbour set* of X, denoted by $N_G(X)$, is the set of all vertices in $V \backslash X$ that are adjacent to a vertex in X, and the *closed neighbour set* of X is expressed as $N_G[X] \coloneqq N_G(X) \cup X$. The subgraph of G induced by X is denoted by $G[X]$. We simply write $G - X \coloneqq G[V \backslash X]$ as the subgraph obtained from G removing X together with edges incident on any vertex in X. For ease of notation, we may denote a singleton set $\{ v \}$ by v.

The *degree* of v in G is defined by $\deg_G(v) \coloneqq |N_G(v)|$. An edge e is a *bridge* if it is not contained in any cycle of G. A *separator* of a graph is a vertex set such that its deletion increases the number of connected components of the graph. The shorthand $[r]$ is expressed as the set $\{ 1, 2, \ldots, r \}$ for $r \in \mathbb{N}^+$.

In an undirected graph $G = (V, E)$, a set $X \subseteq V$ is a *clique* if every pair of distinct vertices u and v in X are connected by an edge $uv \in E$; X is an *independent set* if $uv \notin E$ for every pair of vertices u and v in X; X is a *vertex cover* if for any edge $uv \in E$ at least one of u and v is in X. A subset $S \subseteq V$ is a vertex cover of G if and only if $V \backslash S$ is an independent set. A vertex v is called *simplicial* in G if $N_G[v]$ is a clique [13]. A clique in G is *simplicial* if it is maximal and contains at least one simplicial vertex. A *matching* is a set of edges without common vertices.

2.2 Chordal Graphs and Split Graphs

A *chord* of a cycle is an edge that connects two non-consecutive vertices of the cycle. A graph G is said to be *chordal* if every cycle of length at least 4 contains a chord. A chordal graph G holds the following properties that will be used in the paper: Every induced subgraph of a chordal graph G is chordal, and every minimal separator of G is a clique [13].

Consider a connected chordal graph G, and let $\mathcal{Q}_G$ denote the set of all maximal cliques in G. A *clique graph* of G is an undirected graph $(\mathcal{Q}_G, \mathcal{E}_G, \sigma)$ with the edge-weighted function $\sigma \colon \mathcal{E}_G \to \mathbb{N}$ satisfying that an edge $Q_1Q_2 \in \mathcal{E}_G$ if $Q_1 \cap Q_2$ is a minimal separator and

$\sigma(Q_1Q_2) \coloneqq |Q_1 \cap Q_2|$. A *clique tree* $\mathcal{T}_G$ of G is a maximum spanning tree of the clique graph of G, and the following facts hold [7, 23, 43]: (1) Each leaf node of a clique tree $\mathcal{T}_G$ is a simplicial clique in G; (2) For a pair of maximal cliques Q_1 and Q_2 such that $Q_1Q_2 \in \mathcal{E}_G$, $Q_1 \cap Q_2$ separates each pair of vertices $v_1 \in Q_1 \backslash Q_2$ and $v_2 \in Q_2 \backslash Q_1$.

Whether a graph is chordal can be checked in linear time $\mathcal{O}(n+m)$ [38]. The number of maximal cliques in a chordal graph G is at most n [22], and all of them can be listed in linear time $\mathcal{O}(n+m)$ [23]. These properties will be used in our algorithm.

A graph is a *split graph* if its vertex set can be partitioned into a clique K and an independent set I [39]. Such a partition (I, K) is called a *split partition.* It is worth noting that every split graph is chordal, and whether a graph is a split graph can also be checked in linear time $\mathcal{O}(n+m)$ by definition.

2.3 Subset Feedback Vertex Set in Split and Chordal Graphs

Given a terminal set $T \subseteq V$ of an undirected graph $G = (V, E)$, a cycle in G is a *T-cycle* if it contains a terminal from T, and a *T-triangle* is specifically a T-cycle of length three. A *subset feedback vertex set* of a graph G with a terminal set T is a subset of V whose removal makes G contain no T-cycle.

In this study, we focus on SFVS in split and chordal graphs. The problem takes as input a chordal graph $G = (V, E)$, a terminal set $T \subseteq V$, and an integer k. The task is to determine whether there is a subset feedback vertex set S of size at most k. Moreover, the following lemma shows that the problem can be transformed into the problem of finding a subset of vertices intersecting all T-triangles instead of all T-cycles.

▶ **Lemma 1** ([37]). *Let $G = (V, E)$ be a chordal graph and $T \subseteq V$ be the terminal set. A vertex set $S \subseteq V$ is a subset feedback vertex set of G if and only if $G - S$ contains no T-triangles.*

For the sake of presentation, this paper considers a slight generalization of SFVS-C. In this generalized version, a set of marked edges $M \subseteq E$ is further given, and we are asked to decide whether there is a subset feedback vertex set of size at most k, which also covers all marked edges, i.e., each marked edge must have at least one of its endpoints included in the set. This set is called a *solution* to the given instance. Among all solutions, a *minimum solution* is the one with the smallest size. The size of a minimum solution to an instance $\mathcal{I}$ is denoted by $s(\mathcal{I})$. Formally, the generalization of SFVS-C is defined as follows.

(Generalized) SFVS-C
Input: A chordal graph $G = (V, E)$, a terminal set $T \subseteq V$, a marked edge set $M \subseteq E$, and an integer k.
Output: Determine whether there is a subset of vertices $S \subseteq V$ of size at most k, such that neither edges in M nor T-cycles exist in $G - S$.

We have the following simple observations. Let abc be a T-triangle with a degree-2 vertex b in the graph. Any solution must contain at least one of the vertices a, b, and c. If vertex b is included in the solution, we can replace it with either a or c without affecting the solution's feasibility. Consequently, we can simplify the graph by removing b and marking edge ac. This observation motivates the consideration of the generalized version.

We will simply use SFVS-C to denote the generalized version. When the input graphs are restricted to split graphs, the problem becomes SFVS-S. An instance of our problem is denoted by $\mathcal{I} = (G, T, M, k)$. During our algorithm, it may be necessary to consider some sub-instances where the graph is a subgraph of G. We define the *instance induced by $X \subseteq V$ or $G[X]$* as $(G[X], T \cap X, M \cap E(G[X]), k)$.

In this paper, our algorithms follow a standard branch-and-reduce paradigm. An operation on the input instance, such as the reduction rule, is *safe* if the input instance is a Yes-instance if and only if the output instance is a Yes-instance. A branching operation is *safe* if the input instance is a Yes-instance if and only if at least one of the resulting sub-instances is a Yes-instance. Additionally, we use branching vectors and branching factors in our analysis. The definitions of these standard concepts can be found in [10].

3 The Dulmage-Mendelsohn Decomposition and Reduction

This section introduces the Dulmage-Mendelsohn decomposition of a bipartite graph [15, 16]. The Dulmage-Mendelsohn decomposition will play a crucial role in our algorithm for SFVS-S.

▶ **Definition 2** (Dulmage-Mendelsohn Decomposition [34, 9]). *Let F be a bipartite graph with bipartition $V(F) = A \cup B$. The Dulmage-Mendelsohn decomposition (cf. Fig. 1) of F is a partitioning of $V(F)$ into three disjoint parts C, H and R, such that*

1. *C is an independent set and $H = N_F(C)$;*
2. *$F[R]$ has a perfect matching;*
3. *H is the intersection of all minimum vertex covers of F; and*
4. *any maximum matching in F includes all vertices in $R \cup H$.*

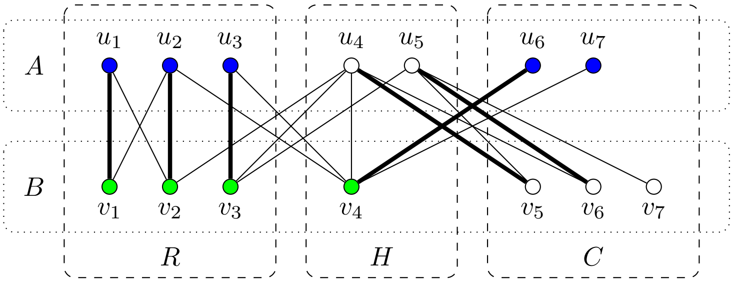


Figure 1 A bipartite graph F with bipartition $V(F) = A \cup B$, where $A = \{u_i\}_{i=1}^{7}$ and $B = \{v_i\}_{i=1}^{7}$. The thick edges form a maximum matching of F. The Dulmage-Mendelsohn decomposition of F is (C, H, R) with $C = \{u_6, u_7, v_5, v_6, v_7\}$, $H = \{u_4, u_5, v_4\}$, and $R = \{u_1, u_2, u_3, v_1, v_2, v_3\}$. If F is an auxiliary subgraph of an instance of SFVS-S, then $\hat{A} = \{u_1, u_2, u_3, u_6, u_7\}$ (denoted by blue vertices) and $\hat{B} = \{v_1, v_2, v_3, v_4\}$ (denoted by green vertices).

The Dulmage-Mendelsohn decomposition always exists and is unique [34], which can be computed in time $\mathcal{O}(m\sqrt{n})$ by finding the maximum matching of the graph F [26]. Leveraging this decomposition, we propose a crucial reduction rule for SFVS-S.

Consider an instance $\mathcal{I} = (G = (V, E), T, M, k)$ of SFVS-S. Let (I, K) be a split partition of G, where I is an independent set and K is a clique. Based on the split partition (I, K) of G, we can uniquely construct an auxiliary bipartite subgraph F with bipartition $V(F) = A \cup B$. In subgraph F, partition A is the subset of the vertices in I that are only incident to marked edges and $B = N_G(A)$. In addition, $E(F)$ is the set of all edges between A and B, i.e., $E(F) \coloneqq \{ab \in E : a \in A,\ b \in B\}$. Notice that F contains no isolated vertex, and all edges in F are marked by the definitions of A and B.

Let (R, H, C) denote the Dulmage-Mendelsohn decomposition of the auxiliary subgraph F. Define $\hat{A} \coloneqq A \cap (R \cup C)$ and $\hat{B} \coloneqq B \cap (R \cup H)$ (see Fig. 1). We have $\hat{B} = N_G(\hat{A})$, and there exists a matching saturating all vertices in $\hat{B}$. This indicates that every solution contains at least $|\hat{B}|$ vertices in $\hat{A} \cup \hat{B}$. On the other hand, $\hat{B}$ is a minimum vertex cover of the subgraph induced by $\hat{A} \cup \hat{B}$. Consequently, there exists a minimum solution to $\mathcal{I}$ containing $\hat{B}$. Next, we introduce the reduction rule, which is called the *DM Reduction*.

▶ **Reduction Rule** (DM Reduction). *Let F be the auxiliary subgraph with bipartition $V(F) = A \cup B$, and let (R, H, C) denote the Dulmage-Mendelsohn decomposition of F. If $\hat{A}$ and $\hat{B}$ are non-empty, delete $\hat{A}$ and $\hat{B}$ from the graph G and decrease k by $|\hat{B}| = |R|/2 + |H \cap B|$.*

▶ **Lemma 3.** *The DM Reduction is safe.*

Proof. Recall that $\hat{A} \coloneqq A \cap (R \cup C)$ and $\hat{B} \coloneqq B \cap (R \cup H)$. According to the definition of the Dulmage-Mendelsohn decomposition, we know that $N_F(\hat{A}) = \hat{B}$, and $\hat{B}$ is a minimum vertex cover of the subgraph induced by $\hat{A} \cup \hat{B}$. For a solution S to the input instance $\mathcal{I}$, the size of $S \cap (\hat{A} \cup \hat{B})$ is no less than $|\hat{B}|$ since S covers every edge in M. Let $S' = (S \backslash \hat{A}) \cup \hat{B}$. Observe that $|\hat{A}| > |\hat{B}|$; otherwise, A would be a minimum vertex cover of F, contradicting that H is a subset of any minimum vertex cover. Consequently, we derive that $|S'| \leqslant |S|$. In addition, we can see that S' is also a solution, leading to the safeness of the DM Reduction. ◀

▶ **Lemma 4.** *Given an instance $\mathcal{I} = (G, T, M, k)$ of SFVS-S, let F be the auxiliary subgraph of G with bipartition $V(F) = A \cup B$. If the DM Reduction cannot be applied, for any non-empty subset $A' \subseteq A$, it holds that $|A'| < |N_G(A')|$.*

Proof. If the DM Reduction cannot be applied, the Dulmage-Mendelsohn decomposition of F must be $(R, H, C) = (\varnothing, A, B)$. According to the definition of the Dulmage-Mendelsohn decomposition, A is a vertex cover, and $H = A$ is the intersection of all minimum vertex covers of F. As a result, we know that A is the unique minimum vertex cover of the auxiliary subgraph F. We assume to the contrary that there exists a subset $A' \subseteq A$ such that $|A'| \geqslant |N_G(A')|$. Then we immediately know that $(A \backslash A') \cup N_G(A')$ is a minimum vertex cover distinct from A, leading to a contradiction. ◀

▶ **Lemma 5.** *Given an instance $\mathcal{I} = (G, T, M, k)$ of SFVS-S, let F be the auxiliary subgraph of G with bipartition $V(F) = A \cup B$. If the DM Reduction cannot be applied and $k < |A|$, the instance $\mathcal{I}$ is a No-instance.*

Proof. The size of the solution to $\mathcal{I} = (G, T, M, k)$ is no less than the size of the minimum vertex cover of F since all marked edges need to be covered. If the DM Reduction cannot be applied, Lemma 4 implies that A is the minimum vertex cover of F. Consequently, the size of the minimum solution to $\mathcal{I}$ must be no less than $|A|$, which means that an instance $\mathcal{I}$ is a No-instance if $k < |A|$. ◀

4 Algorithms for SFVS in Split and Chordal Graphs

This section mainly presents an algorithm for SFVS-S. This algorithm plays a critical role in the algorithm for SFVS-C.

4.1 Good Instances

We begin by introducing a special instance of SFVS-S, which we refer to as a *good instance*. We show that solving good instances is as hard as solving normal instances of SFVS-S in some sense.

▶ **Definition 6** (Good Instances). *An instance $\mathcal{I} = (G = (V, E), T, M, k)$ of SFVS-S is called* good *if it satisfies the following properties:*

(i) *$(T, V \backslash T)$ is the split partition, where terminal set T is the independent set and $V \backslash T$ forms the clique;*

(ii) *every marked edge connects one terminal and one non-terminal; and*

(iii) *the DM reduction cannot be applied on the auxiliary subgraph determined by $(T, V \backslash T)$.*

▶ **Lemma 7.** *For any constant $\alpha > 1$, SFVS-S can be solved in time $\mathcal{O}^*(\alpha^k)$ if and only if SFVS-S on good instances can be solved in time $\mathcal{O}^*(\alpha^k)$.*

Proof. We only need to show that if there exists an algorithm `GoodAlg` solving good instances in time $\mathcal{O}^*(\alpha^k)$, there also exists an algorithm for SFVS-S running in the same time bound $\mathcal{O}^*(\alpha^k)$. The other direction is trivial.

Let $\mathcal{I} = (G = (V, E), T, M, k)$ be an instance of SFVS-S. Notice that α is a constant. We select a sufficiently large constant C such that the branching factor of the branching vector $(1, C, C)$ does not exceed the constant α. Our algorithm for SFVS-S is constructed below.

First, we find the split partition (I, K) of G in polynomial time. If $|K| \leqslant 2C$, we solve the instance directly in polynomial time by brute-force enumerating subsets of K in the solution. Otherwise, we assume that the size of K is at least $2C + 1$. We consider two cases.

Case 1. There is a terminal $t \in K$. In this case, we partition $K \setminus \{t\}$ into two parts K' and K'' such that $|K'| \geqslant C$ and $|K''| \geqslant C$. If t is not included in the solution, at most one vertex in the clique $K \setminus \{t\}$ is not contained in the solution. Consequently, either K' or K'' must be part of the solution. We can branch into three instances by either

- removing t, and decreasing k by 1;
- removing K', and decreasing k by $|K'|$; or
- removing K'', and decreasing k by $|K''|$.

This branching rule yields a branching vector $(1, |K'|, |K''|)$ (w.r.t. the measure k) with the branching factor not greater than α since $|K'| \geqslant C$ and $|K''| \geqslant C$.

Case 2. No terminal is in K. For this case, each non-terminal $v \in I$ is not contained in any T-triangle. However, we cannot directly remove v since it may be incident to marked edges. We can add an edge between v and each vertex $u \in K$ not adjacent to v without creating any new T-triangle. This operation will change v from a vertex in I to a vertex in K, preserving the graph as a split graph. After handling all non-terminal $v \in I$, we know that the terminal set and non-terminal set form a split partition. Subsequently, for each marked edge between two non-terminals v and u, we add a new degree-2 terminal t_{uv} adjacent to u and v and unmark the edge uv. We then apply the DM Reduction and obtain a good instance. Finally, we call the algorithm `GoodAlg` to solve the good instance in time $\mathcal{O}^*(\alpha^k)$.

By either branching with a branching factor not greater than α or solving the instance directly in $\mathcal{O}^*(\alpha^k)$ time, our algorithm runs in time $\mathcal{O}^*(\alpha^k)$. ◀

In the rest of this section, we only need to focus on the algorithm, denoted as `GoodAlg`, for good instances of SFVS-S.

4.2 The Measure and Its Properties

With the help of the auxiliary subgraph and DM Reduction (defined in Section 3), we use the following specific measure to analyze our algorithm.

▶ **Definition 8** (The Measure of Good Instances). *Given a good instance $\mathcal{I} = (G, T, M, k)$ of SFVS-S, let F be the auxiliary subgraph of G with bipartition $V(F) = A \cup B$. We define the measure of the instance $\mathcal{I}$ as*

$$\mu(\mathcal{I}) := k - \frac{2}{3}|A|.$$

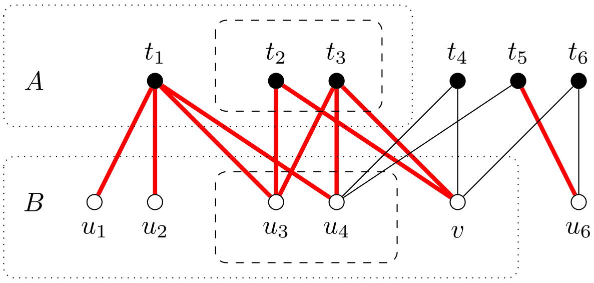


Figure 2 The graph G, where black vertices are terminals, white vertices are non-terminals, thick and red edges are marked edges, and edges between two non-terminals are not presented in the graph; the auxiliary subgraph is F with bipartition $V(F) = A \cup B$, where $A = \{t_1, t_2, t_3\}$ and $B = \{u_1, u_2, u_3, u_4, v\}$ (denoted by dotted boxes). After deleting v, the DM Reduction can be applied. When doing the DM Reduction, $\hat{A} = \{t_2, t_3\}$ and $\hat{B} = \{u_3, u_4\}$ (denoted by dashed boxes) are deleted.

In our algorithm `GoodAlg`, the DM Reduction will be applied as much as possible once the graph changes to keep the instance always good. Additionally, according to Lemma 5, an instance $\mathcal{I}$ can be solved in polynomial time when $\mu(\mathcal{I}) \leqslant 0$. Thus, we can use $\mu(\cdot)$, defined in Definition 8, as our measure to analyze the algorithm.

We may branch on a vertex by including it in the solution or excluding it from the solution in our algorithm. In the first branch, we delete the vertex from the graph and decrease the parameter k by 1. In the second branch, we execute a basic operation of *hiding* the vertex, which is defined according to whether the vertex is a terminal.

Hiding a terminal t: delete every vertex in $N_M(t)$ and decrease k by $|N_M(t)|$.

Hiding a non-terminal v: delete every terminal in $N_M(v)$ and decrease k by $|N_M(v)|$; for each T-triangle vtu containing v, mark edge tu; and last, delete v from the graph.

Here, the notation $N_M(v)$ represents the set of the vertices adjacent to v via a marked edge.

▶ **Lemma 9.** *If there exists a solution containing a vertex v, then it is safe to delete v, decrease k by 1, and do the DM Reduction. If there exists a solution not containing a vertex v, then it is safe to hide v and do the DM Reduction. Moreover, the resulting instance is good after applying either of the above two operations.*

Proof. Assuming a solution S contains v, it is trivial that $S \backslash \{v\}$ is also a solution to the instance $(G - v, T \backslash \{v\}, k - 1)$. Moreover, since the DM Reduction is safe by Lemma 3, the first operation in the lemma is safe.

Now, we assume that a solution S does not contain a vertex v. Since S must cover all edges in M, we know that S contains all neighbours of v in M. This shows that hiding v is safe if v is a terminal. Suppose that v is a non-terminal, for every T-triangle vut containing v, we have that $S \cap \{u, t\} \neq \varnothing$. Consequently, it is safe to mark the edge ut further. Moreover, since the DM Reduction is safe by Lemma 3, the second operation in the lemma is safe.

Finally, either operation only deletes some vertices and marks some edges between terminals and non-terminals. Hence, the terminal set and the non-terminal set still form an independent set and a clique, repetitively. Additionally, the DM Reduction cannot be applied on the resulting instances. Therefore, the resulting instance is good after applying either of the above two operations. ◀

▶ **Lemma 10.** *Given the good instance $\mathcal{I} = (G = (V, E), T, M, k)$, let F be the auxiliary subgraph of G with bipartition $V(F) = A \cup B$, and v be a vertex in $V \backslash A$. Let $\mathcal{I}_1$ be the instance obtained from $\mathcal{I}$ by first deleting v and then doing the DM Reduction (cf. Fig. 2). Then $\mathcal{I}_1$ is a good instance such that $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 0$.*

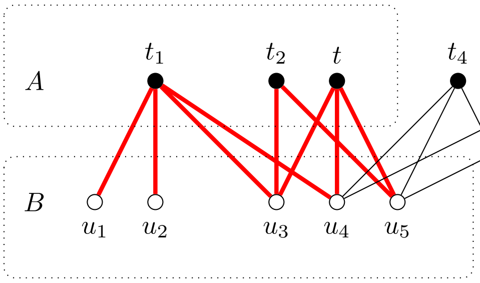


Figure 3 The graph G, where black vertices are terminals, white vertices are non-terminals, thick and red edges are marked edges, and edges between two non-terminals are not presented in the graph; the auxiliary subgraph is F with bipartition $V(F) = A \cup B$, where $A = \{ t_1, t_2, t_3 \}$ and $B = \{ u_1, u_2, u_3, u_4, u_5 \}$ (denoted by dotted boxes). After hiding t, the terminal t_2 becomes isolated, and the DM Reduction cannot be applied.

Proof. Let $\mathcal{I}_0 = (G_0, T_0, M_0, k_0)$ be the instance after deleting v from $\mathcal{I}$. Then, $\mathcal{I}_1 = (G_1, T_1, M_1, k_1)$ is the instance after doing the DM Reduction from $\mathcal{I}_0$. Let F_i with bipartition $V(F_i) = A_i \cup B_i$ be the auxiliary subgraph of G_i, where $i \in \{ 0, 1 \}$.

It is clear that $\mu(\mathcal{I}_0) = \mu(\mathcal{I})$ since no edge is newly marked and $k_0 = k$ holds. Assume that $\hat{A}_0 \subseteq A_0$ and $\hat{B}_0 \subseteq B_0$ are deleted. We note that the DM Reduction cannot be applied if and only if $\hat{A}_0 = \hat{B}_0 = \varnothing$. Observe that $A_1 = A_0 \backslash \hat{A}_0$, $B_1 = B_0 \backslash \hat{B}_0$ and $k_1 = k_0 - |\hat{B}_0|$. Thus, we have

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) = \mu(\mathcal{I}_0) - \mu(\mathcal{I}_1) = (k_0 - 2/3 \cdot |A_0|) - (k_1 - 2/3 \cdot |A_1|) = |\hat{B}_0| - 2/3 \cdot |\hat{A}_0|.$$

If the DM Reduction cannot be applied, then $\hat{A}_0 = \hat{B}_0 = \varnothing$, which already implies that $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) = 0$. Otherwise, the DM Reduction can be applied after deleting v. In this case, we have $v \in B$. Additionally, according to Lemma 4, $\mathcal{I}$ is a good instance implying that $N_G(\hat{A}_0) = \hat{B}_0 \cup \{ v \}$ and $|\hat{B}_0 \cup \{ v \}| > |\hat{A}_0|$. Therefore, we obtain that $|\hat{A}_0| = |\hat{B}_0|$. Thus we get $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 0$. The lemma holds. ◀

▶ **Lemma 11.** *Given a good instance $\mathcal{I} = (G = (V, E), T, M, k)$, let F be the auxiliary subgraph of G with bipartition $V(F) = A \cup B$, and t be a terminal in T. Let $\mathcal{I}_1$ be the instance obtained from $\mathcal{I}$ by first hiding t and then doing the DM Reduction (cf. Fig. 3). Then $\mathcal{I}_1$ is a good instance such that*

- *If $t \in A$, it holds $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 4/3$; and*
- *If $t \notin A$, it holds $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 0$.*

Proof. Let instance $\mathcal{I}_0 = (G_0, T_0, M_0, k_0)$ denote the instance after hiding t from $\mathcal{I}$. Then, $\mathcal{I}_1 = (G_1, T_1, M_1, k_1)$ is the instance after doing the DM Reduction from $\mathcal{I}_0$. Let F_i with bipartition $V(F_i) = A_i \cup B_i$ be the auxiliary subgraph of G_i, where $i \in \{ 0, 1 \}$.

After hiding terminal t, a non-terminal is removed if and only if it is adjacent to t via a marked edge. Thus, $B_0 = B \backslash N_M(t)$ and $k_0 = k - |N_M(t)|$ hold. It follows that

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_0) = (k - 2/3 \cdot |A|) - (k_0 - 2/3 \cdot |A_0|) = |N_M(t)| - 2/3 \cdot (|A| - |A_0|).$$

Notice that after hiding t the only deleted terminal is t. Besides, a terminal $t' \neq t$ is removed from A if and only if it becomes an isolated vertex. It follows that $N_G(t') \subseteq N_M(t)$ if $t' \in A \backslash A_0$.

Assume that $\hat{A}_0 \subseteq A_0$ and $\hat{B}_0 \subseteq B_0$ are deleted after doing the DM Reduction. We note that the DM Reduction cannot be applied if and only if $\hat{A}_0 = \hat{B}_0 = \varnothing$. Observe that $k_1 = k_0 - |\hat{B}_0|$ and $|A_1| = |A_0| - |\hat{A}_0|$. Thus, we derive that

$$\begin{aligned}\mu(\mathcal{I}) - \mu(\mathcal{I}_1) &= |N_M(t)| - 2/3 \cdot (|A| - |A_0|) + |\hat{B}_0| - 2/3 \cdot |\hat{A}_0| \\ &\geqslant |N_M(t) \cup \hat{B}_0| - 2/3 \cdot |A \backslash A_1|.\end{aligned}$$

Consider a terminal $t' \in A\backslash A_1$. If $t' \in A\backslash A_0$, we know $N_G(t') \subseteq N_M(t)$. Otherwise, we have $t' \in \hat{A}_0$, and all neighbours of t' in G are deleted, which indicates that $N_G(t') \subseteq \hat{B}_0 \cup N_M(t)$. It follows that $N_G(A\backslash A_1) \subseteq \hat{B}_0 \cup N_M(t)$. Since $\mathcal{I}$ is good, by Lemma 4, we have $A\backslash A_1 = \varnothing$ or $|N_G(A\backslash A_1)| > |A\backslash A_1|$.

If $A\backslash A_1 = \varnothing$ holds, every terminal in A is not deleted, which implies that $t \notin A$. In this case, we have

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant |N_M(t) \cup \hat{B}_0| \geqslant 0.$$

If $|N_G(A\backslash A_1)| > |A\backslash A_1|$ holds, we have

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant |N_G(A\backslash A_1)| - 2/3 \cdot |A\backslash A_1| \geqslant 4/3.$$

Therefore, we complete our proof. ◀

▶ **Lemma 12.** *Given a good instance $\mathcal{I} = (G = (V,E), T, M, k)$, let F be the auxiliary subgraph of G with bipartition $V(F) = A \cup B$, and v be a non-terminal in $V\backslash T$. Let $\mathcal{I}_1$ be the instance obtained from $\mathcal{I}$ by first hiding v and then doing the DM Reduction (cf. Fig. 4). Then $\mathcal{I}_1$ is a good instance such that $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 0$.*

Furthermore, if every terminal (resp. non-terminal) is adjacent to at least two non-terminals (resp. terminals) and no two 2-degree terminals have identical neighbours in G, it satisfies that

- *If $v \in B$, it holds*

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant \min\left\{ \frac{2}{3}|N_G(v) \cap T| - \frac{1}{3}|N_M(v) \cap A|, \frac{4}{3} \right\}.$$

- *If $v \notin B$, it holds*

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant \min\left\{ \frac{2}{3}|N_G(v) \cap T| + \frac{1}{3}|N_M(v) \cap A|, \frac{4}{3} \right\} = \frac{4}{3}.$$

Proof. Let instance $\mathcal{I}_0 = (G_0, T_0, M_0, k_0)$ denote the instance after hiding v from $\mathcal{I}$. Then, $\mathcal{I}_1 = (G_1, T_1, M_1, k_1)$ is the instance after doing the DM Reduction from $\mathcal{I}_0$. Let F_i with bipartition $V(F_i) = A_i \cup B_i$ be the auxiliary subgraph of G_i, where $i \in \{0, 1\}$.

After hiding v, a vertex is removed if and only if it is a terminal adjacent to v via a marked edge. Thus, $k_0 = k - |N_M(v)|$ and $A\backslash A_0 = A \cap N_M(v)$. It follows that

$$\begin{aligned}\mu(\mathcal{I}) - \mu(\mathcal{I}_0) &= (k - \frac{2}{3}|A|) - (k_0 - \frac{2}{3}|A_0|) \\ &= (k - k_0) + \frac{2}{3}|A_0\backslash A| - \frac{2}{3}|A\backslash A_0| \\ &= |N_M(v)| + \frac{2}{3}|A_0\backslash A| - \frac{2}{3}|A \cap N_M(v)|.\end{aligned}$$

It is easy to see that $\mu(\mathcal{I}) - \mu(\mathcal{I}_0) \geqslant 0$ since $|A \cap N_M(v)| \leqslant |N_M(v)|$. Thus, if the DM Reduction cannot be applied, we have $\mu(\mathcal{I}_0) = \mu(\mathcal{I}_1)$, leading that $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 0$.

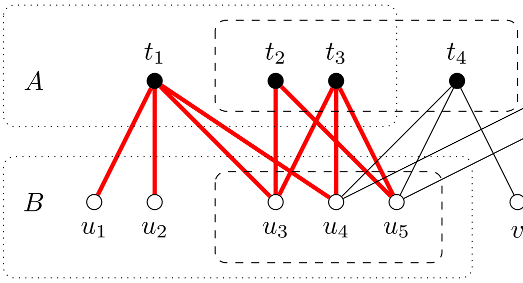


Figure 4 The graph G, where black vertices are terminals, white vertices are non-terminals, thick and red edges are marked edges, and edges between two non-terminals are not presented in the graph; the auxiliary subgraph is F with bipartition $V(F) = A \cup B$, where $A = \{ t_1, t_2, t_3 \}$ and $B = \{ u_1, u_2, u_3, u_4, u_5 \}$ (denoted by dotted boxes). After hiding v, the DM Reduction can be applied. When doing the DM Reduction, $\hat{A} = \{ t_2, t_3, t_4 \}$ and $\hat{B} = \{ u_3, u_4, u_5 \}$ (denoted by dashed boxes) are deleted.

Next, we consider what terminals belong to set $A_0 \backslash A$. On the one hand, a terminal $t \in A_0 \backslash A$ must be adjacent to v via an unmarked edge. On the other hand, t should not be an isolated vertex after hiding v, which implies that the terminal t is adjacent to at least one vertex distinct from v. Thus, if every terminal is adjacent to at least two non-terminals, we derive that $A_0 \backslash A = (N_G(v) \cap T) \backslash N_M(v)$. It follows that

$$\begin{aligned}
\mu(\mathcal{I}) - \mu(\mathcal{I}_0) &= |N_M(v)| + \frac{2}{3}|A_0 \backslash A| - \frac{2}{3}|A \cap N_M(v)| \\
&= |N_M(v)| + \frac{2}{3}|(N_G(v) \cap T) \backslash N_M(v)| - \frac{2}{3}|A \cap N_M(v)| \\
&= |N_M(v)| + \frac{2}{3}(|N_G(v) \cap T| - |N_M(v)|) - \frac{2}{3}|A \cap N_M(v)| \\
&\geqslant \frac{2}{3}|N_G(v) \cap T| - \frac{1}{3}|A \cap N_M(v)|.
\end{aligned}$$

If the DM Reduction cannot be applied on $\mathcal{I}_0$, we can derive that

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) = \mu(\mathcal{I}) - \mu(\mathcal{I}_0) \geqslant \frac{2}{3}|N_G(v) \cap T| - \frac{1}{3}|A \cap N_M(v)|.$$

Furthermore, if $v \notin B$, then v is not adjacent to any terminal in A, leading that $|A \cap N_M(v)| = 0$. In the case that v belongs to B and it is adjacent to at least two terminals, we further have

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) = \mu(\mathcal{I}) - \mu(\mathcal{I}_0) \geqslant \frac{2}{3}|N_G(v) \cap T| \geqslant \frac{4}{3}. \tag{1}$$

Now, we assume that the DM Reduction can be applied on $\mathcal{I}_0$. Suppose $\hat{A}_0 \subseteq A_0$ and $\hat{B}_0 \subseteq B_0$ are deleted. We know that $\hat{A}_0$ and $\hat{B}_0$ are non-empty, and $k_1 = k_0 - |\hat{B}_0| = k - |N_M(v)| - |\hat{B}_0|$ holds. Consider a terminal $t' \in A \backslash A_1$. If $t' \in A \cap N_M(v)$ it is deleted when hiding v; otherwise, it is deleted when doing the DM Reduction which means that $t' \in A \cap \hat{A}_0$. It follows that

$$\begin{aligned}
\mu(\mathcal{I}) - \mu(\mathcal{I}_1) =& (k - \frac{2}{3}|A|) - (k_1 - \frac{2}{3}|A_1|) \\
=& |N_M(v)| + |\hat{B}_0| + \frac{2}{3}|A_1 \backslash A| - \frac{2}{3}|A \backslash A_1| \\
\geqslant& |N_M(v)| + |\hat{B}_0| + \frac{2}{3}|A_1 \backslash A| - \frac{2}{3}(|A \cap \hat{A}_0| + |A \cap N_M(v)|) \\
\geqslant& \frac{1}{3}|N_M(v)| + |\hat{B}_0| + \frac{2}{3}|A_1 \backslash A| - \frac{2}{3}|A \cap \hat{A}_0|.
\end{aligned}$$

Now, we analyze the lower bound of $\mu(\mathcal{I}) - \mu(\mathcal{I}_1)$ and there are two cases.

Case 1.1. $A \cap \hat{A}_0$ is non-empty. We observe that in graph G, v is not adjacent to any terminal in $\hat{A}_0$. This is because all the terminals adjacent to v via a marked edge are deleted after hiding v and they do not appear in the graph G_0. Hence we know $\hat{B}_0 = N_G(\hat{A}_0)$. Besides, we have $B = N_G(A)$, and thus $B \cap \hat{B}_0 = N_G(A \cap \hat{A}_0)$ holds. since $\mathcal{I}$ is good and $A \cap \hat{A}_0$ is non-empty, we get $|B \cap \hat{B}_0| > |A \cap \hat{A}_0|$. Then we derive that the decrease of the measure is

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant |\hat{B}_0| - \frac{2}{3}|A \cap \hat{A}_0| \geqslant 1 + \frac{1}{3}|A \cap \hat{A}_0| \geqslant \frac{4}{3}.$$

Case 1.2. $A \cap \hat{A}_0$ is empty. In this case, we can directly obtain that

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant \frac{1}{3}|N_M(v)| + |\hat{B}_0| + \frac{2}{3}|A_1 \backslash A| \geqslant \frac{1}{3}|N_M(v)| + 1 \geqslant 1. \tag{2}$$

Thus, we have proven the measure does not increase.

Finally, we show that $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 4/3$ always holds when the input graph satisfies the condition in the lemma. We consider two subcases.

Case 2.1. $v \in B$. For this subcase, v is adjacent to at least one terminal in A via a marked edge. Hence, set $N_M(v)$ is non-empty, and we get $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant 1/3 + 1 = 4/3$. This completes that the measure is decreased by at least 4/3 for $v \in B$.

Case 2.2. $v \notin B$. We assume to the contrary that if the DM Reduction can be applied and $\mu(\mathcal{I}) - \mu(\mathcal{I}_1) < 4/3$. According to (2), we can obtain that $|N_M(v)| = 0$, $|\hat{B}_0| = 1$ and $|A_1 \backslash A| = 0$. It means that for every terminal t adjacent to v, it satisfies that

1. $\deg_G(t) \geqslant 2$ holds according to the condition in the lemma;
2. tv is unmarked and $t \notin A$ since $N_M(v)$ is empty;
3. t is in A_0 since $\deg_G(t) \geqslant 2$ and v is the unique non-terminal deleted after hiding v;
4. t is deleted when doing the DM Reduction (i.e., $t \in \hat{A}_0$) since $A_1 \subseteq A$ but $t \notin A$; and
5. t has exactly two neighbours in G which are v and the unique vertex in $\hat{B}_0$ (i.e., $N_G(t) = \{ v \} \cup \hat{B}_0$) since $|\hat{B}_0| = 1$.

Above all, we derive that all terminals in $N_G(v)$ are 2-degree vertices and have the same neighbours. By the condition in the lemma, vertex v is adjacent to at least two terminals, contradicting the condition that no two 2-degree terminals have identical neighbours in G. Combine with (1), we conclude that for any vertex $v \notin B$, it holds

$$\mu(\mathcal{I}) - \mu(\mathcal{I}_1) \geqslant \min\left\{ \frac{2}{3}|N_G(v) \cap T|, \frac{4}{3} \right\} = \frac{4}{3}. \qquad ◀$$

4.3 An Algorithm for Good Instances

We now give the algorithm `GoodAlg` that solves the good instances. When introducing a step, we assume all previous steps cannot be applied.

▷ Step 1. If $|A| > k$ or even $\mu(\mathcal{I}) < 0$, return No and quit. If $|T| \leqslant k$, return Yes and quit.

▷ Step 2. Delete any vertex in G that is not contained in any T-triangle or marked edge, and then do the DM Reduction on the instance.

▷ Step 3. If there exists a non-terminal v such that $|N_G(v) \cap T| = 1$, hide v and then do the DM Reduction on the instance.

Note that the input instance is good, and every terminal is in some T-triangle after Step 3. Thus, every terminal (resp. non-terminal) is adjacent to at least two non-terminals (resp. terminals).

▷ Step 4. This step deals with some degree-2 terminals in $T \backslash A$, and there are two cases.

1. Let t be a degree-2 terminal in $T \backslash A$. If t is adjacent to exactly one marked edge, hide t and do the DM Reduction.
2. Let t and t' be two degree-2 terminals in $T \backslash A$. If t and t' have the same neighbours and none of them is adjacent to a marked edge, delete one of them and do the DM reduction.

After this step, one can easily find that the condition in Lemma 12 holds.

▷ Step 5. If there exists a non-terminal $v \in B$ adjacent to exactly one terminal t via a marked edge and exactly one terminal t' via an unmarked edge, we branch into two instances by either

- hiding the vertex v and doing the DM Reduction; or
- hiding the vertex t and doing the DM Reduction.

▷ Step 6. If there exists a non-terminal $v \in V \backslash T$ incident to at least one unmarked edge, we branch into two instances by either

- deleting the vertex v, decreasing k by 1, and doing the DM Reduction; or
- hiding v and doing the DM Reduction.

Based on Lemmas 9-12, we can show that Steps 1-4 are safe and they do not increase the measure. After applying any one of Steps 1-6, the resulting instance (or each resulting instance of the branching rule) is good. The complete proofs can be found in the full version.

One can easily find that every edge between a terminal and a non-terminal is marked if Step 6 cannot be applied. Thus, we have $T = A$, and Step 1 will be applied and return the answer. Therefore, we obtain the following result.

▶ **Lemma 13.** *SFVS-S can be solved in time* $\mathcal{O}^*(1.8192^k)$.

Proof. `GoodAlg` contains only two branching operations in Steps 5 and 6. By Lemmas 10, 11, and 12, their branching vectors are not worse than $(1, 4/3)$ whose branching factor is 1.81918. Thus, we conclude that `GoodAlg` solves good instances of SFVS-S in time $\mathcal{O}^*(1.81918^k)$. According to Lemma 7, SFVS-S can be solved in time $\mathcal{O}^*(1.81918^k) \leqslant \mathcal{O}^*(1.8192^k)$. ◀

4.4 SFVS in Chordal Graphs

Our result for SFVS-S (i.e., the $\mathcal{O}^*(1.8192^k)$-time parameterized algorithm) can also be effectively adapted to develop fast parameterized algorithms for SFVS-C.

Our algorithm for SFVS-C is divided into two parts. In the first part, we introduce some reduction rules and branching rules to deal with several easy cases and simplify the instance. If none of the steps in the first part can be applied, we call the instance a "thin" instance. In a thin instance, if all terminals are simplicial, we can easily reduce it to a good instance of SFVS-S and solve it by calling `GoodAlg`. However, if there are "inner" terminals (terminals not being simplicial), we employ a divide-and-conquer approach based on the clique-tree decomposition of chordal graphs in the second part. This technique involves branching on a minimal separator containing inner terminals. In each branch, we will obtain a good instance of SFVS-S for each sub-instance and call `GoodAlg` to solve it. We finally obtain Theorem 14. The details of the algorithm and analysis can be found in the full version.

▶ **Theorem 14.** *For any constant $\alpha > 1$, SFVS-C can be solved in time $\mathcal{O}^*(\alpha^k + 1.6191^k)$ if SFVS-S can be solved in time $\mathcal{O}^*(\alpha^k)$.*

▶ **Corollary 15.** *SFVS-C can be solved in time $\mathcal{O}^*(1.8192^k)$.*

5 Prize-Collecting Maximum Independent Set in Hypergraphs

Although we study the subset feedback vertex set problem in graph subclasses, SFVS-S already generalizes other interesting problems.

Several graph connectivity problems [24, 8, 20] can be modeled as natural problems in hypergraphs. In hypergraphs, an edge can connect any number of vertices, whereas in an ordinary graph, an edge connects exactly two vertices. Given a hypergraph H, the set of vertices and hyperedges are denoted by $V(H)$ and $E(H)$, respectively. The maximum independent set problem in hypergraphs aims to find a maximum vertex subset $I \subseteq V(H)$ such that every hyperedge contains at most one vertex from I. The maximum independent set problem in hypergraphs can be easily reduced to the maximum independent set problem in ordinary graphs: we only need to replace each hyperedge $e \in E(H)$ with a clique formed by the vertices in e to get an ordinary graph. In terms of exact algorithms, we may not need to distinguish this problem in hypergraphs and ordinary graphs. However, the prize-collecting version in hypergraphs becomes interesting, which allows us to violate the independent constraint with penalty. As mentioned above, the prize-collecting version of many central NP-hard problems has drawn certain attention recently.

Prize-Collecting Maximum Independent Set in hypergraphs (PCMIS)
Input: A hypergraph H and an integer p.
Output: Determine whether there is a subset of vertices $I \subseteq V(H)$ of the prize at least p, where the prize of I is the size of I minus the number of hyperedges that contain at least two vertices from I.

▶ **Lemma 16.** *PCMIS is polynomially solvable for $p \leqslant 1$, and PCMIS is NP-hard for each constant $p \geqslant 2$.*

Proof. By definition, any singleton set has a prize of 1. Therefore, PCMIS is polynomially solvable when $p \leqslant 1$.

We will prove the NP-hardness of PCMIS with $p \geqslant 2$ by reducing from the maximum independent set problem in ordinary undirected graphs. Let (G, k) be an instance of the maximum independent set problem. We construct an instance (H, p) of PCMIS, where $p \geqslant 2$ is a constant. Since p is a constant, we can assume that $k \geqslant p$.

Suppose $|V(G)| = n$ and $|E(G)| = m$. We now construct a hypergraph H with n vertices and $nm + k - p$ hyperedges. Specifically, for each vertex $v \in V(G)$, we introduce a vertex v', and thus we obtain $V(H) = \{v' : v \in V(G)\}$. Next, for each edge $uv \in E(G)$, we introduce n identical hyperedges $e_i^{uv} = \{u, v\}$ $(i \in [n])$; we also add $k - p$ identical hyperedges $e'_i = V(H)$ $(i \in [k - p])$. Hence, H contains $nm + (k - p)$ hyperedges: $E(H) = E'_1 \cup E'_2$, where $E'_1 = \{e_i^{uv} = \{u, v\} : uv \in E(G),\ i \in [n]\}$ and $E'_2 = \{e'_i = V(H) : i \in [k - p]\}$.

Finally, we show (G, k) is a Yes-instance if and only if (H, p) is a Yes-instance. On the one hand, let $I \subseteq V(G)$ be an independent set of G with the size k. Let $I' = \{v' : v \in I\}$, and we have that every hyperedge in E'_1 contains at most one vertex from I'. Additionally, each hyperedge in E'_2 contains exactly k vertices from I'. Since $k \geqslant p \geqslant 2$, we derive that the prize of I' is $k - |E'_2| = k - (k - p) = p$, which means that (H, p) is a Yes-instance.

On the other hand, let $I' \subseteq V(H)$ be a vertex subset of H with the prize p. Let X be the set of hyperedges containing at least two vertices from I'. We have that $p = |I'| - |X|$. Note that if one hyperedge $e_i^{vu} \in E_1'$ ($i \in [n]$) is in X, then all the n hyperedges identical with e_i^{vu} should be in X, which will make $p \leqslant |I'| - n \leqslant 0$, a contradiction. Therefore, we derive that $I = \{v \in V(G) : v' \in I'\}$ is an independent set in G and $X \cap E_1' = \varnothing$. Since $X \subseteq E_2'$, we have $|I| = |I'| = p + |X| \geqslant p + (k - p) \geqslant k$. We conclude that I is an independent set of G with size at least k, leading that (G, k) is a Yes-instance. ◀

Previously, no exact algorithm for PCMIS faster than $\mathcal{O}^*(2^n)$ is known. We show that PCMIS can be solved by reducing it to SFVS-S, and then we break the "2^n-barrier" for PCMIS. For an instance (H, p) of PCMIS, we construct an instance (G, T, M, k) of SFVS-S. Suppose that H contains n vertices and m hyperedges. We construct a split graph G as follows. We first introduce a clique with vertices $\{v' : v \in V(H)\}$; then for each hyperedge $e \in E(H)$, introduce a new terminal t_e' whose neighbors are exactly the vertices in the clique corresponding to the vertices in hyperedge e. The terminal set is set as $T = \{t_e' : e \in E(H)\}$, the marked edge set is set as $M = \varnothing$, and let $k = n - p$.

▶ **Lemma 17.** *For any constant $\alpha > 1$, an $\mathcal{O}^*(\alpha^k)$-time algorithm for SFVS-S leads to an $\mathcal{O}^*(\alpha^{n-p})$-algorithm for PCMIS.*

Proof. For an instance (H, p) of PCMIS, we construct an instance (G, T, M, k) of SFVS-S. Suppose that H contains n vertices and m hyperedges. We construct a split graph G as follows. We first introduce a clique with vertices $\{v' : v \in V(H)\}$; then for each hyperedge $e \in E(H)$, introduce a new terminal t_e' whose neighbors are exactly the vertices in the clique corresponding to the vertices in hyperedge e. The terminal set is set as $T = \{t_e' : e \in E(H)\}$, the marked edge set is set as $M = \varnothing$, and let $k = n - p$.

We have the key idea: every hyperedge in a hypergraph contains at most one vertex if and only if the corresponding split graph of the hypergraph contains no T-triangle. Let S be a solution to (G, T, M, k) containing m' terminals and n' non-terminals. We can see that $I = \{v : v' \in V(G) \backslash (T \cup S)\}$ is a vertex set with the prize at least $(n - n') - m' = n - k = p$ in (H, p). As for the opposite direction, suppose I is a solution to (H, p) of size n', and the prize of I is p. We can derive that there are at most $m' = n' - p$ hyperedges containing at least two vertices from I. This means that in G, we can remove m' terminals and $n - n'$ non-terminals to obtain a subgraph without any T-triangle, leading that (G, T, M, k) has a solution of size $(n - n') + m' = n - p = k$. Therefore, (G, T, M, k) is a Yes-instance if and only if (H, p) is a Yes-instance. We finish the proof of Lemma 17. ◀

Based on Lemma 13, we obtain an exact algorithm for PCMIS breaking the 2^n barrier.

▶ **Corollary 18.** *PCMIS can be solved in time $\mathcal{O}^*(1.8192^n)$.*

6 Conclusion

In this paper, we broke the "2^k-barrier" for SFVS in Chordal Graphs. As a corollary, we obtained an exact algorithm faster than $\mathcal{O}^*(2^n)$ for Prize-Collecting Maximum Independent Set in hypergraphs. To achieve this breakthrough, we introduced a new measure based on the Dulmage-Mendelsohn decomposition. This measure served as the basis for designing and analyzing an algorithm that addresses a crucial sub-case. Furthermore, we analyzed the whole algorithm using the traditional measure k, employing various techniques such as a divide-and-conquer approach and reductions based on small separators. The bottleneck of our algorithm occurs when dealing with SFVS in Split Graphs.

We think it is interesting to break the "2^k-barrier" or "2^n-barrier" for more important problems, say the Steiner Tree problem and TSP. For SFVS in general graphs, the best result is $\mathcal{O}^*(4^k)$ [30, 31]. It will also be interesting to reduce the gap between the results in general graphs and chordal graphs.

References

1 Faisal N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010.

2 Tian Bai and Mingyu Xiao. Exact and parameterized algorithms for restricted subset feedback vertex set in chordal graphs. In *Theory and Applications of Models of Computation - 17th Annual Conference, TAMC*, volume 13571 of *LNCS*, pages 249–261. Springer, 2022.

3 Tian Bai and Mingyu Xiao. A parameterized algorithm for subset feedback vertex set in tournaments. *Theor. Comput. Sci.*, 975:114139, 2023.

4 Tian Bai and Mingyu Xiao. Exact algorithms for restricted subset feedback vertex set in chordal and split graphs. *Theor. Comput. Sci.*, 984:114326, 2024. doi:10.1016/J.TCS.2023.114326.

5 Jannis Blauth and Martin Nägele. An improved approximation guarantee for prize-collecting TSP. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, (STOC)*, pages 1848–1861. ACM, 2023.

6 Hans L. Bodlaender. On disjoint cycles. *Int. J. Found. Comput. Sci.*, 5(1):59–68, 1994.

7 Peter Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205–212, 1974.

8 Chandra Chekuri and Chao Xu. Computing minimum cuts in hypergraphs. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1085–1100. SIAM, 2017.

9 Jianer Chen and Iyad A. Kanj. Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):833–847, 2003.

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

11 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Subset feedback vertex set is fixed-parameter tractable. *SIAM J. Discret. Math.*, 27(1):290–309, 2013.

12 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Dániel Marx. 09511 open problems – parameterized complexity and approximation algorithms. In *Parameterized complexity and approximation algorithms*, volume 9511 of *Dagstuhl Seminar Proceedings (DagSemProc)*, pages 1–10, Dagstuhl, Germany, 2010.

13 Gabriel Andrew Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25(1):71–76, 1961.

14 Michael Dom, Jiong Guo, Falk Hüffner, Rolf Niedermeier, and Anke Truß. Fixed-parameter tractability results for feedback set problems in tournaments. *J. Discrete Algorithms*, 8(1):76–86, 2010.

15 A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

16 Andrew L Dulmage. A structure theory of bipartite graphs of finite exterior dimension. *The Transactions of the Royal Society of Canada, Section III*, 53:1–13, 1959.

17 Guy Even, Joseph Naor, and Leonid Zosin. An 8-approximation algorithm for the subset feedback vertex set problem. *SIAM J. Comput.*, 30(4):1231–1252, 2000.

18 Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014.

19 Fedor V. Fomin, Tien-Nam Le, Daniel Lokshtanov, Saket Saurabh, Stéphan Thomassé, and Meirav Zehavi. Subquadratic kernels for implicit 3-hitting set and 3-set packing problems. *ACM Trans. Algorithms*, 15(1):13:1–13:44, 2019.

20 Kyle Fox, Debmalya Panigrahi, and Fred Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contractions. *ACM Trans. Algorithms*, 19(2):13:1–13:22, 2023.

21 Takuro Fukunaga. Spider covers for prize-collecting network activation problem. *ACM Trans. Algorithms*, 13(4):49:1–49:31, 2017.

22 Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15(3):835–855, 1965.

23 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

24 Magnús M. Halldórsson and Elena Losievskaja. Independent sets in bounded-degree hypergraphs. *Discret. Appl. Math.*, 157(8):1773–1786, 2009.

25 Eva-Maria C. Hols and Stefan Kratsch. A randomized polynomial kernel for subset feedback vertex set. *Theory Comput. Syst.*, 62(1):63–92, 2018.

26 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

27 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 47(1):196–217, 2010.

28 Yoichi Iwata. Linear-time kernelization for feedback vertex set. In *44th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 68:1–68:14, 2017.

29 Yoichi Iwata and Yusuke Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021.

30 Yoichi Iwata, Magnus Wahlström, and Yuichi Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM J. Comput.*, 45(4):1377–1411, 2016.

31 Yoichi Iwata, Yutaro Yamaguchi, and Yuichi Yoshida. 0/1/all CSPs, half-integral A-path packing, and linear-time FPT algorithms. In *59th IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 462–473, 2018.

32 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, 1972.

33 Mithilesh Kumar and Daniel Lokshtanov. Faster exact and parameterized algorithm for feedback vertex set in tournaments. In *33rd Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 47 of *LIPIcs*, pages 49:1–49:13, 2016.

34 László Lovász and Michael D. Plummer. *Matching theory*, volume 121 of *North-Holland Mathematics Studies*. Elsevier Science Ltd., London, 1 edition, 1986.

35 Charis Papadopoulos and Spyridon Tzimas. Polynomial-time algorithms for the subset feedback vertex set problem on interval graphs and permutation graphs. *Discret. Appl. Math.*, 258:204–221, 2019.

36 Lehilton Lelis Chaves Pedrosa and Hugo Kooki Kasuya Rosado. A 2-approximation for the k-prize-collecting steiner tree problem. *Algorithmica*, 84(12):3522–3558, 2022.

37 Geevarghese Philip, Varun Rajan, Saket Saurabh, and Prafullkumar Tale. Subset feedback vertex set in chordal and split graphs. *Algorithmica*, 81(9):3586–3629, 2019.

38 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.

39 Földes Stephane and Peter Hammer. Split graphs. In *Proceedings of the 8th south-east Combinatorics, Graph Theory, and Computing*, volume 9, pages 311–315, 1977.

40 Stéphan Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Trans. Algorithms*, 6(2):32:1–32:8, 2010.

41 Kangyi Tian, Mingyu Xiao, and Boting Yang. Parameterized algorithms for cluster vertex deletion on degree-4 graphs and general graphs. In *Computing and Combinatorics - 29th International Conference, (COCOON)*, volume 14422 of *LNCS*, pages 182–194. Springer, 2023.

42 Magnus Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems.* PhD thesis, Linköping University, Sweden, 2007.

43 James Richard Walter. *Representations of rigid cycle graphs.* PhD thesis, Wayne State University, 1972.

44 Mihalis Yannakakis and Fanica Gavril. The maximum k-colorable subgraph problem for chordal graphs. *Inf. Process. Lett.*, 24(2):133–137, 1987.

Tractability of Packing Vertex-Disjoint A-Paths Under Length Constraints

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

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

Diptapriyo Majumdar
Indraprastha Institute of Information Technology Delhi, New Delhi, India

Abhishek Sahu
National Institute of Science Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, Odisha, India

Abstract

Given an undirected graph G and a set $A \subseteq V(G)$, an A-path is a path in G that starts and ends at two distinct vertices of A with intermediate vertices in $V(G) \setminus A$. An A-path is called an (A, ℓ)-path if the length of the path is exactly ℓ. In the (A, ℓ)-Path Packing problem (ALPP), we seek to determine whether there exist k vertex-disjoint (A, ℓ)-paths in G or not.

The problem is already known to be fixed-parmeter tractable when parameterized by $k + \ell$ via color coding while it remains Para-NP-hard when parameterized by k (Hamiltonian Path) or ℓ (P_3-Partition) alone. Therefore, a logical direction to pursue this problem is to examine it in relation to structural parameters. Belmonte et al. initiated a study along these lines and proved that ALPP parameterized by $\mathsf{pw} + |A|$ is W[1]-hard where pw is the pathwidth of G. In this paper, we strengthen their result and prove that it is unlikely that ALPP is fixed-parameter tractable even with respect to a bigger parameter $(|A| + \mathsf{dtp})$ where dtp denotes the distance between G and a path graph (distance to path). We use a randomized reduction to achieve the mentioned result. Toward this, we prove a lemma similar to the influential "isolation lemma": Given a set system $(X, \mathcal{F})$ if the elements of X are assigned a weight uniformly at random from a set of values *fairly large*, then each subset in $\mathcal{F}$ will have a unique weight with *high probability*. We believe that this result will be useful beyond the scope of this paper.

ALPP being hard even for structural parameters like distance to path+$|A|$ rules out the possibility of any FPT algorithms for many well-known other structural parameters, including FVS+$|A|$ and treewidth+$|A|$. There is a straightforward FPT algorithm for ALPP parameterized by vc, the vertex cover number of the input graph. Following this, we consider the parameters CVD (cluster vertex deletion)+$|A|$ and CVD +$|\ell|$ and show the problem to be FPT with respect to these parameters. Note that CVD is incomparable to the treewidth of a graph and has been in vogue recently.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Parameterized complexity, (A, ℓ)-Path Packing, Kernelization, Randomized-Exponential Time Hypothesis, Graph Classes

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.16

Funding *Diptapriyo Majumdar*: Supported by Science and Engineering Research Board (SERB) grant SRG/2023/001592.

Acknowledgements We express our sincere gratitude to Dr. Karthik C. S. and Professor Saket Saurabh for their invaluable suggestions and enlightening discussions.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 16; pp. 16:1–16:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Disjoint Path problems form a fundamental class within algorithmic graph theory. These well-studied problems seek the largest collection of vertex-disjoint (or edge-disjoint) paths that satisfy specific additional constraints. Notably, in the absence of such constraints, the problem reduces to the classical maximum matching problem. One of the most well-studied variants for disjoint path problems is the Disjoint s-t Path where given a graph G and two vertices s and t, the objective is to find the maximum number of internally vertex-disjoint paths between s and t. This problem is polynomial-time solvable by reducing to max-flow problem using Menger's Theorem.

Another classical version of the disjoint path problem is Mader's $\mathcal{S}$-Path. For a graph G and a set $\mathcal{S}$ of disjoint subsets of $V(G)$, an $\mathcal{S}$-Path is a path between two vertices in different members of $\mathcal{S}$. Given G and $\mathcal{S}$, the objective of Mader's $\mathcal{S}$-Path problem is to find the maximum number of vertex-disjoint $\mathcal{S}$-paths. It is solvable in polynomial time, as demonstrated by Chudnovsky, Cunningham, and Geelen [5]. One closely related and well-known variant of Mader's $\mathcal{S}$-Path problem is the A-Path Packing problem [1, 3, 6, 14, 12]. Given a graph G and a subset of vertices A, an *A-path* is a path in G that starts and ends at two distinct vertices of A, and the internal vertices of the path are from $V(G) \setminus A$. The A-Path Packing problem aims to find the maximum number of vertex disjoint A-paths in G. A-Path Packing problem can be modelled as an $\mathcal{S}$-path problem, where for every vertex v, we create a set $\{v\}$ in $\mathcal{S}$. Consequently, the A-Path Packing problem becomes polynomial-time solvable.

Recently, Golovach and Thilikos [11], have explored an interesting variant of the classical s-t path problem known as Bounded s-t Path problem by introducing additional constraints on path lengths. In this variant, given a graph G, two distinct vertices s and t, and an integer ℓ, one seeks to find the maximum number of vertex disjoint paths between s and t of length at most ℓ. Surprisingly, the problem becomes hard with this added constraint in contrast to the classical s-t path problem. In a similar line of study, Belmonte et al. [1] considered the following variant of the A-Path Packing problem.

(A, ℓ)-Path Packing (ALPP)
Input: An undirected graph $G = (V, E)$, $A \subseteq V(G)$ and integers k and ℓ.
Question: Are there k vertex-disjoint A-paths each of length ℓ in G?

This version of A-Path Packing problem is also proved to be intractable [1]. While considering this problem in the parameterized framework, the two most natural parameters for ALPP are the solution size k and the length constraint ℓ. While parameterized by the combined parameter of $k + \ell$, the problem admits an easy FPT algorithm via *color-coding*, parameterized by the individual parameters k and ℓ, the problem becomes Para-NP-hard due to reductions from Hamiltonian Path for k and P_3-Packing for ℓ.

Although it may appear that one has exhausted the possibilities for exploration of the problem within the parameterized framework, another set of parameters, known as *structural parameters*, emerges, allowing for further investigation. Belmonte et al. [1] initiated this line of study by considering the size of set A ($|A|$) as a parameter. They proved that the problem is W[1]-hard parameterized by $\mathsf{pw}(G) + |A|$, which translates to $\mathsf{tw}(G) + |A|$ as well, where $\mathsf{pw}(G)$ and $\mathsf{tw}(G)$ denote the pathwidth and treewidth of G respectively.

The intractability result for the parameter $\mathsf{tw}(G)+|A|$ refutes the possibility of getting FPT algorithms for many well-known structural parameters. Nonetheless, one of the objectives of structural parameterization is to delimit the border of the tractability of the problem, i.e., determining the smallest parameter for which the problem becomes FPT or the largest parameters that make the problem W-hard. Therefore one natural direction is to study ALPP with respect to parameters that are either larger than $\mathsf{tw}(G) + |A|$ or incomparable with $\mathsf{tw}(G) + |A|$.

Our Contribution. As our first result, we improve upon the hardness result of Belmonte et al. [1] by showing hardness for a much larger parameter, $\mathsf{dtp}(G) + |A|$. Here $\mathsf{dtp}(G)$ denotes the distance from G to a path graph (formal definitions of all the parameters can be found in Section 2). We present a randomized reduction from a known W[1]-hard problem, which establishes the hardness of (A, ℓ)-Path Packing under the assumption of randomized Exponential Time Hypothesis (rETH). The randomized reduction technique employed in our proof is highly adaptable and can be utilized to demonstrate the hardness of various analogous problems. We use the following lemma to prove our hardness result, which can be of independent interest.

▶ **Lemma 1** (Separation Lemma). *Let $(X, \mathcal{F})$ be a set system where $\mathcal{F}$ is a family of subsets of X. For an arbitrary assignment of weights $w : X \mapsto [M]$, let $w(S) = \sum_{x \in S} w(x)$ denote the weight of the subset $S \subseteq X$. For any random assignment of weights to the elements of X independently and uniformly from $[M]$, with probability at least $1 - \frac{\binom{|\mathcal{F}|}{2}}{M}$, each set $S \in \mathcal{F}$ has a unique weight.*

In addition to refining the boundaries of hardness, we have also considered the problem with respect to the parameter of $\mathsf{cvd}(G)$ denoting the cluster vertex deletion size of G in combination with the natural parameters A and ℓ. The incomparability of $\mathsf{cvd}(G)$ with tw and pw makes it an intriguing parameter to explore. We have proved that ALPP is fixed-parameter tractable with respect to the parameters $\mathsf{cvd}(G) + |A|$ as well as $\mathsf{cvd}(G) + \ell$ where $\mathsf{cvd}(G)$ denotes the cluster vertex deletion size of G (see Section 2). Below we provide brief overviews of these two algorithms.

$\mathbf{cvd(G) + |A|}$. We start by combining $|A|$ and the $\mathsf{cvd}(G)$ to form a “modulator” M. Now, each A-path comprises subpaths between certain modulator vertices. Once we guess the interaction of M with these paths, the role of the cliques in $G - M$ reduces to providing vertices for these subpaths. Next, we first employ a color-coding scheme and color the cliques. This coloring determines the role of a clique to provide required subpaths of a certain kind. And a clique within a color class is deemed *feasible* if it can provide the necessary subpaths barring the length requirements. The feasibility of a clique is identified through its color and *modulator neighborhood*. Following this, we make an important observation that the largest *feasible* cliques are the optimal choices for providing these length-constrained paths. Subsequently, we design an Integer Linear Program that checks their ability to provide subpaths with necessary length requirements.

$\mathbf{cvd(G)} + \ell$. In the context of a modulator M of size m, each clique in $G - M$ is termed “local”. A path is labeled “local” if it contains no vertices from M and lies entirely within a local component (clique). It is worth noting that there are at most m non-local paths in any optimal solution. From each clique, we designate a few vertices with a *marking scheme* that are utilized in providing these non-local paths. The remaining vertices from cliques are utilized in providing local paths and possess very specific characteristics. Exploiting this property, we can extract such local paths from unmarked vertices of cliques, effectively bounding the size of each clique. Subsequently, we identify *equivalent*/indistinguishable cliques based on their *modulator neighborhood* and only need to retain a few of the equivalent cliques (as almost all of them are used in providing local paths, and we can extract these local paths). This bounds the size of the instance as the clique size, the number of equivalent classes, and the number of equivalent cliques inside a class are bounded.

We design a cubic kernel for a larger parameter (than $\mathsf{tw}(G)$) vertex cover $\mathsf{vc}(G)$ in a manner very similar to the algorithm designed for $\mathsf{cvd}(G) + \ell$.

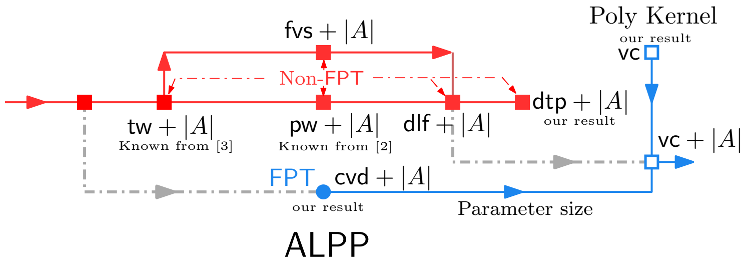


Figure 1 Structural Parameterizations of ALPP. The arrow represents the hierarchy of different structural parameters, while the dashed line represents the parameters that have yet to be explored in the context of our problems.

2 Preliminaries

Sets, Numbers and Graph Theory. We use $\mathbb{N}$ to denote the set of all natural numbers and $[r]$ to denote the set $\{1, \ldots, r\}$ for every $r \in \mathbb{N}$. Given a finite set S and $r \in \mathbb{N}$, we use $\binom{S}{r}$ and $\binom{S}{\leq r}$ to denote the collection of subsets of S with exactly r elements and at most r elements respectively. We use standard graph theoretic notations from the book by Diestel [9]. For any two vertices x, y and a path P, we denote $V[x, y]$ as the number of vertices in the subpath between x and y in P. And for a path P we denote the set of vertices in P by $V(P)$. Further, for a collection $\mathcal{P}$ of paths, $V(\mathcal{P}) = \{\bigcup_{P_i \in \mathcal{P}} V(P_i)\}$. In a graph G, let $P_i(v_1, v_2, \cdots, v_j)$ be a path and $X \subseteq V$ be a set. An *ordered intersection* of P_i with X, denoted as $X_i = (v_a, v_b, \cdots, v_p)$, is defined as $V(P_i) \cap X = X_i$, where the ordering of the vertices in X_i is the same as that in P_i. Additionally, we define $X_1, X_2, \cdots, X_x$ as an *ordered partition* of X if each X_i is an ordered set and $\bigcup_{i=1}^{x} X_i = X$. Given a path $P_i(v_1, v_2, \cdots, v_j)$, we denote $P_i(v_1, v_2, \cdots, v_{j-1})$ as $P_i \setminus (v_{j-1, v_j})$.

Structural Parameters. Given a graph class $\mathcal{H}$ and a graph G, we define the *distance* of G to $\mathcal{H}$ as the minimum number of vertices that need to be deleted to obtain a graph in class $\mathcal{H}$, denoted by $d^{\mathcal{H}}(G)$. For instance, the vertex cover size $\mathsf{vc}(G)$ represents the distance to the class of edgeless graphs or independent sets, the feedback vertex set size $\mathsf{fvs}(G)$ is the distance to the class of forests and cluster vertex deletion set size $\mathsf{cvd}(G)$ denotes the distance to the class of cluster graphs (collection of disjoint cliques). Furthermore, a graph is called a *path graph* if it has only one connected component and that connected component is an induced path. The class of all path graphs and all linear forests are denoted by Γ and $\mathcal{F}$, respectively. We denote $d^{\Gamma}(G)$ and $d^{\mathcal{F}}(G)$ by $\mathsf{dtp}(G)$ and $\mathsf{dlf}(G)$, respectively. Formal definitions of all these parameters can be found in [7].

3 (A, ℓ)-Path Packing Parameterized by $\mathsf{dtp}(G) + |A|$

This section establishes that ALPP is unlikely to be fixed-parameter tractable with respect to the combined parameter $(\mathsf{dtp}(G) + |A|)$ under standard complexity theoretic assumptions. We begin by presenting some key ideas that will be instrumental in our subsequent hardness reduction.

ALPP **Parameter:** $a + m$
Input: A graph G, two subsets $A, M \subseteq V$ of cardinality a and m respectively, and integers k and ℓ such that $G - M \in \Gamma$, where Γ denotes the family of paths.
Question: Are there k vertex-disjoint A-paths each of length exactly ℓ in G?

3.1 Essential Results

We show the hardness for ALPP under the assumption that the randomized Exponential Time Hypothesis (rETH) holds. The concept of rETH was introduced by Dell et al. [8], which states the following. There exists a constant $c > 0$, such that no randomized algorithm can solve 3-SAT in time $\mathcal{O}^*(2^{cn})$ with a (two-sided) error probability of at most $\frac{1}{3}$, where n represents the number of variables in the 3-SAT instance. The $\mathcal{O}^*$ notation hides polynomial factors in the input size.

In the realm of parameterized complexity, rETH has been widely employed to prove hardness for many well-known parameterized problems. The following theorem can be derived from Theorem 12 in [4] when $\epsilon = 1/m$ in Conjecture 5.

▶ **Theorem 2.** *Unless rETH fails, there is no randomized algorithm that decides k-Clique in time $f(k) \cdot n^{o(k)}$ correctly with probability at least $2/3$.*

We establish the intractability result for ALPP parameterized by $(|A| + \mathsf{dtp}(G))$ through a "parameter preserving reduction" from the k-Independent Set problem on a 2-interval graph, which in turn was shown to be W[1]-complete following a reduction from the k-Clique problem by Fellows et al. [10]. A 2-*interval* $\mathcal{I}_i$ is a disjoint pair of intervals $\{I_i^a, I_i^b\}$ on a real line. We say that a pair of 2-intervals, $\mathcal{I}_i$ and $\mathcal{I}_j$ *intersect* if they have at least one point in common, that is $\{I_i^a \cup I_i^b\} \cap \{I_j^a \cup I_j^b\} \neq \emptyset$. Conversely, if two 2-intervals do not intersect, they are called *disjoint*.

A 2-*interval representation* of a graph G is a set of two intervals $\mathcal{J}$ such that there is a one to one correspondence between $\mathcal{J}$ and $V(G)$ such that there exists an edge between u and v if and only if the 2-intervals corresponding to u and v intersect. A graph is a 2-*interval graph* if there is a 2-*interval representation* for G. For a graph G, a set of vertices $W \subseteq V(G)$ is said to be independent if for any pair of vertices u and v in W, $(u, v) \notin E(G)$. Given a graph G, the k-Independent Set problem asks whether there exists a k size independent set in G. Observe that given a 2-interval graph G and a 2-interval representation $\mathcal{J}$ of G a k-independent set W for G corresponds to a set of pairwise disjoint 2-intervals in $\mathcal{J}$.

Fellows et al. [10] presented a parameterized reduction from an arbitrary instance (G, k) of the k-Clique problem to an instance $(\mathcal{J}, k')$ of the k'-Independent Set problem on a 2-interval graph such that there exists a k size clique in G if and only if there exists a $k' = k + 3\binom{k}{2}$ sized independent set in $\mathcal{J}$. Thus, we have the following theorem.

▶ **Theorem 3.** *Unless rETH fails, there is no randomized algorithm that decides k-Independent Set on a 2-interval graph in $f(k) \cdot n^{o(k)}$-time correctly with probability at least $2/3$.*

Next, we present a lemma, which we will use later on. We believe that this can be of independent interest and applicable to various other problem domains. This lemma is similar to the well-known isolation lemma [13]. Recall that $[r]$ denotes the set $\{1, \ldots, r\}$ where $r \in \mathbb{N}^+$.

▶ **Lemma 1** (Separation Lemma). *Let $(X, \mathcal{F})$ be a set system where $\mathcal{F}$ is a family of subsets of X. For an arbitrary assignment of weights $w : X \mapsto [M]$, let $w(S) = \sum_{x \in S} w(x)$ denote the weight of the subset $S \subseteq X$. For any random assignment of weights to the elements of X independently and uniformly from $[M]$, with probability at least $1 - \frac{\binom{|\mathcal{F}|}{2}}{M}$, each set $S \in \mathcal{F}$ has a unique weight.*

Proof. Let w be a random assignment of weights to the elements of X independently and uniformly from $[M]$ and S_1 and S_2 be two arbitrary sets in $\mathcal{F}$. Our objective is to find the probability of the event "$w(S_1) = w(S_2)$". Observe that if $S_1 \setminus S_2 = \emptyset$ or $S_2 \setminus S_1 = \emptyset$, then $\mathbb{P}\big(w(S_1) = w(S_2)\big) = 0$. Let $S_1 \setminus S_2 = \{x_1, x_2, \cdots, x_a\}$ and $S_2 \setminus S_1 = \{y_1, y_2, \cdots y_b\}$. We define a random variable W_{12} as follows.

$$W_{12} = \{w(x_1) + \cdots + w(x_a)\} - \{w(y_2) + \cdots + w(y_b)\} = w(S_1) - w(S_2) + w(y_1)$$

From the law of total probability, we have the following.

$$\begin{aligned}
&\mathbb{P}\big(w(S_1) = w(S_2)\big) = \sum \mathbb{P}\big(w(S_1) = w(S_2) | W_{12} = z\big) \cdot \mathbb{P}(W_{12} = z) \\
&= \sum \mathbb{P}\big(w(y_1) = z\big) \cdot \mathbb{P}(W_{12} = z) \\
&= \sum_{z \in [M]} \mathbb{P}\big(w(y_1) = z\big) \cdot \mathbb{P}(W_{12} = z) \qquad \text{if } z \notin [M] \text{ then } \mathbb{P}\big(w(y_1) = z\big) = 0 \\
&= \sum_{z \in [M]} \frac{1}{M} \mathbb{P}(W_{12} = z) = \frac{1}{M} \sum_{z \in [M]} \mathbb{P}(W_{12} = z) \leq \frac{1}{M}
\end{aligned}$$

Using Boole's inequality, we can prove that none of the two sets in $\mathcal{F}$ are of equal weight with probability at least $1 - \frac{\binom{|\mathcal{F}|}{2}}{M}$. Thus, the claim holds. ◀

3.2 Hardness Proof

We are ready to present a randomized reduction from the k-Independent Set problem in 2-interval graphs to the ALPP problem. Throughout this section, we denote the family of path graphs by Γ. Let $(G_{\mathcal{J}}, k)$ be an instance of k-Independent Set problem in 2-interval graph where the set of 2-intervals representing $G_{\mathcal{J}}$ be $\mathcal{J}$.

Now, we present a randomized construction of an ALPP problem instance (G, A, M, ℓ, k) from $(G_{\mathcal{J}}, k)$ where $H = G \setminus M$ is in Γ and $|M| = 4k$. We assume that we are given $\mathcal{J}$. The construction of G is done in two phases. In the first phase, we generate a set of points P on the real line $\mathbb{R}$. In the second phase, we construct the graph G. Observe that the points in P naturally induce a path graph H which is defined as follows. Corresponding to each point in P, we define a vertex in $V(H)$, and there is an edge between two vertices if the points corresponding to them are adjacent in $\mathbb{R}$. We additionally add $4k$ vertices. Before detailing our construction, let us establish a few notations and assumptions that can be accommodated without changing the combinatorial structure of the problem. Let $\mathcal{J} = \{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_n\}$ where each 2-interval $\mathcal{I}_j$ is a collection of two intervals I_j^a and I_j^b. We presume that all the intervals in $\mathcal{J}$ are inside the interval $[0, 1]$ in the real line. Furthermore, we assume that all the endpoints of the intervals are distinct, and the distance between any two consecutive endpoints is at least 2ϵ, where ϵ is an arbitrarily small constant. We use $L(I)$ and $R(I)$ to denote the left and right endpoints of an interval I, respectively.

Construction Phase 1. We place a set of points P on $\mathbb{R}$ as follows. For each interval I_j^c, where $c \in \{a, b\}$ and $j \in [n]$, we generate two random numbers $n_r(L(I_j^c))$ and $n_r(R(I_j^c))$ between 1 and N. We will decide on the value of N at a later stage.

Let $P(L(I_j^c))$ be the set of $n_r(L(I_j^c))$ equally spaced points in the interval $[L(I_j^c), L(I_j^c)+\epsilon]$. The first point of $P(L(I_j^c))$ coincides with $L(I_j^c)$, and the last point coincides with $L(I_j^c) + \epsilon$. Let $P_L = \bigcup_{c\in\{a,b\},j\in[n]} P(L(I_j^c))$. Similarly, let $P(R(I_j^c))$ be the set of $n_r(R(I_j^c))$ equally spaced points in the interval $[R(I_j^c) - \epsilon, R(I_j^c)]$. The first point of $P(R(I_j^c))$ coincides with $R(I_j^c) - \epsilon$ and the last point coincides with $R(I_j^c)$. Let $P_R = \bigcup_{c\in\{a,b\},j\in[n]} P(R(I_j^c))$. An illustration of this process can be seen in Figure 2. Observe that the cardinality of $P_L \cup P_R$ is at most $4nN$.

Consider $\mathcal{I}_j = (I_j^a, I_j^b)$ be a 2-interval, and n_j be the total number of points from $P_L \cup P_R$ that are inside $\mathcal{I}_j$. Let $\overline{n_j} = 8nN - n_j$ for all $j \in [n]$. Define C_j as the collection of $\overline{n_j}$ points evenly distributed within the interval $[2j, 2j + 1]$ and let $P_C = \cup_{j\in[n]} C_j$. To ease the notations, we denote $L_j = 2j$, $R_j = 2j + 1$ and I_j as the interval (L_j, R_j). Observe that the total number of points inside I_j^a, I_j^b and I_j is $8nN$.

We add a large number of points (exactly $8nN + 4$ many) between R_j and L_{j+1} for $j \in [n]$ and denote all these points by P_X. Let $P = P_L \cup P_R \cup P_C \cup P_X$.

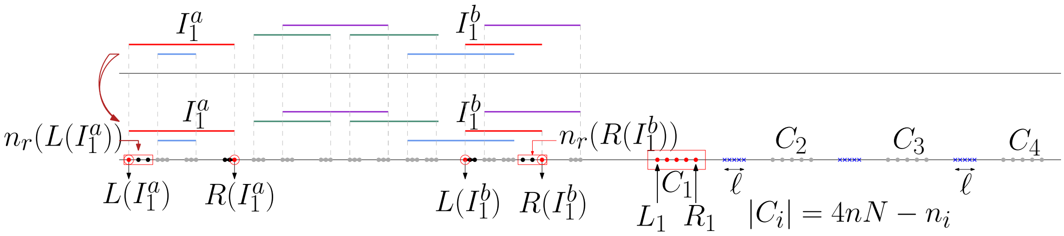


Figure 2 Illustration of Construction Phase 1. Note that $|C_i| = \overline{n_i} = 8nN - n_i$.

Construction Phase 2. Consider the set of points P in $\mathbb{R}$. Observe that there is a natural ordering among the points in P. We define adjacency based on this ordering. Consider the path graph G_P induced by P. Specifically, we introduce a vertex for each point in P and add an edge between two vertices if and only if the points corresponding to them are adjacent. With slight abuse of notation, we denote the vertex corresponding to a point p by p. For any interval I, let $V(I)$ denote the number of points/vertices within the interval I. For two points a and b in P, $\lambda[a, b]$ denotes the path from a to b in G_P.

We construct G, by setting $V(G) = V(G_P) \cup V_M$ where $V_M = \{a_i, b_i, c_i, d_i | \ i \in [k]\}$. And, $E(G) = E(G_P) \cup E_a \cup E_b \cup E_c \cup E_d$, where $E_a = \{(a_i, L(I_j^a)) \mid j \in [n] \text{ and } i \in [k]\}$, $E_b = \{(b_i, R(I_j^a)), (b_i, L_j) \mid j \in [n] \text{ and } i \in [k]\}$, $E_c = \{(c_i, L(I_j^b)), (c_i, R_j) \mid j \in [n] \text{ and } i \in [k]\}$, $E_d = \{(d_i, R(I_j^b)) \mid j \in [n] \text{ and } i \in [k]\}$.
Informally, the edges are defined as follows. Each $a_i \in V_M$ is adjacent to $L(I_j^a)$ for all $j \in [n]$ (see Figure 3). Similarly each $b_i \in V_M$ is adjacent to $R(I_j^a)$ for all $j \in [n]$. Additionally, each b_i is adjacent to every other L_j for all $j \in [n]$. Each c_i is adjacent to $L(I_j^b)$ and R_j, and each d_i is adjacent to $R(I_j^b)$.
We denote $A = \{a_i, d_i | \ i \in [k]\}$ and set $\ell = 8nN + 4$.

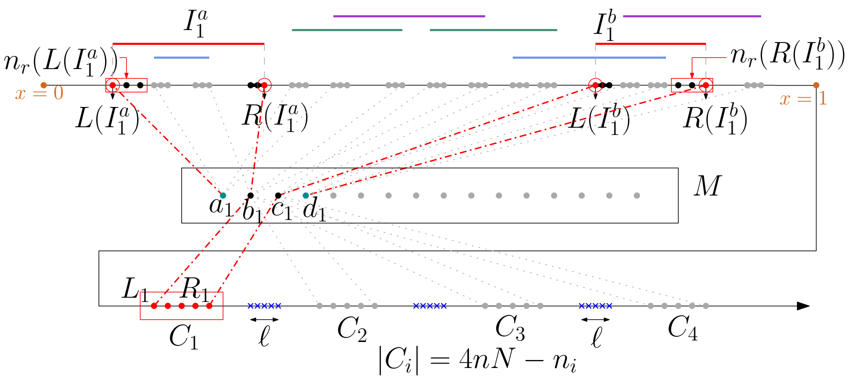


Figure 3 Illustration of construction of (A, ℓ)-Path Packing. Note that $|C_i| = \overline{n_i} = 8nN - n_i$.

Next, we demonstrate that if there exists a k-size independent set in $G_{\mathcal{J}}$, then with probability 1 there exist k many vertex-disjoint (A, ℓ)-paths in G where $\ell = 8nN + 4$ (Lemma 4). Following this, we show that if there exist k many vertex-disjoint (A, ℓ)-paths in G where $\ell = 8nN + 4$, then with *high* probability there is a k-size independent set in $G_{\mathcal{J}}$.

▶ **Lemma 4.** *If there are k disjoint 2-intervals in $\mathcal{J}$, then there are k vertex-disjoint (A, ℓ)-paths in G.*

Proof. Without loss of generality, let us assume that the k disjoint 2-intervals are $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$. Recall that for two vertices a and b in the path graph G_P, $\lambda[a, b]$ denotes the path from a to b in G_P. Consider the following set of paths, defined for $1 \le i \le k$,

$$\lambda_i = a_i \cdot \lambda[L(I_i^a), R(I_i^a)] \cdot b_i \cdot \lambda[L_i, R_i] \cdot c_i \cdot \lambda[L(I_i^b), R(I_i^b)] \cdot d_i$$

Observe that the paths in $\{\lambda_i | 1 \le j \le k\}$ are pairwise vertex-disjoint, each having ℓ vertices, with two endpoints at two different vertices in A (See Figure 4 for an illustration). By construction, they contain $\ell = 8nN + 4$ many vertices. Thus, the claim holds. ◀

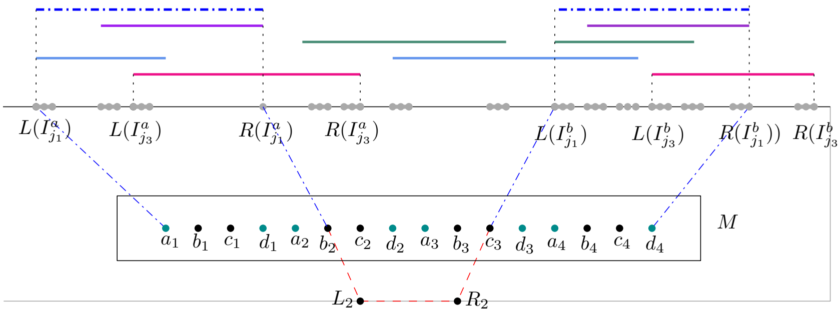


Figure 4 Construction of the paths λ_i in the proof of Lemma 4.

Next, we show that if there exist k many vertex-disjoint (A, ℓ)-paths in G where $\ell = 8nN + 4$, then with *high* probability there is a k size independent set in $G_{\mathcal{J}}$.

Recall that the set of intervals $\{I_j | j \in [n]\}$ where $I_j = [L_j, R_j]$. Let $V_j = V(I_j)$. And, the path corresponding to I_j is $\lambda[L_j, R_j]$. Let us define $\mathcal{C} = \{C_i | 1 \le i \le n\}$. Next, we have the following observation.

▶ **Observation 5.** *For any A-path L of length exactly $\ell = 8nN + 4$ in G, there exists an integer $y \in [n]$ such that $V_y \subset V(L)$ and for any $i \neq y$, $V_i \cap V(L) = \emptyset$. Further L contains the subpath $b_x \cdot \lambda[L_y, R_y] \cdot c_z$ for some $x, z \in [k]$.*

Proof. As the total number of points/vertices in $[0, 1]$ is at most $4nN < \ell$, note that every (A, ℓ)-path should contain all points/vertices from at least one set of points from $\mathcal{C}$. Observe that any path with a length of exactly ℓ contains exactly one set of points $C_i \in \mathcal{C}$ as between two consecutive set of points in $\mathcal{C}$ there are more than ℓ points. As only neighbors of C_i which is not in H is b_x and c_z, where $x, z \in [k]$, therefore, the path must include a subpath of the form $b_x \cdot \lambda[L_y, R_y] \cdot c_z$ for some $y \in [n]$ and $x, z \in [k]$. ◀

Let $\mathcal{P} = \{P_1, \dots, P_k\}$ be a set of k vertex-disjoint A-paths of length exactly ℓ in G. We show that with *high* probability, there is a k size independent set in $G_{\mathcal{J}}$.

▶ **Observation 6.** *Each path $P_i \in \mathcal{P}$ is of the form $p \cdot \lambda[r, s] \cdot b_x \cdot \lambda[L_y, R_y] \cdot c_z \cdot \lambda[t, u] \cdot q$ where $p, q \in A$, $r, s, t, u \in M$, $y \in [n]$ and $x, z \in [k]$.*

Proof. Since the total number of points in A is $2k$, each path must contain exactly two points from A. It follows from Observation 5 that any such path contains a subpath of the form $b_x \cdot \lambda[L_y, R_y] \cdot c_z$. By construction, for any path P, that connects p with b_x, $P \setminus \{p, b_x\}$ induces a continuous set of points on the path graph with endpoints in M (see Figure 4). A similar argument can be made for c_z and q as well. Thus, the claim holds. ◀

Observation 6 indicates that from the disjoint paths in $\mathcal{P}$, it is possible to construct a disjoint set of 3-intervals $\mathcal{J} = \{((x_i, y_i), (z_i, w_i), I_j) | 1 \leq i \leq k\}$ where $x_i, y_i, z_i, w_i \in M$ and $I_j = (L_j, R_j)$. Next we prove that with *high* probability the interval $(x_i, y_i) = I^a_{y_i}$ and the interval $(z_i, w_i) = I^b_{y_i}$, which will give us the desired set of k disjoint 2-intervals.

Let a, b, c, d be any four points in M. Without loss of generality, assume that $a < b < c < d$. Consider the 2-interval $((a, b), (c, d))$ defined by a, b, c, d. Let $\mathcal{J}_{\mathcal{F}}$ be the set of all such intervals, formally defined as $\mathcal{J}_{\mathcal{F}} = \{((a, b), (c, d)) | a, b, c, d \in \text{ and } a < b < c < d\}$. Note that $|\mathcal{J}_{\mathcal{F}}| < n^4$. From Lemma 1, we know that each 2-interval in $\mathcal{J}_{\mathcal{F}}$ contains a unique number of points with probability at least $1 - \frac{\binom{n^4}{2}}{N}$. If we set $N = 3\binom{n^4}{2}$, with probability at least $\frac{2}{3}$ every 2-interval in $\mathcal{J}_{\mathcal{F}}$ contains a unique number of points. Thus with probability at least $\frac{1}{3}$, for every $1 \leq j \leq n$, no other 2-interval except (I^a_j, I^b_j) contains $\ell - V(I_j) - 4$ many points where $V(I)$ denotes the number of points in the interval I. Therefore with probability at least $\frac{2}{3}$, for every 3-interval $((x_i, y_i), (z_i, w_i), I_{y_i})$ in $\mathcal{J}$, $(x_i, y_i) = I^a_{y_i}$ and $(z_i, w_i) = I^b_{y_i}$. Hence we have the following lemma.

▶ **Lemma 7.** *If there are k vertex-disjoint (A, ℓ)-paths in G, then there are k disjoint 2-intervals in $\mathcal{J}$.*

The following conclusive theorem arises from the combination of Theorem 3, Lemma 4, and Lemma 7.

▶ **Theorem 8.** *Unless rETH fails, there is no randomized algorithm for (A, ℓ)-Path Packing which runs in $f(\mathsf{dtp}(G) + |A|) \cdot n^{o(\mathsf{dtp}(G)+|A|)}$-time correctly with probability at least $2/3$.*

4 (A, ℓ)-Path Packing Parameterized by $\mathsf{cvd}(G)+|A|$

In this section, we design an FPT algorithm for the (A, ℓ)-Path Packing problem parameterized by combining the two following parameters: the size of a cluster vertex deletion set and $|A|$. Our algorithm operates under the assumption that we are provided with a minimum size set M as input, satisfying the conditions $A \subseteq M$ and $G - M$ forms a cluster graph. This assumption is justified by the fact that one can efficiently find the smallest size cluster vertex deletion set of $G - A$ in time $1.9102^k \cdot n^{\mathcal{O}(1)}$ [2]. We restate the problem definition and illustrate a brief sketch of our algorithm.

ALPP **Parameter:** $m = |M|$
Input: A graph G, two subsets $A, M \subseteq V$ of cardinality a and m respectively, and integers k and ℓ such that $A \subseteq M$ and $G - M$ is a cluster graph.
Question: Does there exist k vertex-disjoint paths of length exactly ℓ that have endpoints in A?

Overview of the Algorithm. Our algorithm starts by making an educated guess regarding the precisely ordered intersection of each path within an optimal solution ($\mathcal{P}$) with the modulator set M. This involves exploring a limited number of possibilities, specifically on the order of $f(|M|) \cdot n^{\mathcal{O}(1)}$ choices. Once we fix a choice, the problem reduces to finding subpaths (of any given path in $\mathcal{P}$) between modulator vertices satisfying certain length constraints. To provide a formal description, let $P \in \mathcal{P}$ be a path of the form $m_1 P_{1,2} m_2 m_3 m_4 P_{4,5} m_5$ (our guess for $P \in \mathcal{P}$) where each $m_i \in M$ and each $P_{i,i+1}$ is contained in $G - M$. Subsequently, our algorithm proceeds to search for the subpaths $P_{1,2}$ and $P_{4,5}$ each contained in cliques of $G - M$ with endpoints adjacent to vertices m_1, m_2 and m_4, m_5 respectively. A collection of constraints within our final *integer linear program (ILP)* guarantees that the combined length of the paths $P_{1,2}$ and $P_{4,5}$ precisely matches $\ell - 5$, satisfying the prescribed length requirement. Before presenting the ILP formulation, we make informed decisions about the cliques that are well-suited and most appropriate for providing these subpaths. Towards that, we partition the at most $|M|$ many subpaths (originating from all the paths in $\mathcal{P}$), with each partition containing subpaths exclusively from a single clique of $G - M$. Moreover, no two subpaths in separate partitions come from the same clique. Once such a choice is fixed, we apply a color coding scheme on the cliques in $G - M$ where we color the cliques with the same number of colors as the number of sets in the mentioned partition of subpaths into sets. With a *high* probability, each clique involved in the formation of $\mathcal{P}$ is assigned a distinct color. These assigned colors play a crucial role in determining the roles of the cliques in providing subpaths and, consequently, in constructing the final solution $\mathcal{P}$. We show that among all the cliques colored with a single color, a largest size *feasible* clique is an optimal choice for providing the necessary subpaths of $\mathcal{P}$. A *feasible* clique is a clique that is able to provide the necessary subpaths determined by its assigned color, barring the length requirements, and, is identified by its adjacency relation with M. Therefore, we keep precisely one feasible clique of the maximum size for each color and eliminate the others. Thus the number of cliques in the reduced instance is bounded by a function $f(m)$. Following these steps, our problem reduces to finding required subpaths (with length constraints) for which we design a set of ILP equations where the number of variables is a function of m. Below, we give a detailed description of our algorithm.

Algorithm.

Phase 1: The Guessing Phase

1. Find a cluster vertex deletion set S of the minimum size in $G - A$. Then, set $M = A \cup S$.
2. Generate all $M' \subseteq M$. For a fixed M', generate all its *ordered partitions* such that only the first and last vertices of every set of the partitions are from A. Let $\mathcal{M} = \{M'_1, \ldots, M'_{|\mathcal{M}|}\}$ be a fixed such partition of M'.

3. Without loss of generality, let $M'_i = (m_{g(i)}, m_{g(i)+1}, \dots, m_{l(i)})$. For any two consecutive vertices m_j and m_{j+1} where $j \in [g(i), l(i) - 1]$, we introduce a variable $P_{j,j+1}$. These variables serve as placeholders representing subpaths within the optimal solution that we are aiming to find. We use $\mathbb{P}$ to denote the collection of $P_{j,j+1}$.
4. We enumerate all partitions of $\mathbb{P}$. Let $\mathcal{X}_{\mathbb{P}} = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_x\}$ be a partition of $\mathbb{P}$ with $x = |\mathcal{X}_{\mathbb{P}}|$.
5. Additionally, we generate all *valid* functions $h : \mathbb{P} \to \{0, 1, > 1\}$, i.e., we guess whether the length of each subpath is exactly 0, 1, or more than 1. The function h is *valid* if $h(P_{j,j+1}) = 0$, implies m_j and m_{j+1} are adjacent. The validity of h concerning the other two values (1 and > 1) is inherently assured by the presence of a feasible clique.
6. We create a function $T : \mathcal{X}_{\mathbb{P}} \to 2^{2^M}$ as follows. For each part $\mathcal{P}_i \in \mathcal{X}_{\mathbb{P}}$, create the set $T(\mathcal{P}_i) = \{\{m_j, m_{j+1}\} : P_{j,j+1} \in \mathcal{P}_i \text{ and } h(P_{j,j+1}) = 1\} \cup \{\{m_j\} \cup \{m_{j+1}\} : P_{j,j+1} \in \mathcal{P}_i \text{ and } h(P_{j,j+1}) > 1\}$.

At the conclusion of Step 6, we have generated all the tuples denoted as $\tau = (M', \mathcal{M}, \mathbb{P}, \mathcal{X}_{\mathbb{P}}, h, T)$. For each specific τ, we proceed to Phase 2 in order to bound the number of cliques and subsequently generate a set of ILPs.

Phase 2: Bounding the number of Cliques Phase

1. We color all the cliques in $G - M$ with x many colors uniformly at random. From a set of cliques colored with color i, we choose a largest *feasible* clique Q_i. A clique with color i is *feasible* if and only if it has $|T(\mathcal{P}_i)|$ distinct vertices, each being a neighbor to a different set in $T(\mathcal{P}_i)$. Also, we denote the above coloring function by $\mathcal{C}_\tau$.
2. Following the Algorithm, we construct the following set of ILPs.

$$
\begin{aligned}
ILP(\tau, \mathcal{C}_\tau) : \quad & \sum_{j=g(i)}^{l(i)-1} x_{j,j+1} = \ell - |M'_i|, && \forall M'_i \in \mathcal{M} \\
& \sum_{P_{j,j+1} \in \mathcal{P}_i} x_{j,j+1} \leq |Q_i|, && \forall \mathcal{P}_i \in \mathcal{X}_{\mathbb{P}} \\
& x_{j,j+1} = 1, && \text{iff} \;\; h(P_{j,j+1}) = 1
\end{aligned}
$$

Correctness of The Guessing Phase (Steps 1 to 6). Let $\mathcal{P} = \{P_1, \dots, P_p\}$ be an optimal solution of size p where any ℓ-path in $\mathcal{P}$ by definition has both its endpoints in A. In the above ILP, note that the variable $x_{j,j+1}$ represents the path $P_{j,j+1}$. The M' generated in the Step 2 is $V(\mathcal{P}) \cap M$. Each M'_i is the *ordered intersection* of a path P_i with M' , i.e., the sequence of vertices of $V(P_i) \cap M$ appearing in the path is given by M'_i (Step 2). In Step 3, we create the variables (corresponding to the subpaths of $\mathcal{P}$) for each pair of consecutive vertices from M'_i for every $M'_i \in \mathcal{M}$ (Step 3). Any such subpath with a non-zero length is contained in exactly one of the cliques in $G - M$. The subpaths of $\mathcal{P}$ that come from single cliques together are denoted by the partition $\mathcal{X}_{\mathbb{P}}$, i.e., the subpaths (in $\mathcal{P}$) in a part of the partition are exactly the subpaths that are contained in a single clique of $G - M$ (Step 4). We further divide these subpaths into three groups ($h^{-1}(0)$, $h^{-1}(1)$, $h^{-1}(> 1)$) based on whether their lengths are exactly 0, 1, or more than 1 (Step 5). If $h(P_{j,j+1}) = 0$, then the corresponding subpath has length *zero* implying m_j and m_{j+1} are adjacent in $\mathcal{P}$. If $h(P_{j,j+1}) = 1$, then the corresponding subpath has length exactly one and the lone vertex in the subpath is adjacent to both m_j and m_{j+1} in $\mathcal{P}$. When $h(P_{j,j+1}) > 1$, the correcponding subpath has length *more than one* and has two vertices, one is adjacent to m_j while another is adjacent to m_{j+1}. The set $T(\mathcal{P}_i)$ basically stores the adjacency relations (required) between the endpoints of non-zero length subpaths of $\mathcal{P}$ and M. The correctness of the first 6 steps follows directly because of the fact that we exhaust all possible choices at each step.

Correctness of Phase 2. We apply the *color-coding* scheme in the first step of second phase of our algorithm. Each clique $Q_i \in \{Q_1, \ldots, Q_x\}$ that contains vertices from $\mathcal{P}$ gets a different color with *high* probability. Moreover, each Q_i colored with color i exactly contains the subpaths denoted by $\mathcal{P}_i$. We compute this exact probability later in the runtime analysis of our algorithm. Notice the role of the cliques in $G - M$ is to provide subpaths of certain lengths between the vertices from M. And, a *feasible* clique of color i is able to provide all the subpaths in $\mathcal{P}_i$ between the vertices of M, barring the length requirements. Thus, given a feasible clique Q'_i of maximum size, we can reconstruct an equivalent optimal solution $\mathcal{P}'$ in which all its paths within $\mathcal{P}_i$ are entirely contained within Q'_i, all the while sticking to the specified length requirements. This reconstruction can be systematically applied to guarantee the existence of an optimal solution where all its subpaths are derived from a collection of feasible cliques with the largest size available from each color class. Let τ be a correctly guessed tuple, $\mathcal{C}_\tau$ be a correct coloring scheme (coloring each of the x cliques involved in the solution distinctly), and Q'_i be a feasible clique of the largest possible size colored with color i respecting the guessed tuple for each color i. Then, there exists an optimal solution that is entirely contained in the subgraph $G[\bigcup_{i=1}^{x} Q'_i \cup M]$. To obtain the desired solution for a YES instance, we narrow our attention to the subgraph and formulate the specified ILP denoted as ILP$(\tau, \mathcal{C}_\tau)$. The primary objective of the ILP equations is to guarantee that every path we are seeking has an exact length of ℓ. The first set of constraints enforces the specified length requirements for the subpaths, ensuring that each subpath adheres to its designated length. And the second set of constraints ensures that the combined total of all vertices to be utilized from a clique (across all subpaths) in a solution to the ILP does not surpass the total number of vertices within the largest feasible clique.

Runtime Analysis. The total number of ordered partitions generated in Step 2 is $\mathcal{O}(2^m \cdot m^m)$. In Step 3, $|M'_i|$ can be of $\mathcal{O}(m)$. Hence for a fixed $\mathcal{M}$, the number of permutations enumerated is of $\mathcal{O}(m!) \cdot m$. Notice the number of $P_{j,j+1}$ ($|\mathbb{P}|$) is bounded by m. Therefore in the next step, $\mathcal{X}_{\mathbb{P}}$ can be partitioned in m^m ways. In Step 5, each $P_{j,j+1} \in \mathbb{P}$ takes one of the three values. Thus there can be at most 3^m assignments for a fixed $\mathbb{P}$. Hence total number of tuples generated at the end of Step 6 is bounded by $\mathcal{O}(2^m \cdot m^m \cdot m! \cdot m \cdot m^m \cdot 3^m) \equiv 2^{\mathcal{O}(m \log m)}$. Since $x \leq m$, the probability that we get a coloring that colors all the x cliques properly and distinctly is at least $\frac{1}{m!}$. Once we have a good coloring instance, we formulate the ILP$(\tau, \mathcal{C}_\tau)$ to solve the problem. Since both $|\mathcal{M}|$ and $|\mathcal{X}_{\mathbb{P}}|$ are bounded by $\mathcal{O}(m)$, the ILP can be solved in time $m^{\mathcal{O}(m)}$. This immediately implies a randomized FPT algorithm running in time $2^{\mathcal{O}(m \log m)}$. Notice the randomization step (Phase 2) can be derandomized using (m, x)-universal family [7]. And we have the following theorem.

▶ **Theorem 9.** *(A, ℓ)-Path Packing is FPT parameterized by* cvd(G)+$|A|$.

5 (A, ℓ)-Path Packing Parameterized by cvd$(G) + \ell$

In this section, we design an FPT algorithm for the instance (G, S, A, k, ℓ) of ALPP parameterized by the combined parameter cvd$(G) + \ell$.

(A, ℓ)-Path Packing Problem **Parameter:** $|M| (= m) + \ell$
Input: A graph G, two subsets $A, M \subseteq V$ of cardinality a and m respectively, and integers k and ℓ such that $G - M$ is a cluster graph.
Question: Does there exist k vertex-disjoint paths of length exactly ℓ that have endpoints in A?

We denote the set of cliques in $G - M$ by $\mathcal{Q}$ and the vertices in the cliques by $V_{\mathcal{Q}} = \cup_{Q \in \mathcal{Q}} V(Q)$. Let $\mathcal{I} = (G, M, A, k, \ell)$ be a YES instance of ALPP and let $\mathcal{P}$ be any arbitrary solution for $\mathcal{I}$. We denote the set paths in $\mathcal{P}$ that contain at least one vertex from M by $\mathcal{P}^{\mathrm{M}}$ and the set of paths in $\mathcal{P}$ that are completely inside a clique by $\mathcal{P}^{\mathrm{Q}}$. Note that $\mathcal{P}^{\mathrm{M}} \cap \mathcal{P}^{\mathrm{Q}} = \emptyset$ and $\mathcal{P} = \mathcal{P}^{\mathrm{M}} \cup \mathcal{P}^{\mathrm{Q}}$.

▶ **Observation 10.** *The total number of vertices present in the paths $\mathcal{P}^{M}$ is at most $\ell \cdot m$, i.e. $|\cup_{P \in \mathcal{P}^M} V(P)| \leq \ell \cdot m$.*

Next, we present a marking procedure followed by a few reduction rules to bound the size of each clique.

Marking Procedure.

1. For each vertex $u \in M$, mark $\ell m + 1$ many of its neighbors from both $A \cap V(Q)$ and $V(Q) \setminus A$ for each clique $Q \in \mathcal{Q}$. If any clique does not contain that many neighbors of u, we mark all the neighbors of u in that clique.
2. For each pair of vertices u, v in M, mark $\ell m + 1$ many common neighbors of u and v outside A, in every clique of $\mathcal{Q}$.
3. Additionally, mark $\ell m + 1$ many vertices from both $A \cap V(Q)$ and $V(Q) \setminus A$ for each clique $Q \in \mathcal{Q}$.

In the marking procedure the upper bound on the number of marked vertices for each clique Q from A (in $A \cap V(Q)$) is $f_1(\ell, m) = (m+1)(\ell m + 1)$ and the number of marked vertices outside A (in $V(Q) \setminus A$) is $f_2(\ell, m) = (m^2 + m + 1)(\ell m + 1)$.

Exchange Operation. Consider any two arbitrary paths $P_1, P_2 \in \mathcal{P}$ and $< a_1, a_2, a_3 >$ and $< b_1, b_2, b_3 >$ be any two subsequences of vertices in P_1 and P_2 respectively. Let a_2 be a neighbour of b_1 and b_3 and b_2 be a neighbour of a_1 and a_3. We define the operation *exchange* with respect to P_1, P_2, a_2, and b_2 as follows. We create the path P_1' by replacing the vertex a_2 with b_2, and we create the path P_2' by replacing the vertex b_2 with a_2. Observe that $\mathcal{P} \setminus \{P_1, P_2\} \cup \{P_1', P_2'\}$ also forms a solution.

▶ **Lemma 11.** *There exists a solution $\mathcal{P}$ for (G, M, A, k, ℓ) such that all the vertices in $\mathcal{P}^{M}$ are either from M or are marked.*

Proof. Suppose there is a path P in $\mathcal{P}^{\mathrm{M}}$ that contains an unmarked vertex w. There are at most two neighbors of w in P. We assume here that there are exactly two neighbors of w. The case when w has only one neighbor in P can be argued similarly. Let the neighbors of w in P be w_1 and w_2. We have the following three exhaustive cases.

$\boldsymbol{w_1, w_2 \in V(Q)}$**:** Recall that we have marked an additional $\ell m + 1$ many vertices from outside A in each clique (vertices that w may be replaced with) and from Observation 10, we know at most $m\ell$ many of them are contained in $\mathcal{P}^{\mathrm{M}}$. Thus, there is at least one marked vertex, say, w' in $V(Q)$, that is not contained in any path of $\mathcal{P}^{\mathrm{M}}$. If w' is also not contained in any path of $\mathcal{P}^{\mathrm{Q}}$, we simply replace w by w' in P. If it is in a path $P' \in \mathcal{P}^{\mathrm{Q}}$, we do an *exchange* operation with respect to P, P', w and w' and reconstruct a new solution.

$\boldsymbol{w_1 \in V(Q)}$ **and** $\boldsymbol{w_2 \in M}$**:** Recall that we have marked $\ell m + 1$ many vertices from $N(w_2) \cap V(Q) \setminus A$. From Observation 10, at most $m\ell$ many of them are contained in $\mathcal{P}^{\mathrm{M}}$. Hence, there is at least one marked vertex in $V(Q) \setminus A$ that is not contained in any path from $\mathcal{P}^{\mathrm{M}}$. Similar to the arguments outlined in the previous case, we replace the vertex w by

w' when w' is not contained in any path of $\mathcal{P}^{\mathrm{Q}}$, or perform an *exchange* operation with respect to P, P', w and w' and reconstruct a new solution when w' is contained in some $P' \in \mathcal{P}^{\mathrm{Q}}$.

$\boldsymbol{w_1, w_2 \in M}$**:** Using arguments similar to the previous case, we can again replace an unmarked vertex in $\mathcal{P}^{\mathrm{M}}$ with a marked vertex and reconstruct a new solution for this case as well.

After exhaustively replacing unmarked vertices of $\mathcal{P}^{\mathrm{M}}$ (that are not in M), we derive a solution $\mathcal{P}$ in which paths from $\mathcal{P}^{\mathrm{M}}$ do not include unmarked vertices from the cliques. ◀

Henceforth, we seek for a solution $\mathcal{P}$ for (G, M, A, k, ℓ) such that all the vertices in any path of $\mathcal{P}^{\mathrm{M}}$ are either from M or marked. Next, we have the following reduction rule.

▶ **Reduction Rule 1.** *If there exists a clique Q containing a pair of unmarked vertices $u, v \in A$ and a set X of $(\ell - 2)$ unmarked vertices outside A, then delete u, v along with X and return the reduced instance $(G - \{X \cup \{u, v\}\}, M, A \setminus \{u, v\}, k - 1, \ell)$.*

We prove the safeness of the reduction rule below.

▶ **Lemma 12.** *(G, M, A, k, ℓ) is a* YES *instance if and only if $(G - \{X \cup \{u, v\}\}, M, A \setminus \{u, v\}, k - 1, \ell)$ is a* YES *instance.*

Proof. If $(G - \{X \cup \{u, v\}\}, M, A \setminus \{u, v\}, k - 1, \ell)$ is a YES instance with a solution $\mathcal{P}^R$, then (G, M, A, k, ℓ) is also a YES instance as $\mathcal{P}^R$ along with the path formed by $X \cup \{u, v\}$ forms a solution to the instance.

Conversely, let (G, M, A, k, ℓ) be a YES instance with a solution $\mathcal{P}$. Now, we will obtain a solution $\mathcal{P}'$ for $G - \{X \cup \{u, v\}\}$ of size at least $k - 1$. We denote the paths in $\mathcal{P}$ that intersect with $X \cup \{u, v\}$ by $\mathcal{P}_X$ (with slight abuse of notation). If $|\mathcal{P}_X| \leq 1$ then $\mathcal{P}' = \mathcal{P} \setminus \mathcal{P}_X$ is the desired solution. From now on, we assume that $|\mathcal{P}_X| > 1$. Observe that as the vertices in $X \cup \{u, v\}$ are unmarked, $\mathcal{P}_X \cap \mathcal{P}^{\mathrm{M}} = \emptyset$ (from Lemma 11). We can reconstruct a new solution $\mathcal{P}_1$ from $\mathcal{P}$ where exactly one path P in $\mathcal{P}_1$ intersects $X \cup \{u, v\}$, and the rest of the paths in $\mathcal{P}_1$ do not intersect with $X \cup \{u, v\}$, by repeatedly utilizing the *exchange* operation among the paths in $\mathcal{P}_X$. Observe that $\mathcal{P}' = \mathcal{P}_1 \setminus \{P\}$ is the desired solution. Thus, the claim holds. ◀

Note that the upperbound on the number of marked vertices from $A \cap V(Q)$ is $f_1(\ell, m) = (m+1)(\ell m+1)$ and the number of marked vertices $V(Q) \setminus A$ is $f_2(\ell, m) = (m^2+m+1)(\ell m+1)$. And, after exhaustive application of Reduction Rule 1, in any clique Q, either there are at most $\ell - 3$ unmarked vertices in $V(Q) \setminus A$ or at most one unmarked vertex in $A \cap V(Q)$.

Case (i): There is at most one unmarked vertex in $A \cap V(Q)$.
Case (ii): There are at most $\ell - 3$ unmarked vertices in $V(Q) \setminus A$.

Based on the aforementioned cases, we introduce two reduction rules – one for each case – that help us limit the overall number of unmarked vertices in Q, thereby bounding the size of each clique in $G - M$. First we consider the Case (i) when the number of unmarked vertices form $A \cap V(Q)$ is bounded by one and bound the number of the unmarked vertices in $V(Q) \setminus A$ with the following reduction rule.

▶ **Reduction Rule 2.** *If there exists a clique Q containing at most one unmarked vertex from A and at least $(f_1(\ell, m) + 1) \cdot \frac{\ell}{2} + 1$ unmarked vertices outside A, then delete one unmarked vertex $u \in V(Q) \setminus A$ and return the reduced instance $(G - \{u\}, M, A, k, \ell)$.*

Let $G' = G - \{u\}$ be the new graph following an application of Reduction Rule 2. We prove the safeness of the reduction rule in the following lemma.

▶ **Lemma 13.** *(G, M, A, k, ℓ) is a* YES *instance if and only if (G', M, A, k, ℓ) is a* YES *instance.*

Proof. If (G', M, A, k, ℓ) is a YES instance, then (G, M, A, k, ℓ) is a YES instance since G' is a subgraph of G. Conversely, suppose (G, M, A, k, ℓ) is a YES instance, and $\mathcal{P}$ is a solution. If u does not belong to any path in $\mathcal{P}$, then $\mathcal{P}$ is a solution to (G', M, A, k, ℓ) as well. Otherwise, let $P \in \mathcal{P}^{\mathrm{Q}}$ contain u. This is true since any unmarked vertex can only be used in a path in $\mathcal{P}^{\mathrm{Q}}$. But any such path uses exactly 2 vertices from $V(Q) \cap A$. Hence we can upper bound the number of unmarked vertices outside A that are contained in $\mathcal{P}^{\mathrm{Q}}$ and hence $\mathcal{P}$ by $(f_1(\ell, m) + 1) \cdot \frac{\ell}{2}$. Hence, there is at least one unmarked vertex $u' \neq u$ in $V(Q) \setminus A$ which is not used by any path in $\mathcal{P}$. We replace u with u' in P to get a desired solution to (G', M, A, k, ℓ). ◀

For the Case (ii) when the number of unmarked vertices form $V(Q) \setminus A$ is bounded by $l - 3$ and we bound the number of the unmarked vertices in $V(Q) \cap A$ with the following reduction rule.

▶ **Reduction Rule 3.** *If there exists a clique Q containing at most $\ell-3$ unmarked vertices from $V(Q) \setminus A$ and at least $(f_2(\ell, m) + (\ell - 3)) \cdot \frac{1}{\ell-2} + 1$ many unmarked vertices in A, then delete an unmarked vertex $u \in A \cap Q$ and return the reduced instance $(G - \{u\}, M, A \setminus \{u\}, k, \ell)$.*

Proof. If $(G - \{u\}, M, A \setminus \{u\}, k, \ell)$ is a YES instance, then (G, M, A, k, ℓ) is trivially a YES instance since G' is a subgraph of G. Conversely, suppose (G, M, A, k, ℓ) is a YES instance, and $\mathcal{P}$ is a solution. If u does not belong to any path in $\mathcal{P}$, then $\mathcal{P}$ is a solution to (G', M, A, k, ℓ) as well. Otherwise, let $P \in \mathcal{P}^{\mathrm{Q}}$ contain u. This is true since any unmarked vertex can only be used in a path in $\mathcal{P}^{\mathrm{Q}}$. But any such path uses exactly $\ell - 2$ vertices from $V(Q) \cap A$. Hence we can upper bound the number of unmarked vertices from A that are contained in $\mathcal{P}^{\mathrm{Q}}$ and hence $\mathcal{P}$ by $(f_2(\ell, m) + (\ell - 3)) \cdot \frac{1}{\ell-2}$. Hence, there is at least one unmarked vertex $u' \neq u$ in $V(Q) \cap A$ which is not used by any path in $\mathcal{P}$. We replace u with u' in P to get a desired solution to (G', M, A, k, ℓ). ◀

After exhaustive application of Reduction Rules 2 and 3, the upper bound on the number of vertices of different types in each clique is as follows:

- Marked vertices in A: $f_1(\ell, m) = (m + 1)(\ell m + 1)$
- Marked vertices in $V(Q) \setminus A$: $f_2(\ell, m) = (m^2 + m + 1)(\ell m + 1)$
- Unmarked vertices in A: $(f_2(\ell, m) + (\ell - 3)) \cdot \frac{1}{\ell-2} + 2$
- Unmarked vertices in $V(Q) \setminus A$: $(f_1(\ell, m) \cdot \ell + 1$

Hence the total number of vertices in each clique is bounded by $\mathcal{O}(\ell^2 m^2 + \ell m^3)$.

Equivalent cliques. Now we aim to bound the number of cliques by introducing the concept of *equivalent* cliques. Two cliques Q_i and Q_j, are equivalent (belong to the same *equivalent class*) if and only if the number of vertices from the cliques that are in A, and that are outside A with an exact neighborhood of $M' \subseteq M$ is same for each $M' \in 2^M$. Two cliques Q_i, Q_j in an equivalence class are essentially *indistinguishable* from each other, i.e., there is a bijective mapping $g_{ij} : V(Q_i) \mapsto V(Q_j)$, so that $N(u) \cap M = N(g(u)) \cap M$, for all $u \in V(Q_i)$. This fact is crucial in the construction of our next reduction rule. Observe that the number of equivalence classes is at most $\mathcal{O}(\ell^2 m^2 + \ell m^3)^{2^m}$=$f(\ell, m)$. The following reduction rule bounds the number of cliques in each equivalent class.

▶ **Reduction Rule 4.** *If there exists an equivalent class $\mathcal{C}$ with at least $\ell m + 1$ cliques, then delete one of the cliques $Q_i \in \mathcal{C}$ and return the reduced instance $(G - Q_i, M, A \setminus (A \cap V(Q_i)), k - x_i, \ell)$ where, $x_i = \min\left\{\frac{|A \cap V(Q_i)|}{2}, \frac{|V(Q_i) \setminus A|}{\ell - 2}\right\}$.*

▶ **Lemma 14.** *(G, M, A, k, ℓ) is a* YES *instance if and only if $(G - Q_i, M, A \setminus (A \cap V(Q_i)), k - x_i, \ell)$ is a* YES *instance.*

Proof. In the forward direction, let (G, M, A, k, ℓ) be a YES instance. Recall that the number of the vertices contained in paths of $\mathcal{P}_M$ for any optimal solution $\mathcal{P}$ is bounded by ℓs. Thus, there are at most ℓm many cliques in total and also from any equivalence class that has vertices in paths from $\mathcal{P}_M$. Let Q_j be one such clique in the equivalence class $\mathcal{C}$ that does not contain any vertex in the paths from $\mathcal{P}_M$. From the definition of an equivalence class, it is evident that the two cliques Q_i, Q_j in the equivalence class $\mathcal{C}$ are *indistinguishable* from each other, i.e., there is a bijective mapping $g_{ij} : V(Q_i) \mapsto V(Q_j)$, so that $N(u) \cap M = N(g(u)) \cap M$, for all $u \in V(Q_i)$. Let $X_i = V(\mathcal{P}) \cap V(Q_i)$ and $X_j = V(\mathcal{P}) \cap V(Q_j)$, i.e, the set of vertices from Q_i and Q_j that are used in paths from $\mathcal{P}$, respectively. We construct an alternate solution, $\mathcal{P}'$, where we replace X_i with $g_{ij}(X_i)$ and X_j with $g_{ij}^{-1}(X_j)$ in $\mathcal{P}$. Since $X_j \cap M = \emptyset$, we have $g_{ij}^{-1}(X_j) \cap M = \emptyset$. Therefore in $\mathcal{P}'$, there is no path that contain vertices from both M and $V(Q_i)$. In other words, vertices in Q_i can only be contained in paths from $\mathcal{P}' \setminus \mathcal{P}'_M$ (paths that are completely contained inside the clique). And, the number of such paths is bounded by $x_i = \min\left\{\frac{|A \cap V(Q_i)|}{2}, \frac{|V(Q_i) \setminus A|}{\ell - 2}\right\}$. Hence $(G - Q_i, M, A \setminus (A \cap V(Q_i)), k - x_i, \ell)$ is a YES instance.

In the reverse direction, let $(G - Q_i, M, A \setminus (A \cap V(Q_i)), k - x_i, \ell)$ be a YES instance with a solution $\mathcal{P}$. But there are x_i many paths (say $\mathcal{P}_i$) that are completely contained in Q_i.Hence, $\mathcal{P} \cup \mathcal{P}_i$ is a set of k vertex-disjoint (A, ℓ)-paths contained in G, making (G, M, A, k, ℓ) a YES instance. ◀

After exhaustively applying all the aforementioned reduction rules, the following bounds hold.

- The number of vertices in each clique is bounded by $\mathcal{O}(\ell^2 m^2 + \ell m^3)$.
- The number of equivalence classes is at most $\mathcal{O}(\ell^2 m^2 + \ell m^3)^{2^m}$.
- The number of cliques in each equivalence class is at most $\ell m + 1$.

Consequently, the size of the reduced instance is upper-bounded by a computable function of ℓ and m, thus directly implying the following theorem.

▶ **Theorem 15.** ALPP *parameterized by* $\mathsf{cvd}(G) + \ell$ *admits an algorithm running in FPT time.*

6 (A, ℓ)-Path Packing parameterized by vc(G)

In this section, we design a polynomial kernel for (A, ℓ)-Path Packing parameterized by the size of a vertex cover of the graph.

ALPP **Parameter:** $m = |M|$
Input: An undirected graph $G = (V, E)$, $A, M \subseteq V(G)$ such that M is a vertex cover of G and integers k and ℓ.
Question: Are there k vertex-disjoint A-paths each of length ℓ in G?

For a YES instance (G, M, A, k), a solution $\mathcal{P}$ contains at most $3m$ vertices. This limitation arises because there are no consecutive vertices from $I = V(G) - M$ in any path within $\mathcal{P}$.

▶ **Observation 16.** *Any solution $\mathcal{P}$ to a* YES *instance of (G, M, A, k, ℓ), has at most $3m$ vertices.*

Our kernelization approach comprises the following marking process followed by a reduction rule that bounds the instance size by a polynomial function of m.

Marking Procedure.

1. For each vertex $u \in M$, mark $3m + 1$ many of its neighbors in $I \cap A$. If any vertex $u \in M$ has less than $3m + 1$ neighbors, we mark all of them.
2. For each pair of vertices $u, v \in M$, mark $3m + 1$ many common neighbors in $I \setminus A$ for each clique. If any pair $u, v \in M$ has less than $3m + 1$ common neighbors, we mark all of them.

Now we apply the following reduction rule to eliminate unmarked vertices in I.

▶ **Reduction Rule 5.** *We delete any unmarked vertex $u \in I$ from G, and return the reduced instance $(G - \{u\}, M, A, k, \ell)$.*

Let $G' = G - \{u\}$ be the new graph obtained after an application of Reduction Rule 5. The safeness of the reduction rule is not very difficult to see and will be provided in the full version.

Following the exhaustive application of the Reduction Rule 5, there are $3m + 1$ vertices marked in I for each pair of vertices as well as each individual vertex in M. Consequently, in the reduced instance, $|I|$ is bounded by $\mathcal{O}(m^3)$. As a result, we have the following theorem.

▶ **Theorem 17.** ALPP *parameterized by* vc(G) *admits a kernel with $\mathcal{O}(m^3)$ vertices.*

7 Conclusion

Our results have extended the works of Belmonte et al. [1] by addressing the parameterized complexity status of (A, ℓ)-Path Packing (ALPP) across numerous structural parameters. It was known from Belmonte et al. [1] that ALPP is W[1]-complete when parameterized by pw $+ |A|$. We prove an intractability result for a much larger parameter of dtp$(G) + |A|$. Also, the parameterized complexity of ALPP when parameterized by the combined parameter of cliquewidth and ℓ was an open question [1]. While that problem still remains open, we have been successful in making slight progress by obtaining an FPT algorithm for the problem when parameterized by the combined parameter of cvd(G) and ℓ. Another direction to explore would be to determine the fixed-parameter tractability status of the problem when parameterized by cvd(G) only. It would be interesting to explore if this FPT result can be generalized to the combined parameter of cograph vertex deletion set size and ℓ since cographs are graphs of cliquewidth at most two. We believe that the positive results presented in this paper are not optimal and some of those results can be improved with more involved structural analysis. Therefore, improving the efficiency of our positive results are exciting research direction for future works.

References

1 Rémy Belmonte, Tesshu Hanaka, Masaaki Kanzaki, Masashi Kiyomi, Yasuaki Kobayashi, Yusuke Kobayashi, Michael Lampis, Hirotaka Ono, and Yota Otachi. Parameterized Complexity of (A, l)-Path Packing. *Algorithmica*, 84(4):871–895, 2022.

2 A. Boral, M. Cygan, T. Kociumaka, and M. Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory of Computing Systems*, 58(2):357–376, 2016.

3 Henning Bruhn and Arthur Ulmer. Packing a-paths of length zero modulo four. *Eur. J. Comb.*, 99:103422, 2022.

4 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From gap-eth to fpt-inapproximability: Clique, dominating set, and more. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 743–754, 2017.

5 M. Chudnovsky, William H. Cunningham, and James F. Geelen. An algorithm for packing non-zero a-paths in group-labelled graphs. *Combinatorica*, 28:145–161, 2008.

6 Maria Chudnovsky, Jim Geelen, Bert Gerards, Luis A. Goddyn, Michael Lohman, and Paul D. Seymour. Packing non-zero a-paths in group-labelled graphs. *Comb.*, 26(5):521–532, 2006.

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

8 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the tutte polynomial. *ACM Trans. Algorithms*, 10(4):21:1–21:32, 2014.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics.* Springer, 2012.

10 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

11 Petr A. Golovach and Dimitrios M. Thilikos. Paths of bounded length and their cuts: Parameterized complexity and algorithms. *Discret. Optim.*, 8:72–86, 2011.

12 Hiroshi Hirai and Gyula Pap. Tree metrics and edge-disjoint s-paths. *Math. Program.*, 147(1-2):81–123, 2014.

13 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 345–354. Association for Computing Machinery, 1987.

14 Gyula Pap. Packing non-returning a-paths algorithmically. *Discret. Math.*, 308(8):1472–1488, 2008.

On the Descriptive Complexity of Vertex Deletion Problems

Max Bannach
European Space Agency, Advanced Concepts Team, Noordwijk, The Netherlands

Florian Chudigiewitsch
Universität zu Lübeck, Germany

Till Tantau
Universität zu Lübeck, Germany

Abstract

Vertex deletion problems for graphs are studied intensely in classical and parameterized complexity theory. They ask whether we can delete at most k vertices from an input graph such that the resulting graph has a certain property. Regarding k as the parameter, a dichotomy was recently shown based on the number of quantifier alternations of first-order formulas that describe the property. In this paper, we refine this classification by moving from quantifier alternations to individual quantifier patterns and from a dichotomy to a trichotomy, resulting in a complete classification of the complexity of vertex deletion problems based on their quantifier pattern. The more fine-grained approach uncovers new tractable fragments, which we show to not only lie in FPT, but even in parameterized constant-depth circuit complexity classes. On the other hand, we show that vertex deletion becomes intractable already for just one quantifier per alternation, that is, there is a formula of the form $\forall x \exists y \forall z(\psi)$, with ψ quantifier-free, for which the vertex deletion problem is W[1]-hard. The fine-grained analysis also allows us to uncover differences in the complexity landscape when we consider different kinds of graphs and more general structures: While basic graphs (undirected graphs without self-loops), undirected graphs, and directed graphs each have a different frontier of tractability, the frontier for arbitrary logical structures coincides with that of directed graphs.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Complexity theory and logic; Theory of computation → Fixed parameter tractability; Theory of computation → W hierarchy

Keywords and phrases graph problems, fixed-parameter tractability, descriptive complexity, vertex deletion

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.17

Related Version *Full Version*: https://arxiv.org/abs/2406.18299 [4]

1 Introduction

A recent research topic in parametrized complexity are *distance to triviality problems.* We are asked how many modification steps (the "distance") we need to apply to a logical structure in order to transform it into a "trivial" one – which can mean anything from "no edges at all" to "no cycles" or even more exotic properties like "no cycles of odd length." Such problems have been found highly useful in modern algorithm design [1, 2, 11, 21] and are now an important test bed for new algorithmic ideas and data reduction procedures [14, 15, 22, 23].

Many problems that have been studied thoroughly in the literature turn out to be vertex deletion problems. The simplest example arises from *vertex covers,* which measure the "distance in terms of vertex deletions" of a graph from being edge-free: A graph has a vertex cover of size k iff it can be made edge-free by deleting at most k vertices. For a slightly more complex example, the *cluster deletion problem* asks whether we can delete at most k vertices from a graph so that it becomes a cluster graph, meaning that every connected component is a clique or, equivalently, is P_3-free (meaning, there is no induced path on three vertices). The *feedback vertex set problem* asks if we can delete at most k vertices, such that the resulting graph has no cycles. The *odd cycle transversal problem* asks if there is a set of vertices of size at most k, such that removing it destroys every odd cycle. Equivalently, the problem asks if we can delete at most k vertices, such that the resulting graph is bipartite.

To investigate the complexity of vertex deletion problems in a systematic way, it makes sense to limit the graph properties to have some structure. An early result in this direction [25] is the NP-completeness of vertex deletion to hereditary graph properties that can be tested in polynomial time. Intuitively, vertex deletion problems should be easier to solve for graph properties that are simpler to express. Phrased in terms of descriptive complexity theory, if we can describe a graph property using, say, a simple first-order formula, the corresponding vertex deletion problem should also be simple. The intuition was proven to be correct in 2020, when Fomin et al. [17] established a dichotomy based on the number of quantifier alternations that characterizes the classes of first-order logic formulas for which the vertex deletion problem is fixed-parameter tractable.

The results of Fomin et al. directly apply to some of the above examples: Consider the problem p-VERTEX-COVER, whose "triviality" property is described by the formula $\phi_{\text{vc}} = \forall x \forall y (x \not\sim y)$, or the problem p-CLUSTER-DELETION, whose triviality property is described by $\phi_{\text{cd}} = \forall x \forall y \forall z \big((x \sim y \land y \sim z) \to x \sim z\big)$. Both first-order formulas use *no quantifier alternations,* which by [17] already implies that the problems lie in para-P = FPT. Naturally, not all problems can be characterized so easily: Properties like acyclicity (which underlies the feedback vertex set problem) cannot be expressed in first-order logic and, thus, the results of Fomin et al. do not apply to them. Fomin et al. also show that if there are enough quantifier alternations (three, to be precise) in the first-order formulas describing the property, then the resulting vertex deletion problem can be W[1]-hard. Nevertheless, the descriptive approach allows us to identify large fragments of logical formulas and hence large classes of vertex deletion problems that are (at least fixed-parameter) tractable.

A first central question addressed in the present paper is whether the number of quantifier *alternations* (the property studied in [17]) overshadows all other aspects in making problems hard, or whether the individual quantifier pattern of the formula plays a significant role as well. This question appears to be of particular importance given that formulas describing natural problems (like ϕ_{vc} and ϕ_{cd} above) tend to have short and simple quantifier patterns: We might hope that even though we describe a particular triviality property using, say, four alternations, the fact that we use only, say, two existential quantifiers in total still assures us that the resulting vertex deletion problem is easy.

A second central question is whether the *kind* of graphs that we allow as inputs has an influence on the complexity of the problem. Intuitively, allowing only, say, *basic graphs* (simple undirected graphs without self-loops) should result in simpler problems than allowing directed graphs or even arbitrary logical structures as input. This intuition is known to be correct in the closely related question of deciding graph properties described in existential second-order logic. As we will see, in the context of vertex deletion problems it makes a difference whether we consider basic graphs, undirected graphs, or directed graphs, but not whether we consider directed graphs or arbitrary logical structures.

Our Contributions. We completely classify the parameterized complexity of vertex deletion problems in dependence of the quantifier pattern of the formulas that are used to express the triviality property and also in dependence of the kind of graphs that we allow as inputs (basic, undirected, directed, or arbitrary logical structures). An overview of the results is given in Table 1, where the following notations are used (detailed definitions are given later): For a first-order formula ϕ over the vocabulary $\tau = \{\sim^2\}$ of (directed, simple) graphs, the parameterized problem p_k-VERTEX-DELETION$_{\text{dir}}(\phi)$ (abbreviated p-VD$_{\text{dir}}(\phi)$) asks us to tell on input of a directed graph G and a parameter $k \in \mathbb{N}$ whether we can delete at most k vertices from G, so that for the resulting graph G' we have $G' \models \phi$. The problems p-VD$_{\text{undir}}(\phi)$ and p-VD$_{\text{basic}}(\phi)$ are the restrictions where the input graphs are undirected or basic graphs (undirected graphs without self-loops), respectively. For instance, p-VERTEX-COVER = p-VD$_{\text{basic}}(\phi_{\text{vc}})$ = p-VD$_{\text{basic}}\big(\forall x \forall y (x \not\sim y)\big)$. In the other direction, let p-VD$_{\text{arb}}(\phi)$ denote the generalization where we allow an arbitrary logical vocabulary τ and arbitrary (finite) logical structures $\mathcal{A}$ instead of just graphs G (and where "vertex deletion" should better be called "element deletion," but we stick with the established name). For a *(first-order) quantifier pattern p*, which is just a string of a's and e's standing for the universal and existential quantifiers at the beginning of a formula ϕ, we write p-VD$_{\text{basic}}(p)$ for the class of all problems p-VD$_{\text{basic}}(\phi)$ where ϕ has all its quantifiers at the beginning and they form the pattern p. For instance, p-VERTEX-COVER $\in$ p-VD$_{\text{basic}}(aa)$ as ϕ_{vc} has two universal quantifiers. The same notation is used for undirected graphs, directed graphs, and arbitrary structures.

Table 1 Complete complexity classification of vertex deletion problems for first-order formulas in dependence of the quantifier pattern $p \in \{a, e\}^*$ (where $p \preceq q$ means that p is a subsequence of q). The four different considered restrictions on the allowed input structures lead to three distinct complexity landscapes. Note that para-AC$^0 \subsetneq$ para-AC$^{0\uparrow} \subseteq$ para-P = FPT holds and that it is a standard assumption that FPT $\cap$ W[2]-hard $= \emptyset$ also holds.

p-VD$_{\text{basic}}(p)$	$\subseteq$ para-AC^0, when	$p \preceq e^*a^*$ or eae.
	$\not\subseteq$ para-AC^0 but $\subseteq$ para-AC$^{0\uparrow}$, when	$eeae$, aae or $aee \preceq p \preceq e^*a^*e^*$.
	$\cap$ W[2]-hard $\neq \emptyset$, when	$aea \preceq p$.
p-VD$_{\text{undir}}(p)$	$\subseteq$ para-AC^0, when	$p \preceq ae$ or e^*a^*.
	$\not\subseteq$ para-AC^0 but $\subseteq$ para-AC$^{0\uparrow}$, when	eae, aae or $aee \preceq p \preceq e^*a^*e^*$.
	$\cap$ W[2]-hard $\neq \emptyset$, when	$aea \preceq p$.
p-VD$_{\text{dir}}(p)$ and p-VD$_{\text{arb}}(p)$	$\subseteq$ para-AC^0, when	$p \preceq e^*a^*$.
	$\not\subseteq$ para-AC^0 but $\subseteq$ para-AC$^{0\uparrow}$, when	$ae \preceq p \preceq e^*a^*e^*$.
	$\cap$ W[2]-hard $\neq \emptyset$, when	$aea \preceq p$.

The results in Table 1 give an answer to the first central question formulated earlier, which asked whether it is the *number of alternations* of quantifiers in patterns (and not so much the actual number of quantifiers) that are responsible for the switch from tractable to intractable observed by Fomin et al. [17], or whether the frontier is formed by short patterns that "just happen" to have a certain number of alternations. As can be seen, the latter is true: All intractability results hold already for very short and simple patterns. Thus, while it was previously known that there is a formula in Π_3 (meaning it has a pattern of the form $\forall^*\exists^*\forall^*$ or $a^*e^*a^*$ in our notation) defining an intractable problem, we show that already one quantifier per alternation (the pattern aea) suffices. On the positive side, Table 1 shows that all vertex deletion problems that are (fixed-parameter) tractable at all already lie in the classes para-AC^0 or at least para-AC$^{0\uparrow}$. From an algorithmic point of view this means that all of the vertex deletion problems that we classify as fixed-parameter tractable admit efficient *parallel* fixed-parameter algorithms.

Concerning the second central question, which asked whether it makes a difference which kind of graphs or logical structures we consider, Table 1 also provides a comprehensive answer: First, the *frontier of tractability* (the patterns where we switch from membership in FPT to hardness for W[1]) *is the same for all kinds of inputs* (namely from "does not contain aea as a subsequence" to "contains aea as a subsequence"). Second, if we classify the tractable fragments further, a more complex complexity landscape arises: While p-$\mathrm{VD}_{\mathrm{dir}}(p)$ and p-$\mathrm{VD}_{\mathrm{arb}}(p)$ have the same classification for all p, the classes p-$\mathrm{VD}_{\mathrm{basic}}(p)$ and p-$\mathrm{VD}_{\mathrm{undir}}(p)$ each exhibit a different behavior. In other words: For simple patterns p, it makes a difference whether the inputs are basic, undirected, or directed graphs.

The just-discussed structural results are different from classifications in dependence of quantifier patterns p established in previous works: Starting with Eiter et al. [13] and subsequently Gottlob et al. [20], Tantau [27] and most recently Bannach et al. [3], different authors have classified the complexity of *weighted definability problems* by the quantifier patterns used to describe them. In these problems, formulas have a free set variable and we ask whether there is an assignment to the set variable with at most k elements such that the formula is true. Since it is easy to see that the vertex deletion problems we study are special cases of this question, upper bounds from earlier research also apply in our setting. However, our results show that (as one would hope) for vertex deletion problems for many patterns p we get better upper bounds than in the more general setting. Furthermore, there is an interesting structural insight related to our second central question: While the results in [3] for weighted definability show that, there, the complexities for undirected graphs, directed graphs, and arbitrary logical structures all coincide (but differ for basic graphs), for the vertex deletion setting, we get three different complexity characterizations for basic, undirected, and directed graphs – but the latter coincide with arbitrary structures once more.

Related Work. The complexity-theoretic investigation of vertex deletion problems has a long and fruitful history. Starting in classical complexity theory, results on vertex deletion problems were established as early as in the late 1970s [24, 25, 28]. The focus was mostly on deletion to commonly known graph properties, such as planarity, acyclicity or bipartiteness.

Since it is very natural to regard the number of allowed modifications as the parameter of the problem, the investigation of vertex deletion problems quickly gained traction in parameterized complexity, with continued research to this day [7, 19, 26]. Specifically for graphs, similar problems like the deletion or modification of edges [8] or alternative distance measures such as elimination distance [18] are also considered. Regarding first-order definable properties, a dichotomy is shown in [17].

The quantifier patterns we employ in this paper have also received a lot of attention, especially in the context of descriptive complexity. Early uses go as far back as the classification of decidable fragments of first-order logic [6]. They were then considered in the context of classical complexity [13, 20, 27] and later also in the context of parameterized complexity [3].

Organization of this Paper. Following a review of basic concepts and terminology in Section 2, we present the complexity-theoretic classification of the vertex deletion problems for basic, undirected and directed graphs in Sections 3, 4 and 5, respectively. For theorems and lemmas marked "▼ [4]", the proofs can be found in the full version [4].

2 Background in Descriptive and Parameterized Complexity

Terminology from Finite Model Theory. In this paper, we will use standard terminology from finite model theory, for a thorough introduction, see, for example [12]. A *relational vocabulary* τ (also known as a *signature*) is a set of *relation symbols* to each of which we assign a positive *arity,* denoted using a superscript. For example, $\tau = \{P^1, E^2\}$ is a relational vocabulary with a monadic relation symbol P and a dyadic relation symbol E. A τ*-structure* $\mathcal{A}$ consists of a *universe* A and for each relation symbol $R \in \tau$ of some arity r of a relation $R^{\mathcal{A}} \subseteq A^r$. We denote the set of *finite* τ*-structures* as $\textsc{struc}[\tau]$. For a first-order τ-sentence ϕ, we write $\textsc{models}(\phi)$ for the class of finite models of ϕ. A *decision problem* P is a subset of $\textsc{struc}[\tau]$ which is closed under isomorphisms. A formula ϕ *describes* P if $\textsc{models}(\phi) = P$.

For τ-structures $\mathcal{A}$ and $\mathcal{B}$ with universes A and B, respectively, we say that $\mathcal{A}$ is an *induced substructure* of $\mathcal{B}$ if $A \subseteq B$ and for all r-ary $R \in \tau$, we have $R^{\mathcal{A}} = R^{\mathcal{B}} \cap A^r$. For a set $S \subseteq B$, we denote by $\mathcal{B} \setminus S$ the substructure induced on $B \setminus S$.

We regard *directed* graphs $G = (V, E)$ (which are pairs of a nonempty vertex set V and an edge relation $E \subseteq V \times V$) as logical structures $\mathcal{G}$ over the vocabulary $\tau_{\text{digraph}} = \{\sim^2\}$ where V is the universe and $\sim^{\mathcal{G}} = E$. An *undirected* graph is a directed graph that additionally satisfies $\phi_{\text{undirected}} := \forall x \forall y (x \sim y \to y \sim x)$, while a *basic* graph satisfies $\phi_{\text{basic}} := \forall x \forall y \bigl(x \sim y \to (y \sim x \land x \neq y)\bigr)$.

For a first-order logic formula in prenex normal form (meaning all quantifiers are at the front), we can associate a *quantifier prefix pattern* (or *pattern* for short), which are words over the alphabet $\{e, a\}$.[1] For example, the formula ϕ_{basic} has the pattern aa, while the formula $\phi_{\text{degree-}\geq 2} := \forall x \exists y_1 \exists y_2 \bigl((x \sim y_1) \land (x \sim y_2) \land (y_1 \neq y_2)\bigr)$ has the pattern aee. As another example, the formulas in the class Π_2 (which start with a universal quantifier and have one alternation) are exactly the formulas with a pattern $p \in \{a\}^* \circ \{e\}^*$, which we write briefly as $p \in a^* e^*$. We write $p \preceq q$ if p is a subsequence of q.

Terminology from Parameterized Complexity. We use standard definitions from parameterized complexity, see for instance [9, 10, 16]. A *parameterized problem* is a set $Q \subseteq \Sigma^* \times \mathbb{N}$ for an alphabet Σ. In an *instance* $(x, k) \in \Sigma^* \times \mathbb{N}$ we call x the *input* and k the *parameter.* The central problem we consider in this paper is the following:

▶ **Problem 2.1** ($\text{p-}\textsc{vd}_{\text{arb}}(\phi)$, where ϕ is a first-order τ-formula).

Instance: *(An encoding of) a logical τ-structure $\mathcal{A}$ and an integer $k \in \mathbb{N}$.*
Parameter: *k.*
Question: *Is there a set $S \subseteq A$ with $|S| \leq k$ such that $\mathcal{A} \setminus S \models \phi$?*

As mentioned earlier, we also consider the problems $\text{p-}\textsc{vd}_{\text{basic}}(\phi)$, where the input structures are basic graphs (formally, $\text{p-}\textsc{vd}_{\text{basic}}(\phi) = \text{p-}\textsc{vd}_{\text{arb}}(\phi) \cap \bigl(\textsc{models}(\phi_{\text{basic}}) \times \mathbb{N}\bigr)$), the problems $\text{p-}\textsc{vd}_{\text{undir}}(\phi)$, where the input structures are undirected graphs, and $\text{p-}\textsc{vd}_{\text{dir}}(\phi)$, where the input structures are directed graphs. For a pattern $p \in \{a, e\}^*$, the class $\text{p-VD}_{\text{arb}}(p)$ contains all problems $\text{p-}\textsc{vd}_{\text{arb}}(\phi)$ such that ϕ has pattern p. The classes with the subscripts “basic”, “undir”, and “dir” are defined similarly.

[1] One uses “a” and “e” in patterns rather than “$\forall$” and “$\exists$” since in the context of second-order logic one needs a way to differentiate between first-order and second-order quantifiers and, there, “E” refers to a “second-order $\exists$” while “e” refers to a “first-order $\exists$”. In our paper, we only use first-order quantifiers so only lowercase letters are needed.

We will consider some parameterized circuit complexity classes. We define para-AC^0 as the class of parameterized problems that can be decided by a family of unbounded fan-in circuits $(C_{n,k})_{n,k\in\mathbb{N}}$ of constant depth and size $f(k)\cdot n^{O(1)}$ for some computable function f. Similarly, para-FAC^0 is the class of functions that can be computed by a family of unbounded fan-in circuits $(C_{n,k})_{n,k\in\mathbb{N}}$ of constant depth and size $f(k)\cdot n^{O(1)}$ for some computable function f. For para-AC$^{0\uparrow}$, we allow the circuit to have depth $f(k)$. Questions of uniformity will not be important in the present paper. For these classes, we have the following inclusions: para-AC^0 $\subsetneq$ para-AC$^{0\uparrow}$ $\subseteq$ para-P = FPT.

A parameterized problem $Q\subseteq\Sigma^*\times\mathbb{N}$ is para-AC^0-many-one-reducible to a problem $Q'\subseteq\Gamma^*\times\mathbb{N}$, written $Q\leq_{\text{m}}^{\text{para-AC}^0} Q'$, if there is a function $f\colon\Sigma^*\times\mathbb{N}\to\Gamma^*\times\mathbb{N}$, such that (1) for all $(x,k)\in\Sigma^*\times\mathbb{N}$ we have $(x,k)\in Q$ iff $f(x,k)\in Q'$, (2) there is a computable function $g\colon\mathbb{N}\to\mathbb{N}$ such that for all $(x,k)\in\Sigma^*\times\mathbb{N}$, we have $k'\leq g(k)$, where $f(x,k)=(x',k')$, and (3) $f\in$ para-FAC^0. The more general para-AC^0 disjunctive truth table reduction, written $Q\leq_{\text{dtt}}^{\text{para-AC}^0} Q'$, is defined similarly, only f maps (x,k) to a sequence $(x_1,k_1),\dots,(x_\ell,k_\ell)$ of instances such that (1$'$) $(x,k)\in Q$ iff there is an $i\in\{1,\dots,\ell\}$ with $(x_i,k_i)\in Q'$ and (2$'$) $k_i\leq g(k)$ holds for all $i\in\{1,\dots,\ell\}$. Both para-AC^0 and para-AC$^{0\uparrow}$ are closed under $\leq_{\text{m}}^{\text{para-AC}^0}$- and $\leq_{\text{dtt}}^{\text{para-AC}^0}$-reductions.

3 Basic Graphs

Basic graphs, that is, undirected graphs without self-loops, are one of the simplest non-trivial logical structures one can imagine. Despite that, many NP-hard problems on graphs, like vertex cover, clique or dominating set, are NP-hard even for basic graphs. This also transfers in some sense to our setting: The "tractability frontier", the dividing line between the fragments which are tractable and those where we can express intractable problems, is the same for all graph classes we consider. However, when we shift our attention to the complexity landscape inside the tractable fragments, we also see that the complexity of the logical structure has an impact on the complexity of the problems we can define: Basic, undirected, and directed graphs all have provably distinct complexity characterizations.

We begin by stating the main theorem of the section, the complexity classification for basic graphs. In the rest of the section, we show the upper and lower bounds that lead to this classification.

▶ **Theorem 3.1** (Complexity Trichotomy for p-VD$_{\text{basic}}(p)$). *Let $p\in\{a,e\}^*$ be a pattern.*

1. p-VD$_{\text{basic}}(p)$ $\subseteq$ para-AC^0, *if $p\preceq eae$ or $p\preceq e^*a^*$.*
2. p-VD$_{\text{basic}}(p)$ $\subseteq$ para-AC$^{0\uparrow}$ *but* p-VD$_{\text{basic}}(p)$ $\not\subseteq$ para-AC^0, *if $eeae\preceq p$, $aae\preceq p$ or $aee\preceq p$ holds, but also still $p\preceq e^*a^*e^*$.*
3. p-VD$_{\text{basic}}(p)$ *contains a* W[2]*-hard problem, if $aea\preceq p$.*

The theorem covers all possible patterns. It follows from the following lemma, where we state the individual complexity characterizations we will prove:

▶ **Lemma 3.2** (Detailed Bounds for p-VD$_{\text{basic}}(p)$).

1. p-VD$_{\text{basic}}(eae)$ $\subseteq$ para-AC^0.
2. p-VD$_{\text{basic}}(e^*a^*)$ $\subseteq$ p-VD$_{\text{arb}}(e^*a^*)$ $\subseteq$ para-AC^0.
3. p-VD$_{\text{basic}}(e^*a^*e^*)$ $\subseteq$ p-VD$_{\text{arb}}(e^*a^*e^*)$ $\subseteq$ para-AC$^{0\uparrow}$.
4. p-VD$_{\text{basic}}(eeae)$ *contains a problem not in* para-AC^0.
5. p-VD$_{\text{basic}}(aae)$ *contains a problem not in* para-AC^0.
6. p-VD$_{\text{basic}}(aee)$ *contains a problem not in* para-AC^0.
7. p-VD$_{\text{basic}}(aea)$ *contains a* W[2]*-hard problem.*

Notice that in particular, we know unconditionally that $\mathrm{W}[2] \not\subseteq \text{para-AC}^0$, and, hence, a W[2]-hard problem cannot lie in para-AC^0. It is furthermore widely conjectured that $\mathrm{W}[2] \not\subseteq \text{para-AC}^{0\uparrow}$, as $\text{para-AC}^{0\uparrow} \subseteq \text{FPT}$. We devote the rest of this section to proving the individual items of the lemma.

Upper Bounds. Previous work by Bannach et al. [3] showed that in the weighted definability setting, formulas with the pattern ae already suffice to describe W[2]-hard problems. We now show that the situation is more favorable in the vertex deletion setting, which is a special case of weighted definability: All problems in $\text{p-VD}_{\text{basic}}(e^*a^*e^*)$ are tractable and the problems in $\text{p-VD}_{\text{basic}}(e^*a^*)$ and in $\text{p-VD}_{\text{basic}}(eae)$ are even in para-AC^0, the smallest class commonly considered in parameterized complexity. We start with the last claim:

▶ **Lemma 3.3** (▼ [4]). $\text{p-VD}_{\text{basic}}(eae) \subseteq \text{para-AC}^0$.

Proof idea. To check whether we can delete at most k vertices to satisfy a formula with prefix pattern eae, we first branch over the possible assignments to the first existentially quantified variable. Now, the neighborhood of this variable induces a 2-coloring on the rest of the graph. For the rest of the prefix, ae, we prove that a vertex has to be deleted if and only if there is no special set of constant size, called *stable set*. This can all be checked in para-AC^0. ◀

Since the algorithms used to prove the next two upper bounds do not make use of the fact that the input structure is a basic graph, we prove them for arbitrary input structures.

▶ **Lemma 3.4.** $\text{p-VD}_{\text{arb}}(e^*a^*) \subseteq \text{para-AC}^0$.

Proof. For a given formula ϕ of the form $\exists x_1 \cdots \exists x_f \forall y_1 \cdots \forall y_g(\psi)$ for a quantifier-free formula ψ, we show that $\text{p-VD}_{\text{arb}}(\phi) \leq_{\text{dtt}}^{\text{para-AC}^0}$ p-g-HITTING-SET, where the hitting set problem is defined as shown below. Since p-g-HITTING-SET is known [5] to lie in para-AC^0, we get the claim.

▶ **Problem 3.5** (p-d-HITTING-SET for fixed $d \in \mathbb{N}$).

Instance: *A universe U and a set E of subsets $e \subseteq U$ (called* hyperedges*) with $|e| \leq d$ for all $e \in E$, and a number k.*

Parameter: k.

Question: *Is there a* hitting set *$X \subseteq V$, meaning that $X \cap e \neq \emptyset$ holds for all $e \in E$, with $|X| \leq k$?*

For an arbitrary input structure $\mathcal{A}$ with universe A, we proceed as follows: For the existentially bound variables x_1 to x_f we consider all possible assignments to them in parallel. For each of these, we prepare a query to the hitting set problem, resulting in n^f queries in total. For a given assignment, which fixes each x_i to some constant c_i, replace each occurrence of x_i in ϕ by c_i. Build a hitting set instance H as follows: The universe is $A \setminus \{c_1, \dots, c_f\}$. For each assignment $(d_1, \dots, d_g)$ of to the g universally quantified variables, check if the formula ψ is true, that is, whether $\mathcal{A} \models \psi(c_1, \dots, c_f, d_1, \dots, d_g)$. If this is not the case, add the hyperedge $\{d_1, \dots, d_g\} \setminus \{c_1, \dots, c_f\}$ to make sure that at least one element is deleted from the universe of $\mathcal{A}$ that cause this particular violation. If $\{d_1, \dots, d_g\} \setminus \{c_1, \dots, c_f\}$ is empty, an empty hyperedge is generated and the hitting set solver correctly rejects the input.

We claim that $\mathcal{A} \in \text{p-VD}_{\text{arb}}(\phi)$ iff for at least one of the constructed H we have $(H, k) \in$ p-g-HITTING-SET: For the first direction, let S with $|S| \leq k$ be the elements of $\mathcal{A}$'s universe that we can delete, that is, for which $\mathcal{A} \setminus S \models \phi$. Then there are constants $(c_1, \dots, c_f)$ that we can assign to the existentially bound variables such that $\mathcal{A} \setminus S \models \forall y_1 \cdots \forall y_g \big(\psi(c_1, \dots, c_f, y_1, \dots, y_g)\big)$. But, then, S is a hitting set of the instance corresponding to these constants: If there were an edge $e \subseteq A$ with $e \cap S = \emptyset$ in the hitting set instance, there would be an assignment to the y_i to elements in $A \setminus S$ that makes ψ false, violating the assumption.

For the other direction, let X with $|X| \leq k$ be the solution of one of the produced hitting set instances with $(H, k) \in$ p-g-HITTING-SET (at least one must exist). Then $\mathcal{A} \setminus X \models \phi$, since we can assign the existentially bound variables to the values that correspond to H (which will not be in X by construction) and there can be no assignment to the universally quantified variables that makes ψ false as any assignment where this would be case is hit by X by construction and, thus, at least one element of the tuple that causes the violation gets removed in $\mathcal{A} \setminus X$. ◀

▶ **Lemma 3.6.** p-$\mathrm{VD}_{\mathrm{arb}}(e^*a^*e^*) \subseteq$ para-$\mathrm{AC}^{0\uparrow}$.

Proof. Let ϕ be fixed and of the form $\exists x_1 \cdots \exists x_f \forall y_1 \cdots \forall y_g \exists z_1 \cdots \exists z_h(\psi)$ for a quantifier-free formula ψ. We describe a para-$\mathrm{AC}^{0\uparrow}$-algorithm that, given an arbitrary input structure $\mathcal{A}$ with universe A, decides whether there is a set S with $|S| \leq k$ such that $\mathcal{A} \setminus S \models \phi$.

Now, we have for each assignment to the universally quantified variables a witness which is bound by the block of h existential quantifiers. The problem compared to the e^*a^*-fragment is that by the deletion of elements, we could potentially destroy witnesses needed to satisfy other assignments. Because of this, we use a direct search tree algorithm to resolve violations of the universal quantifiers.

In detail, we once more consider all possible assignments $(c_1, \dots, c_f)$ to the x_i in parallel. Then we use k layers to find and resolve violations: At the start of each layer, we will already have fixed a set D of vertices that we wish to delete, starting in the first layer with $D = \emptyset$. Then in the layer, we find the (for example, lexicographically) first assignment of the y_i to elements $(d_1, \dots, d_f)$ that all lie in $A \setminus D$ for which we cannot find an assignment of the z_i to elements $(e_1, \dots, e_h)$ in $A \setminus D$ such that $\mathcal{A} \setminus D \models \psi(c_1, \dots, c_f, d_1, \dots, d_g, e_1, \dots, e_h)$. When we cannot find such an assignment, we can accept since we have found a D for which $\mathcal{A} \setminus D \models \phi$ holds. Otherwise, we *have to* delete one of the elements in $\{d_1, \dots, d_g\} \setminus \{c_1, \dots, c_f\}$ to make the formula true, so we branch over these at most g possibilities, entering g copies of the next layers, where the ith copy starts with $D \cup \{d_i\}$.

Since the block of universal quantifiers has constant length, the number of branches in each level of the search tree is constant, so the total size of the search tree is at most g^k. The depth of the search tree is bounded by the number of vertices we can delete, which is our parameter. In total, we get a para-$\mathrm{AC}^{0\uparrow}$ circuit. ◀

Lower Bounds. We now go on to show the lower bounds claimed in Lemma 3.2. The next lemmas all follow the same rough strategy: To show that some problems that can be expressed in the given fragments are (unconditionally) not in para-AC^0, we reduce from a variant of the reachability problem. In contrast, the last lower bound is obtained via a reduction from p-SET-COVER, and improves a result from Fomin et al. [17]. They establish that there is a formula $\phi \in \Pi_3$, such that p-$\mathrm{VD}_{\mathrm{basic}}(\phi)$ is W[2]-hard. In terms of patterns, the formula they construct has the pattern $a^5e^{26}a$. We show that there is a formula with pattern aea for which this holds.

The reachability problem that will be central for the following lower bounds is:

▶ **Problem 3.7** (p-MATCHED-REACH).

Instance: *A directed layered graph G with vertex set $\{1, \dots, n\} \times \{1, \dots, k\}$, where the ith layer is $V_i := \{1, \dots, n\} \times \{i\}$, such that for each $i \in \{1, \dots, k-1\}$ the edges point to the next layer and they form a perfect matching between V_i and V_{i+1}; and two designated vertices $s \in V_1$ and $t \in V_k$.*
Parameter: *k.*
Question: *Is t reachable from s in G?*

(We require that in the encoding of G the vertex "addresses" (i, l) are given explicitly as, say, pairs of binary numbers, so that even a AC^0 circuit will have no trouble determining which vertices belong to a layer V_i or what the number k of layers is.)

Observe that the input instance can be alternatively described as a collection of n directed paths, each of length k. We call the paths in this graph *original paths* with *original vertices and edges*. We call the vertices in the layers V_1 and V_k the *outer vertices* and the vertices in the layers V_i for $i \in \{2, \dots, k-1\}$ the *inner vertices*. The reductions add vertices and edges to the graphs, which will be referred to as the *new vertices and edges* (and will be indicated in yellow in figures).

▶ **Fact 3.8** ([3]). *p-MATCHED-REACH $\notin$ para-AC^0 and, thus, for any problem Q with p-MATCHED-REACH $\leq_{\mathrm{m}}^{\text{para-AC}^0}$ Q we have $Q \notin$ para-AC^0.*

The proof of every lemma using a reduction from the matched reachability problem will consist of four parts:

1. The construction of a formula ϕ with the quantifier pattern p given in the lemma.
2. The construction of the instance for the vertex deletion problem (G', k') from the input instance of the matched reachability problem (G, s, t) (typically by adding new vertices and edges).
3. Showing $(G, s, t) \in$ p-MATCHED-REACH implies $(G', k') \in \text{p-VD}_{\text{basic}}(\phi)$, called the *forward direction.*
4. Showing $(G', k') \in \text{p-VD}_{\text{basic}}(\phi)$ implies $(G, s, t) \in$ p-MATCHED-REACH, called the *backward direction.*

We present the application of the above steps in detail in the following lemma. In subsequent lemmas, which follow the same line of arguments, but with appropriate variations in the constructions and correctness proofs, we only highlight the differences.

▶ **Lemma 3.9.** $\text{p-VD}_{\text{basic}}(eeae) \not\subseteq \text{para-AC}^0$.

Proof. We want there to be a deletion strategy for (G', k') iff in the instance (G, s, t), the vertices s and t lie on the same original path. We take $k' = k$, the number of layers in G, and construct a graph G' from G by adding two special vertices c_1 and c_2, and regard the adjacency of every vertex on the original paths to the vertices c_1 and c_2 as a 3-coloring with colors $i \in \{0, 1, 2\}$. We then add appropriate gadgets at the start and the end of each original path, with special gadgets being added at s and at t (although, in this proof, their "special gadgets" are just the empty gadget).

The formula. Consider the following formulas, where ϕ_a specifies that every vertex that is neither c_1 nor c_2 should be connected in a certain way to them, and ϕ_b asks that every vertex of color i should have a neighbor of color $(i-1) \pmod 3$. We encode the color 0 with $(x \sim c_1 \land x \not\sim c_2)$, the color 1 with $(x \not\sim c_1 \land x \sim c_2)$, and the color 2 with $(x \sim c_1 \land x \sim c_2)$.

$$
\begin{aligned}
\phi_a(c_1,c_2,x) &= (c_1 \neq c_2) \wedge (c_1 \sim x \vee c_2 \sim x)\\
\phi_b(c_1,c_2,x,y) &= x \sim y \wedge \big((x \sim c_1 \wedge x \sim c_2) \to (y \not\sim c_1 \wedge y \sim c_2)\big)\\
&\qquad \wedge \big((x \not\sim c_1 \wedge x \sim c_2) \to (y \sim c_1 \wedge y \not\sim c_2)\big)\\
&\qquad \wedge \big((x \sim c_1 \wedge x \not\sim c_2) \to (y \sim c_1 \wedge y \sim c_2)\big)\\
\phi_{3.9} &= \exists c_1 \exists c_2 \forall x \exists y \Big(\big((x \neq c_1) \wedge (x \neq c_2)\big) \to\\
&\qquad\qquad \big((y \neq c_1) \wedge (y \neq c_2) \wedge\\
&\qquad\qquad\; \phi_a(c_1,c_2,x) \wedge \phi_b(c_1,c_2,x,y)\big)\Big)
\end{aligned}
$$

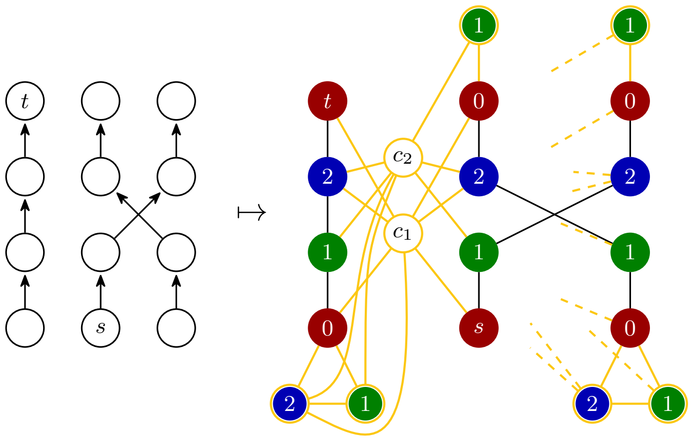


Figure 1 Example for the reduction from Lemma 3.9. The input graph on the left is a directed layered graph with perfect matchings between consecutive layers. The reduction maps it to the undirected graph shown right by forgetting about the direction of edges, by adding gadgets at the beginnings and ends of the paths (with special empty gadgets at s and t), and by adding two special vertices c_1 and c_2 that are connected in three different ways to the other vertices, corresponding to three different colors. Newly added vertices and edges are indicated in yellow. Note that the indicated colors, numbers, and labels are not part of the output, they are only for explaining how the formula interprets the connection of the vertices to c_1 and c_2.

The reduction. On input (G, s, t) the reduction first checks that the graph is, indeed, a layered graph with perfect matchings between consecutive levels (this can easily be done by an AC^0 circuit due to the way we encode G). Then, we let k' be the number k of layers in $G = (V, \sim)$ and construct $G' = (V', \sim')$ by first forgetting about the direction of the edges (making the graph undirected). We then add the following gadgets:

1. At each end $v \in V_k$ of a path, *except for* $v = t$, we add a vertex v' to V' and connect v to v', so $v \sim' v'$. Let V_{k+1} be the set of all new vertices added in this way. The gadget for $t \in V_k$ is empty: We do not add anything.
2. At each beginning $v \in V_1$ of a path, *except for* $v = s$, add two vertices v' and v'' to V' and connect the three vertices to a triangle, so $v \sim' v' \sim' v'' \sim' v$. Let V_0 contain all vertices v' added in this way and let V_{-1} contain all vertices v'' added in this way. Once more, the special gadget for $s \in V_1$ is just the empty gadget.
3. Finally, we add two further vertices c_1 and c_2 and connect them to the other vertices as follows: For $v \in V_i$ with $i \in \{-1, 0, 1, 2, \ldots, k + 1\}$:

- If $i \equiv 0 \mod 3$, let $c_1 \sim' v$.
- If $i \equiv 1 \mod 3$, let $c_2 \sim' v$.
- If $i \equiv 2 \mod 3$, let $c_1 \sim' v$ and $c_2 \sim' v$.

An example for the reduction is depicted in Figure 1. We claim that through this construction, the instance (G', k') is in p-VD$_{\text{basic}}(\phi_{3.9})$ iff the input graph with vertices s and t is in p-MATCHED-REACH:

Forward direction. Suppose that $(G, s, t) \in$ p-MATCHED-REACH. We show that $(G', k') \in$ p-VD$_{\text{basic}}(\phi_{3.9})$: In input G', just delete every vertex in the original s-t-path. Then every vertex $v \in V_i$ for $i \in \{2, \dots, k\}$ has its predecessor in the original path as a neighbor, and the predecessor has the previous color regarding the ordering. Furthermore, every vertex $v \in V_1$ is part of a triangle where the three vertices each have a different color, so every one of these three vertices has a neighbor of the previous color.

Backward direction. Suppose that $(G', k') \in$ p-VD$_{\text{basic}}(\phi_{3.9})$. We show that $(G, s, t) \in$ p-MATCHED-REACH. By assumption, there is a set D of size $|D| \leq k = k'$ such that $G' \setminus D$ is a model of $\phi_{3.9}$. Observe that $c_1 \notin D$ and $c_2 \notin D$ must hold since they are the only vertices satisfying the formula part ϕ_a, which requires that there are two different vertices that are connect to everyone else. On the other hand, we *have to* delete s, since by construction, it has no neighbor with the previous color (s has color 0, the successor of s has color 1). But, now, the successor of s has no neighbor of the previous color, so we have to delete it as well. We have to continue for the whole original path of s, so D has to contain at least the vertices on the original path starting at s, which encompasses k vertices. If the last vertex $v \in V_k$ on the original path starting at s is not t (that is, if t is not reachable from s), then there is another vertex $v' \in V_{k+1}$ with $v \sim' v'$ and we also have to delete v', contradicting the assumption that we only have to delete k vertices. Thus, t must be reachable from s. ◀

▶ **Lemma 3.10** (▼ [4]). p-VD$_{\text{basic}}(aae) \not\subseteq$ para-AC^0.

▶ **Lemma 3.11** (▼ [4]). p-VD$_{\text{basic}}(aee) \not\subseteq$ para-AC^0.

▶ **Lemma 3.12** (▼ [4]). p-VD$_{\text{basic}}(aea)$ *contains a* W[2]*-hard problem.*

4 Undirected Graphs

Whether allowing self-loops has an impact on the complexity of the problems is hard to predict: While in the setting of Fomin et al. [17], the same dichotomy arises for basic and undirected graphs, in the setting of weighted definability considered by Bannach et al. [3], one class of problems jumps from being contained in para-AC^0 to containing para-NP-hard problems just by allowing self-loops. In our setting, we get an *intermediate* blow-up of the complexities by allowing self-loops: While the tractability frontier stays the same, the frontier of fragments that are solvable in para-AC^0 shifts.

Let us now classify the complexity of vertex deletion problems on undirected graphs. We can use some of the upper and lower bounds established in the section before, and only consider the differences.

▶ **Theorem 4.1** (Complexity Trichotomy for p-VD$_{\text{undir}}(p)$). *Let $p \in \{a, e\}^*$ be a pattern.*

1. p-VD$_{\text{undir}}(p) \subseteq$ para-AC^0*, if $p \preceq ae$ or $p \preceq e^*a^*$.*
2. p-VD$_{\text{undir}}(p) \subseteq$ para-AC$^{0\uparrow}$ *but* p-VD$_{\text{undir}}(p) \not\subseteq$ para-AC^0*, if one of $eae \preceq p$, $aae \preceq p$ or $aee \preceq p$ holds, but still $p \preceq e^*a^*e^*$ holds.*
3. p-VD$_{\text{undir}}(p)$ *contains a* W[2]*-hard problem, if $aea \preceq p$.*

▶ **Lemma 4.2.**

1. $\text{p-VD}_{\text{undir}}(ae) \subseteq \text{para-AC}^0$.
2. $\text{p-VD}_{\text{undir}}(e^*a^*) \subseteq \text{para-AC}^0$.
3. $\text{p-VD}_{\text{undir}}(e^*a^*e^*) \subseteq \text{para-AC}^{0\uparrow}$.
4. $\text{p-VD}_{\text{undir}}(eae)$ *contains a problem not in* para-AC^0.
5. $\text{p-VD}_{\text{undir}}(aae)$ *contains a problem not in* para-AC^0.
6. $\text{p-VD}_{\text{undir}}(aee)$ *contains a problem not in* para-AC^0.
7. $\text{p-VD}_{\text{undir}}(aea)$ *contains a* $\text{W}[2]$*-hard problem.*

Proof. Item 1 is proven below in Lemma 4.3. Items 2 and 3 follow directly from Lemmas 3.4 and 3.6. Item 4 is proven below in Lemma 4.4, Item 5 follows from Lemma 3.10, Item 6 from Lemma 3.11 and Item 7 from Lemma 3.12. ◀

▶ **Lemma 4.3** (▼ [4]). $\text{p-VD}_{\text{undir}}(ae) \subseteq \text{para-AC}^0$.

▶ **Lemma 4.4** (▼ [4]). $\text{p-VD}_{\text{undir}}(eae) \nsubseteq \text{para-AC}^0$.

5 Directed Graphs and Arbitrary Structures

The final class of logical structures we investigate in this paper are directed graphs. Interestingly, from the viewpoint of quantifier patterns, this class of structures is as complex as arbitrary logical structures.

▶ **Theorem 5.1** (Complexity Trichotomy for $\text{p-VD}_{\text{dir}}(p)$). *Let $p \in \{a, e\}^*$ be a pattern.*

1. $\text{p-VD}_{\text{dir}}(p) \subseteq \text{para-AC}^0$, *if* $p \preceq e^*a^*$.
2. $\text{p-VD}_{\text{dir}}(p) \subseteq \text{para-AC}^{0\uparrow}$ *but* $\text{p-VD}_{\text{dir}}(p) \nsubseteq \text{para-AC}^0$, *if* $ae \preceq p \preceq e^*a^*e^*$.
3. $\text{p-VD}_{\text{dir}}(p)$ *contains a* $\text{W}[2]$*-hard problem, if* $aea \preceq p$.

▶ **Lemma 5.2.**

1. $\text{p-VD}_{\text{dir}}(e^*a^*) \subseteq \text{para-AC}^0$.
2. $\text{p-VD}_{\text{dir}}(e^*a^*e^*) \subseteq \text{para-AC}^{0\uparrow}$.
3. $\text{p-VD}_{\text{dir}}(ae)$ *contains a problem not in* para-AC^0.
4. $\text{p-VD}_{\text{dir}}(aea)$ *contains a* $\text{W}[2]$*-hard problem.*

Proof. Items 1 and 2 follow directly from Lemmas 3.4 and 3.6. Item 3 is shown in Lemma 5.3, and Item 4 follows from Lemma 3.12. ◀

▶ **Lemma 5.3** (▼ [4]). $\text{p-VD}_{\text{dir}}(ae) \nsubseteq \text{para-AC}^0$.

6 Conclusion

In this paper, we fully classified the parameterized complexity of vertex deletion problems where the target property is expressible by first-order formulas and where the inputs are basic graphs, undirected graphs, directed graphs, or arbitrary logical structures. The classification is based on the quantifier patterns of the formulas, and sheds additional light on the complexity properties that emerge from these patterns: We have seen that while the tractability barrier is the same for all logical structures, $\text{p-VD}_{\text{basic}}(e^*a^*e^*)$, $\text{p-VD}_{\text{undir}}(e^*a^*e^*)$, $\text{p-VD}_{\text{dir}}(e^*a^*e^*)$ and $\text{p-VD}_{\text{arb}}(e^*a^*e^*)$ all being tractable and $\text{p-VD}_{\text{basic}}(aea)$, $\text{p-VD}_{\text{undir}}(aea)$, $\text{p-VD}_{\text{dir}}(aea)$ as well as $\text{p-VD}_{\text{arb}}(aea)$ all containing intractable problems, in the tractable cases, basic, undirected and directed graphs have provably different complexities, the latter coinciding with arbitrary structures.

The granularity we gained with the viewpoint of quantifier patterns could be useful to examine the complexity of vertex deletions problems where the property is given by a formula of a more expressive logic: For both *monadic second-order logic* (MSO) and *existential second-order logic* (ESO), even the model checking problem becomes NP-hard. This would allow us to express many more natural problems such as feedback vertex set, that have no obvious formalization as a vertex deletion problem to plain FO-properties. Similarly, we could allow extensions such as transitive closure or fixed point operators.

Compared to previous work on weighted definability, where the objective is to instantiate a free set variable with *at most, exactly,* or *at least* k elements such that a formula holds, we only considered deleting *at most* k elements. How does the complexity of vertex deletion problems change, if we *have to* delete exactly k elements – or, for that matter, *at least* k elements?

References

1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, USA, January 10, 2004*, pages 62–69, 2004.

2 Akanksha Agrawal and M. S. Ramanujan. Distance from triviality 2.0: Hybrid parameterizations. In *Combinatorial Algorithms - 33rd International Workshop, IWOCA 2022, Trier, Germany, June 7–9, 2022, Proceedings*, pages 3–20, 2022. doi:10.1007/978-3-031-06678-8_1.

3 Max Bannach, Florian Chudigiewitsch, and Till Tantau. Existential second-order logic over graphs: Parameterized complexity. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.3.

4 Max Bannach, Florian Chudigiewitsch, and Till Tantau. On the descriptive complexity of vertex deletion problems. Technical Report abs/2406.18299, Cornell University, 2023. arXiv:2406.18299.

5 Max Bannach and Till Tantau. Computing hitting set kernels by AC^0-circuits. *Theory of Computing Systems*, 64(3):374–399, 2020. doi:10.1007/s00224-019-09941-z.

6 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.

7 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *Journal of the ACM*, 55(5), November 2008. doi:10.1145/1411509.1411511.

8 Christophe Crespelle, Pål Grønås Drange, Fedor V. Fomin, and Petr Golovach. A survey of parameterized algorithms and the complexity of edge modification. *Computer Science Review*, 48:100556, 2023. doi:10.1016/j.cosrev.2023.100556.

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 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

11 Maël Dumas and Anthony Perez. An improved kernelization algorithm for trivially perfect editing. In *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6–8, 2023, Amsterdam, The Netherlands*, pages 15:1–15:17, 2023. doi:10.4230/LIPICS.IPEC.2023.15.

12 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Springer, 2nd edition, 2005. doi:10.1007/3-540-28788-4.

13 Thomas Eiter, Yuri Gurevich, and Georg Gottlob. Existential second-order logic over strings. *Journal of the ACM*, 47(1):77–131, 2000. doi:10.1145/331605.331609.

14 Damir Ferizovic, Demian Hespe, Sebastian Lamm, Matthias Mnich, Christian Schulz, and Darren Strash. Engineering kernelization for maximum cut. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5–6, 2020*, pages 27–41, 2020. doi:10.1137/1.9781611976007.3.

15 Aleksander Figiel, Vincent Froese, André Nichterlein, and Rolf Niedermeier. There and back again: On applying data reduction rules by undoing others. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5–9, 2022, Berlin/Potsdam, Germany*, pages 53:1–53:15, 2022. doi:10.4230/LIPICS.ESA.2022.53.

16 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006. doi:10.1007/3-540-29953-X.

17 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. On the parameterized complexity of graph modification to first-order logic properties. *Theory of Computing Systems*, 64(2):251–271, 2020. doi:10.1007/s00224-019-09938-8.

18 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Parameterized complexity of elimination distance to first-order logic properties. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470540.

19 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting topological minors is FPT. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1317–1326, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384318.

20 Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *Journal of the ACM*, 51(2):312–362, 2004. doi:10.1145/972639.972646.

21 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14–17, 2004, Proceedings*, pages 162–173, 2004. doi:10.1007/978-3-540-28639-4_15.

22 Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. Wegotyoucovered: The winning solver from the PACE 2019 challenge, vertex cover track. In *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing, CSC 2020, Seattle, USA, February 11–13, 2020*, pages 1–11, 2020. doi:10.1137/1.9781611976229.1.

23 Demian Hespe, Christian Schulz, and Darren Strash. Scalable kernelization for maximum independent sets. *ACM Journal of Experimental Algorithmics*, 24(1):1.16:1–1.16:22, 2019. doi:10.1145/3355502.

24 M. S. Krishnamoorthy and Narsingh Deo. Node-deletion NP-complete problems. *SIAM Journal on Computing*, 8(4):619–625, 1979. doi:10.1137/0208049.

25 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

26 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in O*(2.7k) time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 971–989, 2020. doi:10.1137/1.9781611975994.58.

27 Till Tantau. Existential second-order logic over graphs: A complete complexity-theoretic classification. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4–7, 2015, Garching, Germany*, pages 703–715, 2015. doi:10.4230/LIPIcs.STACS.2015.703.

28 Mihalis Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 253–264, New York, NY, USA, 1978. Association for Computing Machinery. doi:10.1145/800133.804355.

Sparse Graphic Degree Sequences Have Planar Realizations

Amotz Bar-Noy
City University of New York (CUNY), NY, USA

Toni Böhnlein
Huawei, Zurich, Switzerland

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Yingli Ran
Weizmann Institute of Science, Rehovot, Israel

Dror Rawitz
Bar Ilan University, Ramat-Gan, Israel

Abstract

A sequence $d = (d_1, d_2, \ldots, d_n)$ of positive integers is *graphic* if it is the degree sequence of some simple graph G, and *planaric* if it is the degree sequence of some simple planar graph G. It is known that if $\sum d \leq 2n - 2$, then d has a realization by a forest, hence it is trivially planaric. In this paper, we seek bounds on $\sum d$ that guarantee that if d is graphic then it is also planaric. We show that this holds true when $\sum d \leq 4n - 4 - 2\omega_1$, where ω_1 is the number of 1's in d. Conversely, we show that there are graphic sequences with $\sum d = 4n - 2\omega_1$ that are non-planaric. For the case $\omega_1 = 0$, we show that d is planaric when $\sum d \leq 4n - 4$. Conversely, we show that there is a graphic sequence with $\sum d = 4n - 2$ that is non-planaric. In fact, when $\sum d \leq 4n - 6 - 2\omega_1$, d can be realized by a graph with a 2-page book embedding.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Degree Sequences, Graph Algorithms, Graph Realization, Planar Graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.18

Funding Supported by a US-Israel BSF grant (2022205).

1 Introduction

Background. In a graph G with n vertices, the degree of a vertex is the number of edges incident to it. Let $\deg(G)$ denote the sequence of length n of vertex degrees of G. The Degree Realization problem concerns deciding, given a sequence d of n positive integers, whether d has a *realizing graph*, namely, a graph G with n vertices such that $\deg(G) = d$, and finding such a graph if exists. A *graphic* sequence is one admitting a realizing graph. A full characterization of graphic degree sequences was given by Erdös and Gallai [7]. Havel and Hakimi [11, 12] described an algorithm that, given a sequence d, generates a realization, or verifies that d is not graphic.

The realizability characterization of [7] for general graphs takes into account all the elements of the sequence d. In contrast, the realizability of degree sequences by some special graph classes can be characterized more economically. An extreme example is realizability by a *forest* (cycle-free graph). Here, a single parameter suffices, namely, the *volume* $\sum d = \sum_{i=1}^{n} d_i$ of d. Concretely, if $\sum d \leq 2n - 2$, then d can be realized by a forest, and if $\sum d \geq 2n$ then it cannot [10]. A slightly more involved and less economical characterization applies for realizations by *cacti* graphs. A cactus graph is a connected graph in which every edge occurs on at most one cycle, namely, different cycles do not share edges (but may share one vertex).

The paper [16] gives a full characterization for sequences that can be realized by a cactus graph based only on the volume of the sequence, the number of 2's in the sequence, and the number of odd degrees in the sequence.

In this paper, we are interested in realizability by planar graphs, which turns out to be a challenging task that is still an open problem after more than half a century. A sequence $d = (d_1, d_2, \ldots, d_n)$ of positive integers is *planaric* if it is the degree sequence of some planar graph G. Planaric sequences were studied in, e.g., [13, 1, 18, 8, 9, 17] and more. At the moment, however, a complete characterization for planaric sequences is not yet available.

Contributions. Let d be a sequence with n positive integers and let ω_i be the *multiplicity* of i in d. This paper investigates the impact of the volume $\sum d$ and the multiplicity parameters ω_1 and ω_2 of the sequence d on its realizability by a planar graph. One direction follows from [18] which implies that if $\sum d > 6n - 12 - 2\omega_2 - 4\omega_1$ then d is not planaric. (This is tight in the sense that there are planaric sequences with $\sum d \leq 6n - 12 - 2\omega_2 - 4\omega_1$.) In this paper, we focus on the converse direction, i.e., we seek bounds on these parameters that guarantee that if d is graphic then it is also planaric. A simple bound is obtained by recalling the above-mentioned known fact that if $\sum d \leq 2n - 2$, then d has a a realization by a forest with $(2n - \sum d)/2$ components, hence it is planaric [10]. Here, we give stronger bounds for this problem, depending on $\sum d$ and ω_1. It turns out that most of the technical effort involves handling the *leaf-free* case (the case in which d does not contain 1s). We establish the following.

▶ **Theorem 1.** *Every graphic sequence d with $\omega_1 = 0$ and $\sum d \leq 4n - 4$ is planaric.*

This in turn enables us to prove our more general main result.

▶ **Theorem 2.** *Every graphic sequence d with $\sum d \leq 4n - 4 - 2\omega_1$ is planaric.*

In fact, when $\sum d \leq 4n - 6 - 2\omega_1$, our constructed realizing graphs are not only planar but also enjoy a 2-page book embedding, yielding the following corollary.

▶ **Corollary 3.** *Every graphic sequence d with $\sum d \leq 4n - 6 - 2\omega_1$ can be realized by a graph with a 2-page book embedding.*

This corollary can be interpreted as saying that the family $\mathcal{D}$ of all graphic sequences d such that $\sum d \leq 4n - 6 - 2\omega_1$ enjoy a 2-page book embedding realization. In comparison, the main result of [2] gives a partition of $\mathcal{D}$ into *non-outerplanaric* sequences and sequences enjoying a 2-book embedding realization. Moreover, by [2] if d is outerplanaric and $\sum d \geq 2n$, then $\sum d \leq 4n - 6 - 2\omega_1$. Therefore, this corollary can be seen as an alternative way to obtain the main result of [2].

Conversely, we show that there are graphic sequences with $\sum d \leq 4n - 2\omega_1$ that are non-planaric. For the case of $\omega_1 = 0$, there is a known graphic but non-planaric sequence with $\sum d = 4n - 2$. The gap between the bounds for $\omega_1 > 0$ is left for future study.

Note that the parameters $\sum d$, ω_1 and ω_2 are insufficient for charting the borderline between planaric and non-planaric sequences, and leave a "grey area" in between our upper and lower bounds, in which some sequences are planaric and some are not. This hints that a full characterization may require using additional parameters, and perhaps involve all the degrees, as is the case with realizability by general graphs.

Related work. Planaric sequences for regular planar graphs were classified in [13], and planaric bipartite biregular degree sequences were studied in [1]. In [18], Schmeichel and Hakimi determined which graphic sequences with $d_1 - d_n = 1$ are planaric, and presented similar results for $d_1 - d_n = 2$ with a small number of unsolved cases. Some of the sequences left unsolved in [18] were later resolved in [8, 9]. Some additional studies on special cases of the planaric degree realization problem are discussed in Rao's survey [17].

An economical characterization is given in [5] for the class of 2-trees, which is a subfamily of planar graphs, that is based on $\sum d$, ω_2, and ω_{odd}. For the Outerplanar Degree Realization problem, a full characterization of *forcibly* outerplanar graphic sequences (namely, sequences each of whose realizations is outerplanar) was given in [6]. A characterization of the degree sequence of maximal outerplanar graphs having exactly two 2-degree nodes was provided in [5]. A characterization of the degree sequences of maximal outerplanar graphs with at most four vertices of degree 2 was given in [15]. In [3] it is shown that a nonincreasing n-element graphic sequence d is outer-planaric if either $\omega_1 = 0$ and $\sum d \leq 3n - 3$, or $\omega_1 > 0$ and $\sum d \leq 3n - \omega_1 - 2$. Conversely, there are graphic sequences that are not outer-planaric with $\omega_1 = 0$ and $\sum d = 3n - 2$, as well as ones with $\omega_1 > 0$ and $\sum d = 3n - \omega_1 - 1$

2 Preliminaries

Given a sequence $d = (d_1, \dots, d_n)$ of n integers, we assume that it is non-increasing, namely that $d_{i+1} \leq d_i$, for every $i \in \{1, \dots, n-1\}$. Given two sequences d and d', denote by $d \ominus d' = (d_1 - d'_1, \dots, d_n - d'_n)$ their componentwise difference. For a nonincreasing sequence d of n nonnegative integers, let $\mathsf{pos}(d)$ denote the prefix consisting of the positive integers of d. We use the shorthand a^k to denote a subsequence of k consecutive a's. For any graph G, let $E(G)$ be the edge set of G.

Euler's theorem implies that if d is planaric, where $n \geq 3$, then $\sum d \leq 6n - 12$. Call d a *maximal* Euler sequence if $\sum d = 6n - 12$.

Known planaric sequences. A sequence d is called a *k-sequence* if $d_1 - d_n = k$. Schmeichel and Hakimi [18] divided the analysis for 2-sequences into maximal and non-maximal 2-sequences. They left a few open cases, some of which were resolved by Fanelli [8, 9].

▶ **Lemma 4** ([9, 18]). *Every graphic non-maximal Euler 2-sequence is planaric except for* $(4^5, 2)$, $(5^5, 3^3)$, $(5^{11}, 3)$, $(5^{13}, 3)$, $(6^{n-7}, 4^7)$ *for* $n > 7$, $(7, 5^{15})$, $(7, 5^{17})$, *and possibly* $(7^3, 5^{17})$, *whose status is unresolved.*

The following lemma describess another (relatively small[1]) class of degree sequences known to be planaric.

▶ **Lemma 5** ([18],Theorem 5(a)). *For $n \geq 3$, if d such that $d_1 \geq d_2 \geq \cdots \geq d_n$ is graphic, $\sum d \leq 6n - 12$ and $d_3 \leq 3$, then d is planaric.*

▶ **Observation 6.** *If G is a planar graph, then adding a parallel edge to $E(G)$ maintains the planarity of the resulting graph.*

Minimum pivot Havel-Hakimi algorithm. The *minimum pivot* version of the Havel-Hakimi algorithm [12, 11] for realizing a degree sequence $d = (d_1, \dots, d_n)$ associated with the vertices $v_1, \dots, v_n$, presented explicitly in [19] is based on repeatedly performing the following operation, hereafter referred to as the *MP-step*, until all the vertices reach their required degrees. Suppose that the current sequence of residual degrees is $\delta = (\delta_1, \cdots, \delta_h)$.

[1] The condition $d_3 \leq 3$ implies that the number of sequences in the class is upper bounded by n^4.

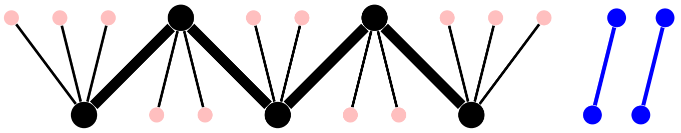


Figure 1 Realization of the forestic sequence $(4^5, 1^{16})$ by an alternating caterpillar and a matching. The spine is depicted by bold black vertices and edges.

The MP-step.

- Pick as a pivot one of the vertices with the minimum non-zero residual degree v_i whose degree is δ_i (break ties arbitrarily).
- Set v_i's neighbors to be the δ_i vertices with the highest residual degrees $v_{i_1}, v_{i_2}, \ldots, v_{i_{\delta_i}}$ (break ties arbitrarily).
- Set $\delta_i \leftarrow 0$ and reduce by 1 the residual degrees of its selected neighbors. That is, set $\delta_{i_j} \leftarrow \delta_{i_j} - 1$ for $j \in \{1, \ldots, \delta_i\}$.

The Minimum pivot Havel-Hakimi algorithm terminates when all the n residual degrees are zero, that is, when $\delta_j = 0$ for $j \in \{1, \ldots, n\}$. The key observation is that, whenever the MP-step transforms the residual degree sequence δ into δ', the following holds: δ is graphic if and only if δ' is graphic.

Caterpillar-based realizations. It is known that if $\sum d \leq 2n - 2$, then d can be realized by a cycle-free graph (forest). In this case, d is called a *forestic* sequence. If $\sum d = 2n - 2$, then d can be realized by a tree and the sequence is called a *treeic* sequence. The following lemma concerns the realization of forestic sequences. We make use of a special type of realizations of forestic and treeic sequences by *caterpillar trees.* In a caterpillar tree $G = (V, E)$, all the non-leaves vertices are arranged on a path, called the *spine.*

▶ **Lemma 7.** *A forestic sequence $d = (d_1, d_2, \ldots, d_n)$ of positive integers can be realized by a union of a caterpillar tree and a matching. Moreover, the order of the vertices on the spine may be chosen arbitrarily.*

For the sake of our later constructions, let us outline the way the realization of Lemma 7 is obtained. Run the *minimum pivot* version of the Havel-Hakimi algorithm while applying the MP-step until all the degrees in the residual sequence are at most 2. Then realize the residual sequence with a path (of arbitrary order) and a matching. The interior of the path is the spine of the caterpillar while the pivots and the two end vertices of the path are the leaves of the caterpillar. Our later constructions make critical use of the "arbitrary ordering" property. It is convenient to illustrate a caterpillar tree with its spine drawn horizontally (in a zigzagged fashion), and its groups of leaves drawn alternately above and below the spine. We refer to this representation as an *alternating caterpillar.* (See Figure 1.)

Outer-planar graphs. An *outer-planar* graph is a graph that has a planar embedding in which all the vertices occur on the outer face. A *maximal outer-planar* (MOP) graph is an outer-planar graph such that adding any new edge to it results in a non-outer-planar graph. Given a planar embedding in which all the vertices occur on the outer face, an *external* edge is an edge residing on the outer face. If $d = (d_1, \ldots, d_n)$ is an outer-planaric degree sequence where $n \geq 2$, then $\sum d \leq 4n - 6$, with equality if and only if d is maximal outer-planaric [20].

A (directed) *circuit* in G is an *ordered set* of vertices $C = \{v_0, v_1, \ldots, v_{k-1}\}$ such that $v_i \neq v_{(i+1) \bmod k}$ and $(v_i, v_{(i+1) \bmod k}) \in E$ for every $i = 0, \ldots, k-1$. In a directed circuit, vertices may appear more than once while each edge may appear at most once. Note that

$e = (u, w)$ is an external edge if u and w are neighbors on a circuit which is part of the outer face. All other edges of G are *internal*. An *internal triangle* in G is a triangle all of whose edges are internal. Jao and West [14] show that the number of internal triangles in a maximal outer-planar graph (MOP) is related to ω_2.

▶ **Lemma 8** ([14]). *Let G be a MOP on n vertices, let $d = \deg(G)$, and let t be the number of internal triangles. If $n \geq 4$, then $t = \omega_2 - 2$.*

Given a graph $G = (V, E)$, let $E = E_1 \cup \ldots \cup E_p$ be a partition of its edges such that each subgraph $G_i = (V, E_i)$ is outerplanar. For a book embedding of G, think of a book in which the pages (half-planes) are filled by outerplanar embeddings of the G_i's such that the vertices are embedded on the spine of the book and in the same location on each page. This constraint is equivalent to requiring that the vertices appear in the same order along the cyclic order of each of the outerplanar embeddings of the G_i's. The *book thickness* or *pagenumber* [4] is the minimal number of pages for which a graph has a valid book embedding. Note that a graph is outerplanar if and only if it has pagenumber 1, and it is known that the pagenumber of planar graphs is at most 4 [21].

▶ **Lemma 9** ([4]). *A graph G has a 2-book embedding if and only if G is a subgraph of a Hamiltonian planar graph.*

3 Tools and sufficient conditions for OP and MOP realizations

3.1 Leaf-free sequences with $\sum d \leq 4n - 6$ and $\omega_2 = 2$ are outer-planaric

We start with a special class of sequences for which we present a basic construction of an *outerplanar* realization. A number of our later constructions of planar realizations start from this basic construction (typically applied to a sub-sequence) and modify it in various ways in order to derive the required planar realization. In many of these cases, the modification requires adding a few more edges. For each additional edge, this requires finding a pair of vertices u, w such that

(i) the edge (u, w) does not appear in the construction, and

(ii) adding it preserves planarity.

We make use of the following lemma, established in [2].

▶ **Lemma 10** ([2]). *Let d be a graphic sequence such that $d_1 \geq d_2 \geq \cdots \geq d_n$ with $\omega_1 = 0$, $\omega_2 = 2$, and $\sum d \leq 4n - 6$. Then d is outer-planaric.*

We outline the construction of the realizing graph, since it will be instrumental in what follows. Let $d' = \mathsf{pos}(d \ominus (2^n))$. Then $n' = n - 2$ and $\sum d' \leq 2n' - 2$ because

$$\sum d' = \sum d - 2n \leq 2n - 6 = 2(n' + 2) - 6 = 2n' - 2 \ .$$

By Lemma 7, d' can be realized by a graph $G' = (V', E')$ composed of a union of an alternating caterpillar T' and a matching M'. Next, construct an outer-planar realization G for d based on G' as follows.

Let $S = (x_1, \ldots, x_s)$ be the vertices on the spine of T', and let $X_i = \{\ell_{i,1}, \ldots, \ell_{i,k_i}\} \subseteq V'$ be the leaves adjacent to the spine vertex x_i, for $i \in \{1, \ldots, s\}$. Note that in T', $deg(x_i) = k_i + 1$ for $i \in \{1, s\}$ and $deg(x_i) = k_i + 2$ otherwise. Assume that the matching is $M' = \{(y_1, z_1), (y_2, z_2), \ldots, (y_t, z_t)\}$. To construct an outer-planar realization of d, add to

G' a set of edges that form a Hamiltonian cycle (including two additional new vertices of degree 2). The construction consists of two steps. We describe it for $s = 0$ and for an odd s; an analogous construction applies for a positive even s.

(1) Construct two paths
$P_1 = (x_1, \ell_{2,1}, \dots, \ell_{2,k_2}, x_3, \ell_{4,1}, \dots, \ell_{4,k_4}, \dots, x_{s-2}, \ell_{s-1,1}, \dots, \ell_{s-1,k_{s-1}}, x_s, y_1, y_2, \dots, y_t)$,
$P_2 = (z_t, \dots, z_2, z_1, \ell_{s,k_s}, \dots, \ell_{s,1}, x_{s-1}, \dots, \ell_{3,k_3}, \dots, \ell_{3,1}, x_2, \ell_{1,k_1}, \dots, \ell_{1,1})$,
connecting the spine vertices in odd and even positions, respectively. If $s = 0$, then $P_1 = (y_1, \dots, y_t)$ and $P_2 = (z_t, \dots, z_1)$.

(2) Add two new vertices x_0 and x_{s+1}. If $s > 0$ and $t > 0$, then connect x_0 with x_1 and $\ell_{1,1}$ and connect x_{s+1} with y_t and z_t. If $s = 0$ and $t > 0$, then connect x_0 with y_1 and z_1 and connect x_{s+1} with y_t and z_t. If $t = 0$ and $s > 0$, then connect x_0 with x_1 and $\ell_{1,1}$ and connect x_{s+1} with x_s and ℓ_{s,k_s}. These two vertices form a cycle C together with P_1 and P_2. The new edges added to G' to construct G are the edges of the cycle C ($E(C)$) which are the edges from the two paths P_1 ($E(P_1)$) and P_2 ($E(P_2)$) and the four edges that connect x_0 and x_{s+1} to these paths.

Observe that after adding the cycle to G', the degree of each one of the n' vertices is increased by 2. Together, with the two new vertices of degree 2 the modified graph G is an outer-planar realization of the original sequence d. For an illustration of the resulting outer-planar graph G for $s = 5$ and $t = 2$, see Figure 2.

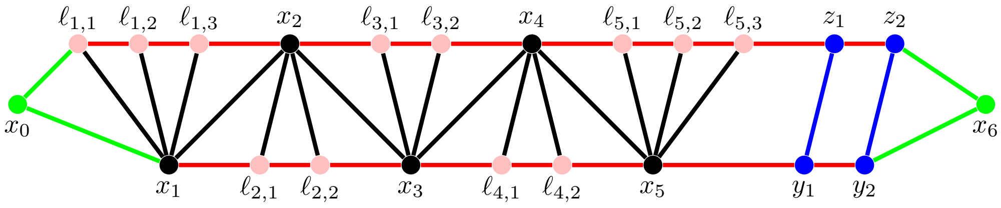


Figure 2 Illustration for the outer-planar construction described in Lemma 10.

3.2 Leaf-free sequences with $\sum d = 4n - 6$ and $\omega_2 = 3$ are outer-planaric

The following lemma is used in our analysis for sequences in which $\sum d = 4n - 4$, $w_1 = 0$ and $w_2 = 3$.

▶ **Lemma 11.** *Let d, such that $d_1 \geq d_2 \geq \dots \geq d_n$, be a degree sequence such that*
(i) $\sum d = 4n - 6$,
(ii) $d_1 \geq 5$,
(iii) $d_3 \geq 4$,
(iv) $d_n = 2$, *and*
(v) $\omega_2 = 3$.
Then
(a) *d can be realized by a maximal outer-planaric graph G,*
(b) *A vertex $v \in V$ cannot be adjacent to three vertices of degree 2 in G.*

Proof. Let d be as in the lemma. By (iii) and (v), $n \geq 6$. Moreover, if $n = 6$ then $d_4 = 2$ by (iv) and (v), and combining it with (i) we get $\sum d = 18 = d_1 + d_2 + d_3 + 6$, so $d_1 + d_2 + d_3 = 12$, which contradicts (ii) and (iii). Therefore, $n \geq 7$. Also note that if $d_4 = 2$ then $n \leq 6$ by (iv)

and (v), leading to the same contradiction. Hence, $d_4 \geq 3$. Due to Lemma 8, if d has a MOP realization G, then G has exactly one internal triangle because $\omega_2 - 2 = 1$. To prove the lemma, we construct a MOP realization of d in which the internal triangle is formed by the vertices whose degrees are d_1, d_2, and d_3. We divide the sequences satisfying the conditions of the lemma into three (possibly overlapping) families, named $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, a nd show the realizations of sequences that belong to each class separately.

Family $\mathcal{A}$. This family contains all the sequences that satisfy the requirements of the lemma with the additional requirement that $d_1 + d_2 < 10$. Since $d_1 \geq 5$ and $d_3 \geq 4$, it follows that in these sequences $d_1 = 5$ and $d_2 = d_3 = 4$. Therefore, $\mathcal{A}$ contains all the sequences of the type $(5, 4^{\omega_4}, 3^{\omega_3}, 2^3)$ for $\omega_4 \geq 2$. To satisfy the $\sum d = 4n - 6$ requirement of the lemma, it must be the case that $\omega_3 = 1$ as shown below.

$$\begin{aligned} \sum d &= 5 + 4\omega_4 + 3\omega_3 + 6 = 4\omega_4 + 3\omega_3 + 11 \\ 4n - 6 &= 4(\omega_4 + \omega_3 + 4) - 6 = 4\omega_4 + 4\omega_3 + 10 \end{aligned}$$

This implies that $\omega_3 = 1$. To summarize, $\mathcal{A}$ contains all the sequences of the type

$$a(\omega_4) = (5, 4^{\omega_4}, 3, 2^3)$$

of length $n = \omega_4 + 5$ for $\omega_4 \geq 2$.

The following describes how to construct a MOP graph with one inner triangle, denoted by $G(\omega_4)$, realizing the sequence $a(\omega_4)$ for $\omega_4 \geq 3$. Let the $n = \omega_4 + 5$ vertices of $G(\omega_4)$ be

$$(u_1, u_2, u_3, t_0, t_1, \ldots, t_{\omega_4-3}, t_{\omega_4-2}, t_{\omega_4-1}, r_1, r_2)$$

and associate them respectively with the degrees $(5, 4, 4, \overbrace{4, \ldots, 4}^{\omega_4-2}, 3, 2, 2, 2)$. Note that the ω_4 vertices of degree 4 are $u_2, u_3, t_0, t_1, \ldots, t_{\omega_4-3}$ while the degrees of t_{ω_4-2} and t_{ω_4-1} are 3 and 2 respectively. Let the $2\omega_4 + 7$ $(= \sum d/2)$ edges of $G(\omega_4)$ be the three edges of the triangle (u_1, u_2, u_3), the six edges (u_1, t_0), (u_1, r_1), (u_2, r_1), (u_2, r_2), (u_3, r_2), and (u_3, t_0), the $\omega_4 - 1$ edges forming the path $(t_0, t_1, \ldots, t_{\omega_4-2}, t_{\omega_4-1})$, and the $\omega_4 - 1$ edges forming the two paths $(t_0, t_2, t_4 \ldots)$ and $(u_1, t_1, t_3, \ldots)$ of length $\lfloor(\omega_4 - 1)/2\rfloor$ and $\lceil(\omega_4 - 1)/2\rceil$ respectively. For an odd ω_4 the first path is $(t_0, t_2, t_4 \ldots, t_{\omega_4-3}, t_{\omega_4-1})$ and the second path is $(u_1, t_1, t_3, \ldots, t_{\omega_4-4}, t_{\omega_4-2})$ while for an even ω_4 the first path is $(t_0, t_2, t_4 \ldots, t_{\omega_4-4}, t_{\omega_4-2})$ and the second path is $(u_1, t_1, t_3, \ldots, t_{\omega_4-3}, t_{\omega_4-1})$.

Figure 3 illustrates the outer-planar layout of $G(5)$. Observe that for an odd ω_4 the outer Hamiltonian cycle of $G(\omega_4)$ that contains all of its vertices is

$$(u_1, r_1, u_2, r_2, u_3, t_0, t_2, \ldots, t_{\omega_4-3}, t_{\omega_4-1}, t_{\omega_4-2}, t_{\omega_4-4}, \ldots, t_3, t_1, u_1)$$

and for an even ω_4 it is

$$(u_1, r_1, u_2, r_2, u_3, t_0, t_2, \ldots, t_{\omega_4-4}, t_{\omega_4-2}, t_{\omega_4-1}, t_{\omega_4-3}, \ldots, t_3, t_1, u_1)$$

In both cases, (u_1, u_2, u_3) is the only inner triangle.

Family $\mathcal{B}$. This family contains all the sequences that satisfy the requirements of the lemma with an additional requirement that $d_4 = 3$. Hence, $d_1 = 4 + j$ for $j \geq 1$ because $d_1 \geq 5$, $d_2 = 4 + i$ and $d_3 = 4 + h$ for $i, h \geq 0$ because $d_2 \geq d_3 \geq 4$, and $j \geq i \geq h$ because d is a non-increasing sequence. Therefore, $\mathcal{B}$ contains all the sequences of the type

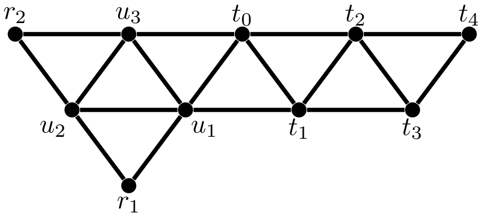


Figure 3 The MOP realization $G(5)$ of the sequence $(5, 4^5, 3, 2^3)$.

$b(j,i,h) = ((4+j),(4+i),(4+h),3^{\omega_3},2^3)$ for $j \geq i \geq h \geq 0$ and $j \geq 1$. To satisfy the $\sum d = 4n-6$ requirement of the lemma, it must be the case that $\omega_3 = (j+i+h)$ as shown below.

$$\begin{aligned}\sum d &= (4+j)+(4+i)+(4+h)+3\omega_3+6 = (j+i+h)+3\omega_3+18\\ 4n-6 &= 4(\omega_3+6)-6 = 4\omega_3+18\end{aligned}$$

This implies $\omega_3 = (j+i+h)$. To summarize, $\mathcal{B}$ contains all the sequences of the type

$$b(j,i,h) = ((4+j),(4+i),(4+h),3^{j+i+h},2^3)$$

of length $n = j+i+h+6$ for $j \geq i \geq h \geq 0$ and $j \geq 1$.

The following describes how to construct a MOP graph with one inner triangle, denoted by $G(j,i,h)$, realizing the sequence $b(j,i,h)$ for $j \geq i \geq h \geq 0$ and $j \geq 1$. Let the $n = j+i+h+6$ vertices of $G(j,i,h)$ be

$$(u_1, u_2, u_3, p_0, \ldots, p_{j-1}, q_0, \ldots, q_{i-1}, r_0, \ldots, r_{h-1}, p_j, q_i, r_h)$$

and associate them respectively with the degrees

$$((4+j),(4+i),(4+h),\overbrace{3,\ldots,3}^{j},\overbrace{3,\ldots,3}^{i},\overbrace{3,\ldots,3}^{h},2,2,2)$$

Let the $2(j+i+h)+9$ $(=\sum d/2)$ edges of $G(j,i,h)$ be the three edges of the triangle (u_1,u_2,u_3), the three edges (u_1,q_0), (u_2,r_0), and (u_3,p_0), the $j+1$ edges (u_1,p_ℓ) for $0 \leq \ell \leq j$, the $i+1$ edges (u_2,q_ℓ) for $0 \leq \ell \leq i$, the $h+1$ edges (u_3,r_ℓ) for $0 \leq \ell \leq h$, the j edges forming the path $(p_0,p_1,\ldots,p_j)$, the i edges forming the path $(q_0,q_1,\ldots,q_i)$, and the h edges forming the path $(r_0,r_1,\ldots,r_h)$,

Figure 4 illustrates the outer-planar layout of $G(4,3,2)$. Observe that the outer Hamiltonian cycle of $G(j,i,h)$ that contains all of its vertices is

$$(u_1, p_j, p_{j-1}, \ldots, p_0, u_3, r_h, r_{h-1}, \ldots, r_0, u_2, q_i, q_{i-1}, \cdots, q_0, u_1)$$

and that (u_1,u_2,u_3) is the only inner triangle.

Family $\mathcal{C}$. This family contains all the sequences that satisfy the requirements of the lemma but do not belong to $\mathcal{A} \cup \mathcal{B}$. That is, the requirement *(iii)* is modified and a new requirement *(vi)* is added as follows,

(iii) $d_4 \geq 4$

(iv) $d_1 + d_2 \geq 10$

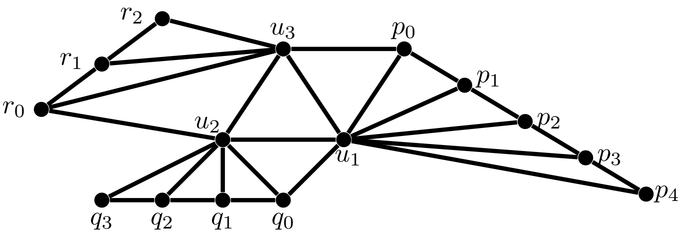


Figure 4 The MOP realization $G(4,3,2)$ of the sequence $(8,7,6,3^9,2^3)$.

The outline of the construction of the MOP realization of d that satisfies the new set of six requirements is as follows. The construction is done in two phases. In the first phase, d is modified to a shorter sequence d' of length $n' = n-3$ by contracting the three largest degrees in d into one degree and eliminating one appearance of 2 in the sequence d. We will show that the new sequence d' satisfies all the requirements of Lemma 10. As a result, this lemma will provide an outer-planar realization G' of d'. In the second phase, the construction of the realization G of the sequence d of length n will be completed by replacing the highest degree vertex in G' with a triangle of vertices and adding another vertex of degree 2 while making sure that all the $n = n' + 3$ vertices in G has the required degrees from the sequence d.

Formally, generate the sequence d' such that $d'_1 \geq d'_2 \geq \cdots \geq d'_n$ of length $n' = n-3$ by removing d_n and merging d_1, d_2 and d_3 as follows.

$$d'_i = \begin{cases} d_1 + d_2 + d_3 - 10, & \text{for } i = 1, \\ d_{i+2}, & \text{for } i = 2, \ldots, n-3. \end{cases}$$

Note that since $d_1 + d_2 \geq 10$, it follows that $d_1 + d_2 + d_3 - 10 \geq d_3 \geq d_4 = d'_2$ and therefore since d is non-increasing it follows that d' is also non-increasing.

Note that $\sum d = 4n - 6$ by the assumptions of the lemma and that $\omega'_2 = 2$ and $d'_{n'} = 2$ by the definition of d'. Consequently,

$$\sum d' = \sum d - 10 - 2 = 4n - 18 = 4(n'+3) - 18 = 4n' - 6 \ .$$

By Lemma 10, d' is outer-planaric.

By the proof of Lemma 10, $d'' = \mathsf{pos}(d' \ominus (2^{n'}))$ can be realized by a caterpillar T and there exists an outer-planar realization G' of d' that is based on a caterpillar T with vertices $S = (x_1, \ldots, x_s)$ forming its spine and a cycle C composed of two paths P_1 and P_2 and two new vertices x_0 and x_{s+1}. (An illustration of this outer-planar graph can be found in Figure 5.) Recall that in T, $X_i = \{\ell_{i,1}, \ldots, \ell_{i,k_i}\} \subseteq V'$ is the set of leaves adjacent to the spine vertex x_i, for $i \in \{1, \ldots, s\}$. This realization G' does not have an internal triangle. We continue referring to these vertices as the spine vertices and leaves although G' is not a caterpillar.

Transforming G' into a realization G of d involves two steps. In the first step, x_1 whose degree is d'_1 will be replaced by three vertices u_1, u_2, and u_3. Note that by Lemma 7, the order of the vertices on the spine can be chosen arbitrarily. As a result, it can be assumed that the degree of x_1 is d'_1. In the second step, the outer cycle of G' will be modified to cover the new vertices with the addition of a new vertex of degree 2.

By the construction of G' from the caterpillar T and since $d'_2 = d_4 \geq 4$ by the new requirement *(iv)*, it follows that the spine of T has at lease two vertices x_1 and x_2 and x_1 is connected in G' to the two vertices x_0 (one of the two vertices of degree 2 in G') and x_2. In

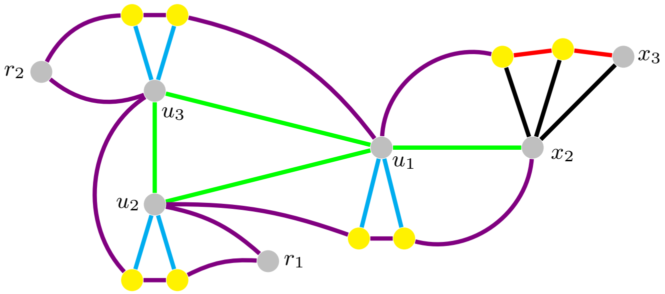


Figure 5 Realization of the sequence $d = (7, 6^2, 5, 3^8, 2^3)$. (Red and black edges are part of the original construction.)

addition, x_1 is connected in G' to the k_1 leaves $\{\ell_{1,1}, \ldots, \ell_{1,k_1}\}$ from the set X_1, as well as to $\ell_{2,1}$ (or to x_3 in case x_2 does not have leaves). Therefore,

$$k_1 = d'_1 - 3 = d_1 + d_2 + d_3 - 13 \; . \tag{1}$$

The expansion of x_1 into a triangle of vertices is done as follows. Remove x_0, x_1, and the leaves of X_1 from G'. Add the three vertices u_1, u_2, and u_3, add a triangle of edges connecting them, namely, the edges (u_1, u_2), (u_1, u_3) and (u_2, u_3), and add the edge (u_1, x_2). (See the green edges in Figure 5.)

Next, split the k_1 leaves in X_1 between the vertices u_1, u_2, and u_3 toward satisfying their degrees d_1, d_2, and d_3, respectively. Specifically, add a set U_1 of $d_1 - 5$ new leaves adjacent to u_1, and a set U_i of $d_i - 4$ new leaves adjacent to u_i, for $i = 2, 3$. (See the blue edges in Figure 5.) The split is perfect since $\sum_{i=1}^{3} |U_i| = k_1$ by Equation 1. In summary,

- u_1 is adjacent to u_2, u_3, x_2, and the $d_1 - 5$ vertices in U_1;
- u_2 is adjacent to u_1, u_3, and the $d_2 - 4$ vertices in U_2; and
- u_3 is adjacent to u_1, u_2, and the $d_3 - 4$ vertices in U_3 .

Therefore, at this point $\deg(u_i) = d_i - 2$, for $i = 1, 2, 3$.

Finally, add two more degree-2 vertices r_1 and r_2 to complete the outer cycle. One of them replaces x_0 while the other is an additional degree 2 vertex, as d_n was removed in the definition of d'. For $i = 1, 2, 3$, connect the leaves in U_i to form a path whose starting and ending vertices (of degree 1) are u_i^s and u_i^e respectively. Next, add the edges (x_2, u_1^s) and (u_1^e, u_2) if $|U_1| > 0$, otherwise add the edge (x_2, u_2). Analogously, add the edges (u_2, r_1), (r_1, u_2^s), (u_2^e, u_3) if $|U_2| > 0$, otherwise add edges (u_2, r_1) and (r_1, u_3). Add edges (u_3, r_2), (r_2, u_3^s), and (u_3^e, u_1) if $|U_3| > 0$, otherwise add edges (u_3, r_2) and (r_2, u_1). Finally, add the edge $(u_1, \ell_{2,1})$, or (u_1, x_3) in case x_2 does not have leaves. (See the violet edges in Figure 5.) At this stage the degrees of u_1, u_2, u_3 are d_1, d_2, and d_3 respectively and the degrees of r_1 and r_2 are 2.

See Figure 5 for an illustration of the realization G of d. One can verify that the construction is a realization of d. ◀

3.3 Degree-Two Removal (Procedure Deg_2_Remove)

As it turns out, sequences with a small number of 2 degrees are easier to realize directly. Consequently, when dealing with a sequence d with many 2's, a convenient approach is to first transform it into a "similar" sequence d' with only a few 2's, construct a graph G' realizing d', and then transform G' into a graph G realizing the original d. We next present a procedure called *Degree-Two Removal (*Deg_2_Remove*)* that will be used to that end in some of our constructions. The input to this procedure is a graphic sequence for which $d_3 \geq 4$,

$\omega_2 \geq 3$ and $\omega_1 = 0$. The procedure applies repeatedly the MP-step of the Havel-Hakimi algorithm until in the residual sequence either $\omega_2 < 3$ or the second maximum degree is less than 4. As a result, throughout its execution it is always the case that the residual sequence is graphic. For any degree sequence $d = (d_1, d_2, \ldots, d_n)$, let $d(\ell)$ be the ℓ'th largest degree in d, for $1 \leq \ell \leq n$. (Note that possibly $d(\ell) = d(\ell+1)$ for some ℓ values.)

Procedure 1 Deg_2_Remove.

```
Set $\bar{d} \leftarrow d$
Set $\bar{\omega}_2 \leftarrow |\{i \mid d_i = 2\}|$
while $\bar{d}(2) \geq 4$ and $\bar{\omega}_2 \geq 3$ do
    Let $j$ be such that $\bar{d}_j = \bar{d}(1)$ and let $j'$ be such that $\bar{d}_{j'} = \bar{d}(2)$
    $\bar{d}_j \leftarrow \bar{d}_j - 1$
    $\bar{d}_{j'} \leftarrow \bar{d}_{j'} - 1$
    Change one degree-2 in $\bar{d}$ to 0
    set $\bar{\omega}_2 \leftarrow \bar{\omega}_2 - 1$
Set $\bar{d}^s \leftarrow$ sort($\bar{d}$)          /* The final $\bar{d}$ sorted in non-increasing order */
```

Assume that Procedure Deg_2_Remove executes k iterations of its while-loop. Observe that during the run of the Deg_2_Remove procedure, no new degree 2 vertices appear because $\bar{d}(2) \geq 4$ is one of the conditions of the while-loop. As a result, $\bar{\omega}_2$, which is initially the number of degree-2 vertices in d, decreases by 1 after each iteration. Let $\bar{\omega}_2^i$ be the value of $\bar{\omega}_2$ after the i'th iteration, for $0 \leq i \leq k$. For $0 \leq i \leq k$, let $\bar{d}^i$ be the sequence $\bar{d}$ after the i'th iteration (note that $\bar{d}^s$ is the sorted version of $\bar{d}^k$), and let $\bar{n}^i$ be the length of $\mathsf{pos}(\bar{d}^i)$. We make use of the set A indices of high degrees that were reduced by Procedure Deg_2_Remove and the set B of indices of 2 degrees that were eliminated by the procedure. Formally,

$$A = \{i \mid d_i > \bar{d}_i,\ d_i \geq 4\}, \qquad B = \{i \mid d_i = 2,\ \bar{d}_i = 0\}. \tag{2}$$

▶ **Observation 12.**
(i) $\bar{n}^{i+1} = \bar{n}^i - 1$,
(ii) $\bar{\omega}_2^{i+1} = \bar{\omega}_2^i - 1$,
(iii) $\sum \bar{d}^{i+1} = \sum \bar{d}^i - 4$,
(iv) $|B| = k$,
(v) $\sum_{i \in A}(d_i - \bar{d}_i^k) = 2k$,
(vi) $\sum \bar{d}^k - 4\bar{n}^k = \sum d - 4n$,
(vii) $\bar{n}^k = n - k$,
(viii) $\bar{d}^i$ *is graphic for* $1 \leq i \leq k$.

▶ **Observation 13.** *When Procedure* Deg_2_Remove *terminates, the following properties hold.*
(i) *Either* $\bar{d}^k(2) \geq 3$ *and* $\bar{\omega}_2 = 2$*, or* $\bar{d}^k(2) = 3$ *and* $\bar{\omega}_2 \geq 3$.
(ii) *If* $\bar{d}_2^s \geq 5$*, then* $\bar{d}_i^k \geq 4$ *for every* $i \in A$.

The following lemma (whose proof is omitted) demonstrates the usefulness of Procedure 1 in reducing the number of degree 2 vertices to generate an outer-planar sequence. Later, it will be shown how to add back the removed degree 2 vertices to get a planar realization for the original sequence.

▶ **Lemma 14.** *Applying Procedure* Deg_2_Remove *(Procedure 1) on a graphic sequence* d *with* $\sum d \leq 4n - 6$*,* $\omega_1 = 0$ *and* $\omega_2 \geq 3$*, the output sequence* $\bar{d}^k$ *satisfies that* $\mathsf{pos}(\bar{d}^k)$ *is outer-planaric.*

4 Main Result

This section presents our main result (Theorem 2). Specifically, Subsection 4.1 shows that this theorem is implied directly by Theorem 1. The following two subsections establish Theorem 1, where Subsection 4.2 handles the easy case of sequences with few 2 degrees ($\omega_2 \leq 2$) and Subsection 4.3 handles the more elaborate case of sequences with many 2 degrees ($\omega_2 \geq 3$). Finally, Subsection 4.4 provides examples showing that our bounds are almost tight.

4.1 The planarity of low-volume sequences with $\omega_1 > 0$

We first rely on Theorem 1 to prove our main result.

Proof of Theorem 2. If $\sum d \leq 2n-2$, then d can be realized by a forest, hence it is planaric [10]. For the case where $\sum d \geq 2n$, we prove the claim by induction on ω_1. In the base case, $\omega_1 = 0$, the claim follows by Theorem 1. Now assume that the claim holds for $\omega_1 \leq i$ and consider $\omega_1 = i+1$. Construct d' by setting $d'_1 = d_1 - 1$, $d'_n = 0$ and $d'_i = d_i$ for $i \in \{2, \ldots, n-1\}$. Since $\sum d \geq 2n$ and $\omega_1 > 0$, we have that $d_1 \geq 3$ and hence $d'_1 \geq 2$. Let $d'' = \mathsf{pos}(d')$ and denote the number of 1-degrees in d'' by ω''_1. Then $\omega''_1 = \omega_1 - 1$ and $n'' = n-1$, implying that $\sum d'' \leq 4n'' - 4 - 2\omega''_1$. Also note that when applying the MP-step of the Havel-Hakimi method on d here, using d_n as pivot, we get d'', and hence d'' is graphic. Therefore, d'' satisfies the conditions of induction hypothesis and hence can be realized by a planar graph G''. To complete the construction, add one leaf to the vertex with degree d'_1 in G''. This yields a planar graph G realizing d. The theorem follows. ◀

The following two subsections are dedicated to the leaf-free case ($\omega_1 = 0$), and prove Theorem 1.

4.2 The planarity of leaf-free sequences with $\sum d \leq 4n-4$ and $\omega_2 \leq 2$

The case of "few degrees 2" is relatively easier, and is covered by the following lemma.

▶ **Lemma 15.** *Every graphic sequence d such that $d_1 \geq d_2 \geq \cdots \geq d_n$ with $\omega_1 = 0$, $\omega_2 \leq 2$, and $\sum d \leq 4n-4$ is planaric.*

Proof. If $d_3 < 4$, then d is planaric by Lemma 5. From now on assume in addition that $d_3 \geq 4$. We consider three cases depending on ω_2.

Case 1: $\omega_2 = 0$.
In this case, $d_n = 3$ since $d_n \geq 4$ would imply $\sum d \geq 4n$. Let $d' = d \ominus (2^n)$. Then $n' = n$ and $\sum d' = \sum d - 2n \leq 2n' - 4$. Therefore, d' is a forestic sequence. By Lemma 7, d' can be realized by a graph $G' = (V', E')$ composed of a union of an alternating caterpillar T' and a matching M'. This matching contains at least one edge because when $\sum d' \leq 2n' - 4$, any realization forest must contain at least two connected components.

Define the spine S, the leaf sets X_i in T' and the matching M' analogously to the proof of Lemma 10. Observe that since $d_3 \geq 4$ and the spine contains all the vertices whose degree in G' is at least 2, it follows that the spine S contains at least three vertices.

To construct a planar realization of d, add to G' a set of edges that form two disjoint cycles. The construction consists of two steps. We describe it for odd s; an analogous construction applies for even s.

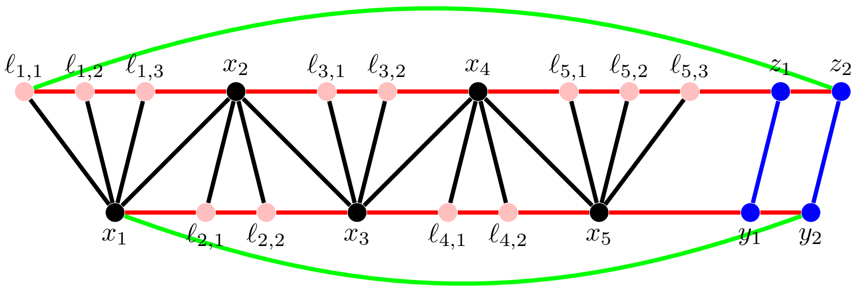


Figure 6 Illustration for a planar construction when $\omega_2 = 0$. This is essentially the outer-planar graph of Figure 2 after omitting the vertices x_0 and x_6 with the addition of the green edges (x_1, y_2) and $(z_2, \ell_{1,1})$.

(1) Construct two paths P_1 and P_2 as in the proof of Lemma 10.

(2) Connect x_1 with y_t and $\ell_{1,1}$ with z_t, thereby transforming the paths P_1 and P_2 into a cycle.

For an illustration of the resulting outer-planar graph G for $s = 5$ and $t = 2$, see Figure 6.

Case 2: $\omega_2 = 1$.
In this case, $d_n = 2$ and $d_{n-1} = 3$, since again $d_{n-1} \geq 4$ would contradict the assumption $\sum d \leq 4n - 4$. If $d_1 \leq 4$, then $d = (4^{\omega_4}, 3^{\omega_3}, 2)$ with even $\omega_3 \geq 2$ is planaric by Lemma 4. Hereafter, we assume that $d_1 \geq 5$.

Define d' by setting $d'_1 = d_1 - 1$, $d'_{n-1} = 2$ (replacing $d_{n-1} = 3$) and $d'_i = d_i$ for all other $i \in \{2, \ldots, n-2, n\}$. Then $\sum d' \leq 4n' - 6$ and $n' = n$. Let $d'' = \mathsf{pos}(d' \ominus (2^n))$. We have $n'' = n' - 2$ and $\sum d'' = \sum d' - 2n \leq 2n' - 6 = 2(n'' + 2) - 6 = 2n'' - 6$. By Lemma 7, d'' can be realized by a graph $G'' = (V'', E'')$ composed of a union of an alternating caterpillar T'' and a matching M''. Notice that $d''_1 \geq 2$ since $d_1 \geq 5$. Therefore, the vertex with degree d''_1 occurs on the spine of T'' and can be identified as x_1 since by Lemma 7 the spine-vertices may appear in any order.

Define the spine S, the leaf sets X_i in T'' and the matching M'' analogously to the proof of Lemma 10. Since $d_3 \geq 4$, by the construction of d' and d'', there are at least two vertices on the spine S and $s \geq 2$. To construct a planar realization of d, we add a set of edges to G'' as done in the proof of Lemma 10 and one extra edge. The construction consists of two steps. Again, we describe it only for odd s.

(1) Construct two paths P_1, P_2 and $E(C)$ as in the proof of Lemma 10

(2) Connect x_{s+1} and x_1 by an edge.

An illustration of the above steps is presented in Figure 7. Note that steps (1) and (2) build an outer-planar graph $G' = (V, E')$ with $E(G') = E(G'') \cup E(C)$ realizing d'. Notice that $(x_{s+1}, x_1) \notin E(G')$ since $s \geq 2$. Therefore, step (3) yields a simple planar graph $G = (V, E)$ with $E(G) = E(G'') \cup E(C) \cup \{(x_{s+1}, x_1)\}$ realizing d.

Case 3: $\omega_2 = 2$ (i.e., $d_n = d_{n-1} = 2$ and $d_{n-2} \geq 3$).
In this case, for $\sum d \leq 4n - 6$, the result follows directly from Lemma 10.

Next consider $\sum d = 4n - 4$. This case is divided into three sub-cases.

Case 3.1: $d_1 = 4$.
In this case, d is a 2-sequence, i.e., it satisfies $d_1 - d_n = 2$, and such d is known to be planaric, see Lemma 4.

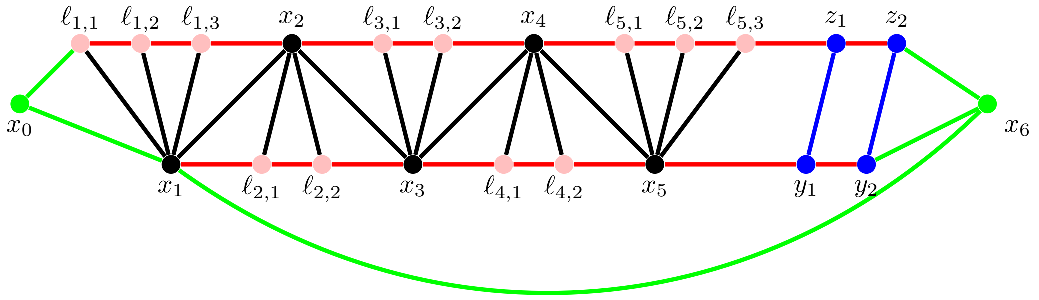


Figure 7 Illustration for planar construction when $\omega_2 = 1$. This is essentially the outer-planar graph of Figure 2 with the addition of the green edge (x_1, x_6).

Case 3.2: $d_1 = 5$ and $d_2 = 4$.
In this case, $\sum d = 4n - 4$ dictates that $d = (5, 4^{\omega_4}, 3, 2^2)$ for $\omega_4 \geq 2$ since $d_3 \geq 4$. The following describes how to construct a planar graph, denoted by $G(\omega_4)$, realizing the sequence d for $\omega_4 \geq 2$. Let the $n = \omega_4 + 4$ vertices of $G(\omega_4)$ be $(u, x_1, x_2, \ldots, x_{\omega_4}, r_1, r_2, r_3)$, and associate them respectively with the degrees $(5, \overbrace{4, \ldots, 4}^{\omega_4}, 3, 2, 2)$. Let the $2\omega_4 + 6$ $(= \sum d/2)$ edges of $G(\omega_4)$ be the five edges (u, r_1), (u, r_2), (u, r_3), (u, x_1), (u, x_2), the four edges $(r_1, x_{\omega_4 - 1})$, (r_1, x_{ω_4}), (r_2, x_{ω_4}), and (r_3, x_1), the $\omega_4 - 1$ edges forming the path $(x_1, x_2, \ldots, x_{\omega_4 - 1}, x_{\omega_4})$, and the $\omega_4 - 2$ edges forming the two paths $(x_1, x_3, \ldots)$ and $(x_2, x_4, \ldots)$ of length $\lceil(\omega_4 - 2)/2\rceil$ and $\lfloor(\omega_4 - 2)/2\rfloor$ respectively. For an even ω_4 the first path is $(x_1, x_3, \ldots, x_{\omega_4 - 1})$ and the second path is $(x_2, x_4, \ldots, x_{\omega_4})$ while for an odd ω_4 the first path is $(x_1, x_3, \ldots, x_{\omega_4})$ and the second path is $(x_2, x_4, \ldots, x_{\omega_4 - 1})$. Figure 8 illustrates the planar layout of $G(6)$.

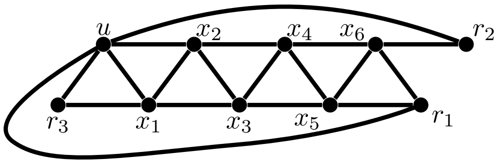


Figure 8 The planar realization of the sequence $G(6) = (5, 4^6, 3, 2^2)$.

Case 3.3: $d_1 \geq 6$ or $d_2 \geq 5$.
If $d_{n-2} \geq 4$, then $\sum d \geq 4n - 2$, which contradicts with $\sum d = 4n - 4$. Therefore, $d_{n-2} = 3$. Construct d' from d by letting $d'_1 = d_1 - 1$, $d'_{n-2} = 2$ and $d'_i = d_i$ for all other $i \in \{2, \ldots, n-3, n-1, n\}$. Then $\mathsf{pos}(d') = d'$, $n' = n$, $\sum d' = 4n' - 6$ and $\omega'_2 = 3$.

Combining $d_3 \geq 4$ and the condition of this case, the maximum degree in d' is at least 5 and the third maximum degree is at least 4. Hence, d' satisfies the conditions of Lemma 11, and therefore it can be realized by an outer-planar graph G'. In the construction of G', there exists a vertex u of degree 2 not adjacent to the vertex v of degree d'_1 by (b) of Lemma 11. Construct the planar graph G realizing d by adding the edge (u, v) to G'.

Summarizing the above three cases, d is planaric. ◀

4.3 The planarity of leaf-free sequences with $\sum d \leq 4n - 4$ and $\omega_2 \geq 3$

This subsection handles the more complex case of "many degrees 2". The analysis is separated into two main parts. First (Lemma 16), we consider sequences of volume $\sum d \leq 4n - 6$. Later (Lemma 17) we analyze the extremal case where $\sum d = 4n - 4$.

▶ **Lemma 16.** *Every graphic sequence d, such that $d_1 \geq d_2 \geq \cdots \geq d_n$, with $\omega_1 = 0$, $\omega_2 \geq 3$, and $\sum d \leq 4n - 6$ is planaric.*

For lack of space, we present an overview of the construction, deferring the complete proof to the full version. The construction of this lemma involves four phases. Phase (1) transforms d into a sequence $\bar{d}^k$ with just two degrees 2 (or, $\bar{d}^k(2) = 3$). This is done by using Procedure `Deg_2_Remove`. Phase (2) constructs an outer-planar graph G_k realizing $\bar{d}^k$, which is illustrated in Figure 9(a). Phase (3) adds a multi-graph $\bar{G}$ with edges connecting the vertices corresponding to indices in the set A defined in Eq. (2). It is illustrated in Figure 9(b). For Phase (4) we need to use the *vertex insertion* operation, defined as follows. For any multi-graph $\hat{G}$, let $I(\hat{G})$ denote the graph obtained from $\hat{G}$ by inserting one new vertex $z_{u,v}$ into every edge $e = (u, v)$ in $E(\hat{G})$, namely, replacing e by the 2-edge path $(u, z_{u,v}, v)$. Note that this operation cancels all parallel edges, so the resulting graph is simple. Observe that if we transform $\bar{G}$ into $I(\bar{G})$, then $I(\bar{G})$ realizes $\mathsf{pos}(d \ominus \bar{d}^k)$ and the combined graph $I(\bar{G}) \cup G_k$ realizes d. Moreover, as G_k is outer-planar, if there are no cross edges between vertices of $\bar{G}$, then $G_k \cup I(\bar{G})$ is a *planar* realization for d, as shown in Figure 9(c). However, the multi-graph $\bar{G}$ might contain crossing edges, as shown in Figure 9(b). In this case, some preliminary processing is needed. At the beginning of Phase (4), apply procedure `Edge_Swap` and replace $\bar{G}$ by a *modified* multi-graph G^M with no edge crossings. Only then, invoke the vertex insertion operation to insert k new vertices into the edges of G^M and get $I(G^M)$. Since the inserted new vertices in $I(G^M)$ do not exist in G_k, the graph $G_k \cup I(G^M)$ is simple. Consequently, $G_k \cup I(G^M)$ can be shown to be a simple planar graph realizing d, as shown in Figure 9(d).

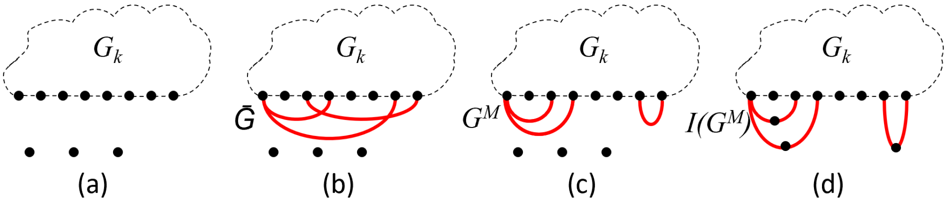


Figure 9 A schematic description of the realization process in Case 4. The preliminary step involves separating the sequence d into a sequence $\bar{d}^k$ with reduced high degrees and only two degrees 2 (depicted by the higher row of vertices in the figures, forming the spine of G_k), and k degrees 2 kept separately (the lower row in the figures).

Careful inspection of the proofs of Lemmas 5, 9 and 16 reveals that when $\sum d \leq 4n - 6 - 2\omega_1$, the constructed realizing graphs are not only planar but also enjoy a 2-page book embedding, yielding Corollary 3.

The next lemma shows the case for $\sum d = 4n - 4$ and $\omega_1 = 0$, $\omega_2 \geq 3$. A schematic description of the construction process in this case is as follows. First apply Procedure `Deg_2_Remove` (Procedure 1 in Section 2) to create a modified degree sequence $\bar{d}^k$ and sets A and B. Next construct a simple graph G_k realizing $\bar{d}^k$, as a combination of an outer-planar graph G' and an edge (u, v), using the method described in the proof of Lemmas 10 and 11. Then, apply procedure `Deg_2_Recover` on G' and output a simple planar graph G. As a final step, construct a graph G'' with $E(G'') = E(G) \cup \{(u, v)\}$. Note that by Procedure `Deg_2_Recover`, $G'' = G_k \cup I(G^M)$. Since $I(G^M)$ realizes $\mathsf{pos}(d \ominus \bar{d}^k)$, G'' realizes d. Since (u, v) does not exist in G, G'' is a simple graph. Combining it with the fact that G is an "almost outer-planaric" graph, if one can show that there are no cross edges between (u, v) and any edges in $E(G^M)$, then G'' is a simple planar graph. The formal proof involves a rather complex case analysis, and is omitted for lack of space.

▶ **Lemma 17.** *Every graphic sequence d with $\omega_1 = 0$, $\omega_2 \geq 3$, and $\sum d = 4n - 4$ is planaric.*

Combining Lemmas 15, 16 and 17 yields the proof for our main Theorem 1.

4.4 Almost tight negative examples

We complement the positive result of Theorem 2 by almost tight negative examples. Consider first the case of $\omega_1 = 0$, for which we present a tight example.

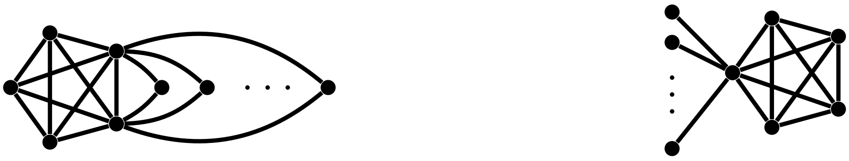
▶ **Lemma 18.** *There exists a graphic sequence of volume $4n - 2$ and $\omega_1 = 0$, which is non-planaric.*

Proof. The sequence $d = (4^5, 2)$ is non-planaric by Lemma 4, and satisfies $\sum d = 4n - 2$. ◀

However, we do not know non-planaric sequences for which $\omega_1 = 0$ and $\sum d = 4n - 2$ for $n > 6$. Instead, the next lemma shows that for any $n \geq 5$ there exists a non-planaric sequence with $\omega_1 = 0$ and $\sum d = 4n$.

▶ **Lemma 19.** *For any non-negative integer k, the sequence $d[k] = ((4 + k)^2, 4^3, 2^k)$, for which $\sum d[k] = 4n$, is graphic but not planaric.*

Proof. See Figure 10a for the unique realization of the sequence $d[k] = ((4 + k)^2, 4^3, 2^k)$. This realization is non-planaric because it has a a K_5 subgraph consisting of the two vertices of degree $4 + k$ and the three vertices of degree 4. This is a unique realization becaus the two degree $4 + k$ vertices each must be connected to all other $n - 1 = 4 + k$ vertices. ◀



(a) The unique realization of $((4 + k)^2, 4^3, 2^k)$.

(b) The unique realization of $(4 + k, 4^4, 1^k)$.

Figure 10 Two non-planaric families of sequences.

Turning to sequences with $\omega_1 > 0$, there is again a small gap. The next lemma shows that for any $n \geq 6$ there exists a non-planaric sequence with $\omega_1 > 0$ and $\sum d = 4n - 2\omega_1$.

▶ **Lemma 20.** *For any non-negative integer k, the sequence $d'[k] = (4 + k, 4^4, 1^k)$, for which $\sum d'[k] = 4n - 2\omega_1$, is graphic but not planaric.*

Proof. See Figure 10b for the unique realization of the sequence $d'[k] = ((4 + k, 4^4, 1^k)$. This realization is non-planaric because it has a K_5 subgraph consisting of the vertex of degree $4 + k$ and the four vertices of degree 4. This is a unique realization becaus the degree $4 + k$ vertex must be connected to all other $n - 1 = 4 + k$ vertices. ◀

Recall that Theorem 2 states that every graphic sequence d such that $\sum_d \leq 4n - 4 - 2\omega_1$ is planaric. In light of Lemma 19 and Lemma 20 it follows that the remaining gap for the case $\omega_1 = 0$ involves sequences d with $\sum d = 4n - 2$ while the remaining gap for the case $\omega_1 > 0$ involves sequences d with $\sum d = 4n - 2 - 2\omega_1$.

References

1 Patrick Adams and Yuri Nikolayevsky. Planar bipartite biregular degree sequences. *Discr. Math.*, 342:433–440, 2019.

2 Amotz Bar-Noy, Toni Böhnlein, David Peleg, Yingli Ran, and Dror Rawitz. Approximate realizations for outerplanaric degree sequences. In *Proc. 35th IWOCA*, 2024.

3 Amotz Bar-Noy, Toni Böhnlein, David Peleg, Yingli Ran, and Dror Rawitz. On key parameters affecting the realizability of degree sequences. In *Proc. 49th MFCS*, 2024.

4 Frank Bernhart and Paul C Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

5 Prosenjit Bose, Vida Dujmović, Danny Krizanc, Stefan Langerman, Pat Morin, David R Wood, and Stefanie Wuhrer. A characterization of the degree sequences of 2-trees. *Journal of Graph Theory*, 58(3):191–209, 2008.

6 SA Choudum. Characterization of forcibly outerplanar graphic sequences. In *Combinatorics and Graph Theory*, pages 203–211. Springer, 1981.

7 Paul Erdös and Tibor Gallai. Graphs with prescribed degrees of vertices [hungarian]. *Matematikai Lapok*, 11:264–274, 1960.

8 Stefano Fanelli. On a conjecture on maximal planar sequences. *Journal of Graph Theory*, 4(4):371–375, 1980.

9 Stefano Fanelli. An unresolved conjecture on nonmaximal planar graphical sequences. *Discrete Mathematics*, 36(1):109–112, 1981.

10 Gautam Gupta, Puneet Joshi, and Amitabha Tripathi. Graphic sequences of trees and a problem of Frobenius. *Czechoslovak Math. J.*, 57:49–52, 2007.

11 S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph –I. *SIAM J. Appl. Math.*, 10(3):496–506, 1962.

12 V. Havel. A remark on the existence of finite graphs [in Czech]. *Casopis Pest. Mat.*, 80:477–480, 1955.

13 AF Hawkins, AC Hill, JE Reeve, and JA Tyrrell. On certain polyhedra. *The Mathematical Gazette*, 50(372):140–144, 1966.

14 Kyle F. Jao and Douglas B. West. Vertex degrees in outerplanar graphs. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 82:229–239, 2012.

15 Zepeng Li and Yang Zuo. On the degree sequences of maximal out erplanar graphs. *Ars Combinatoria*, 2018.

16 A Ramachandra Rao. Degree sequences of cacti. In *Combinatorics and Graph Theory: Proceedings of the Symposium Held at the Indian Statistical Institute, Calcutta, February 25–29, 1980*, pages 410–416. Springer, 1981.

17 S. B. Rao. A survey of the theory of potentially p-graphic and forcibly p-graphic degree sequences. In *Combinatorics and graph theory*, volume 885 of *LNM*, pages 417–440, 1981.

18 E. F. Schmeichel and S. L. Hakimi. On planar graphical degree sequences. *SIAM J. Applied Math.*, 32:598–609, 1977.

19 D.L. Wang and D.J. Kleitman. On the existence of n-connected graphs with prescribed degrees ($n > 2$). *Networks*, 3:225–239, 1973.

20 D.B. West. *Introduction to graph theory.* Prentice Hall, 2001.

21 Mihalis Yannakakis. Four pages are necessary and sufficient for planar graphs. In *18th annual ACM symposium on theory of computing*, pages 104–108, 1986.

The Canadian Traveller Problem on Outerplanar Graphs

Laurent Beaudou
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Pierre Bergé
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Vsevolod Chernyshev
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Antoine Dailly
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Yan Gerard
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Aurélie Lagoutte
Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France

Vincent Limouzy
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Lucas Pastor
Université Clermont-Auvergne, CNRS, Mines de Saint-Etienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Abstract

We study the k-Canadian Traveller Problem, where a weighted graph $G = (V, E, \omega)$ with a source $s \in V$ and a target $t \in V$ are given. This problem also has a hidden input $E_* \subsetneq E$ of cardinality at most k representing blocked edges. The objective is to travel from s to t with the minimum distance. At the beginning of the walk, the blockages E_* are unknown: the traveller discovers that an edge is blocked when visiting one of its endpoints. Online algorithms, also called strategies, have been proposed for this problem and assessed with the competitive ratio, *i.e.*, the ratio between the distance actually traversed by the traveller divided by the distance he would have traversed knowing the blockages in advance.

Even though the optimal competitive ratio is $2k + 1$ even on unit-weighted planar graphs of treewidth 2, we design a polynomial-time strategy achieving competitive ratio 9 on unit-weighted outerplanar graphs. This value 9 also stands as a lower bound for this family of graphs as we prove that, for any $\varepsilon > 0$, no strategy can achieve a competitive ratio $9 - \varepsilon$. Finally, we show that it is not possible to achieve a constant competitive ratio (independent of G and k) on weighted outerplanar graphs.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases Canadian Traveller Problem, Online algorithms, Competitive analysis, Outerplanar graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.19

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 19; pp. 19:1–19:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version *Full Version*: https://arxiv.org/abs/2403.01872 [6]

Funding This work was supported by the International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25 and by the ANR project GRALMECO (ANR-21-CE48-0004).

1 Introduction

The k-*Canadian Traveller Problem* (k-CTP) was introduced by Papadimitriou and Yannakakis [24]. It models the travel through a graph where some obstacles may appear. Given an undirected weighted graph $G = (V, E, \omega)$, with $\omega : E \to \mathbb{Q}^+$, and two of its vertices $s, t \in V$, a traveller walks from s to t on G despite the existence of blocked edges $E_* \subsetneq E$ (also called *blockages*), trying to contain the length of his walk. The traveller does not know which edges are blocked when he begins his journey. He discovers that an edge $e = uv$ is blocked, *i.e.*, belongs to E_*, when he visits one of its endpoints u or v. The parameter k is an upper bound on the number of blocked edges: $|E_*| \leq k$. Several variants have also been studied: where edges are blocked with a certain probability [1, 5, 13, 19], with multiple travellers [11, 25], where we can pay to sense remote edges [19], or where we seek the shortest tour [20, 22]. This problem has applications in robot routing for various kinds of logistics [1, 2, 8, 18, 23].

For a given walk on the graph, its *cost* (also called distance) is the sum of the weights of the traversed edges. The objective is to minimize the cost of the walk used by the traveller to go from s to t. A pair (G, E_*) is called a *road map*. All the road maps considered are feasible: there exists an (s, t)-path in $G \setminus E_*$, the graph G deprived of E_*. In other words, there is always a way to reach target t from source s despite the blockages.

A solution to the k-CTP is an online algorithm, called a *strategy*, which guides the traveller through his walk on the graph : given the input graph, the history of visited nodes, and the information collected so far (here, the set of discovered blocked edges), it tells which neighbor of the current vertex the traveller should visit next. The quality of the strategy can be assessed with competitive analysis [14]. Roughly speaking, the *competitive ratio* is the quotient between the distance actually traversed by the traveller and the distance he would have traversed knowing which edges are blocked in advance. The k-CTP is PSPACE-complete [5, 24] in its decision version that asks, given a positive number r and the input weighted graph, whether there exists a strategy with competitive ratio at most r. Westphal [26] proved that no deterministic strategy achieves a competitive ratio less than $2k + 1$ on all road maps satisfying $|E_*| \leq k$. Said differently, for any deterministic strategy A, there is at least one k-CTP road map for which the competitive ratio of A is at least $2k + 1$. Randomized strategies have also been studied, see *e.g.* [10, 16].

Our goal is to distinguish between graph classes on which the k-CTP has competitive ratio $2k + 1$ (the optimal ratio for general graphs) and the ones for which this bound can be improved. This direction of research has already been explored in [12]: there is a polynomial-time deterministic strategy which achieves ratio $\sqrt{2}k + O(1)$ on graphs with bounded-size maximum (s, t)-cuts. We pursue this study by focusing on a well-known family of graphs: outerplanar graphs, which are graphs admitting a planar embedding (without edge-crossing) where all the vertices lie on the outer face. In [12], an outcome dedicated to a superclass of weighted outerplanar graphs implies that there is a strategy with ratio $2\frac{3}{4}k + O(1)$ on them. Interestingly, however, even very simple unit-weighted planar graphs of treewidth 2, consisting only of disjoint (s, t)-paths, admit the general ratio $2k + 1$ as optimal [15, 26].

Our results and outline. After some preliminaries (Section 2), we describe in Section 3 a polynomial-time strategy achieving a competitive ratio 9 on instances where the input graph is a unit-weighted outerplanar graph:

▶ **Theorem 1.1.** *There is a strategy with competitive ratio 9 for unit-weighted outerplanar graphs.*

In the input outerplanar graph, vertices s and t lie on the outer face. The latter can be seen (provided 2-connectedness) as a cycle embedded in the plane, allowing to explore two sides when we travel from s to t (the two sides are the two internally disjoint (s, t)-paths forming the cycle). The core of the strategy consists in an exploration of both sides via a so-called *exponential balancing*. Then, the most technical part consists in the handling of the chords linking both sides. We maintain a competitiveness invariant of the strategy which produces a final ratio of 9.

Note that Theorem 1.1 can be extended as a corollary to outerplanar graphs where the *stretch*, defined as the ratio between the maximum and minimum weight, is bounded by some fixed S. In this case, the strategy has ratio $9S$.

Surprisingly, the k-CTP on unit-weighted outerplanar graphs has connections with another online problem called the *linear search* problem [3, 7, 9] or the *cow-path* problem [21]. In this problem, a traveller walks on an infinite line, starting at some arbitrary point, and its goal is to reach some target fixed by the adversary. It was shown that applying an exponential balancing on this problem is the optimal way, from the worst case point of view, to reach the target [3]. We explain in Section 3.3 why, on unit-weighted outerplanar graphs, the competitive ratio stated in Theorem 1.1 is optimal and how it can be deduced from the literature on the linear search problem.

▶ **Theorem 1.2.** *For any $\varepsilon > 0$, no deterministic strategy achieves competitive ratio $9 - \varepsilon$ on all road maps (G, E_*), where G is a unit-weighted outerplanar graph.*

Finally, in Section 4, we show that no constant competitive ratio can be achieved on outerplanar graphs where weights can be selected arbitrarily.

▶ **Theorem 1.3.** *There is no constant C, independent from G and k, such that a deterministic strategy achieves competitive ratio C on all road maps (G, E_*) where G is a weighted outerplanar graph.*

We summarize in Table 1 the state-of-the-art of the competitive analysis of deterministic strategies for the k-CTP, giving for each family an upper bound of competitiveness (*i.e.*, a strategy with such ratio exists) and a lower bound (*i.e.*, no strategy can achieve a smaller ratio). Our contributions are framed.

Due to space limitation, the proofs of results marked with **(*)** are omitted here and available in the full version [6].

2 Definitions and first observations

2.1 Graph preliminaries

We work on undirected connected weighted graphs $G = (V, E, \omega)$, where $\omega : E \to \mathbb{Q}^+$. A graph is *equal-weighted* (resp. *unit-weighted*) if the value of $\omega(e)$ is the same (resp. 1) for every edge $e \in E$. This article follows standard graph notations from [17]. We denote by $G[U]$ the subgraph of G induced by $U \subseteq V$: $G[U] = (U, E[U], \omega_{|E[U]})$; and by $G \setminus U$ the graph deprived of vertices in U: $G \setminus U = G[V \setminus U]$. A *simple* (u, v)*-path* is a sequence of

Table 1 Deterministic strategies performances for the k-CTP.

Family of graphs	upper bound	lower bound
unit-weighted planar of treewidth 2	$2k + 1$ [26]	$2k + 1$ [15, 26]
bounded maximum edge (s, t)-cuts	$\sqrt{2}k + O(1)$ [12]	?
outerplanar	$2^{\frac{3}{4}}k + O(1)$ [12]	$\boxed{\text{not constant}}$
unit-weighted outerplanar	$\boxed{9}$	$\boxed{9}$

pairwise different vertices between u and v, while, in a (u, v)-*walk*, vertices can be repeated. The *cost* (or *traversed distance*) of a walk or a path is the sum of the weights of the edges it traverses. A vertex v is an *articulation point* if $G \setminus \{v\}$ is not connected.

An (s, t)-*separator* $X \subsetneq V \setminus \{s, t\}$ in graph G is a set of vertices such that s and t are disconnected in graph $G \setminus X$. We denote by $R_G(s, X)$ (resp. $R_G(t, X)$) the *source* (resp. *target*) *component* of separator X, which is a set made up of the vertices of X together with all vertices reachable from s (resp. t) in $G \setminus X$.

A graph is *outerplanar* if it can be embedded in the plane in such a way that all vertices are on the outer face. An outerplanar graph is 2-connected if and only if the outer face forms a cycle. Given an embedding of a 2-connected outerplanar graph $G = (V, E)$ and two vertices s and t, let $s \cdot p_1 \cdot p_2 \cdots p_h \cdot t \cdot q_1 \cdot q_2 \cdots q_\ell \cdot s$ be the cycle along the outer face of G and let $S_1 = \{p_1, p_2, \ldots, p_h\}$ and $S_2 = \{q_1, q_2, \ldots, q_\ell\}$ with $V = \{s, t\} \cup S_1 \cup S_2$. We can slightly deform the embedding so that s and t are aligned along the horizontal axis; since the outer face forms a cycle, we will refer to S_1 (resp. S_2) as the *upper* (resp. *lower*) *side* of G. A chord xy of the cycle formed by the outer face is said to be (s, t)-*vertical* (resp. (s, t)-*horizontal*) if x and y belong to different sides (resp. to the same side), see Figure 1. When $x = s$ or $y = t$, the chord is considered as (s, t)-horizontal and not (s, t)-vertical. Any (s, t)-vertical chord (simply *vertical chord* when the context is clear) is an (s, t)-separator. Considering a set of vertical chords, we say that the *rightmost* one has the minimal inclusion-wise target component. Due to planarity, the rightmost vertical chord is unique for any such set.

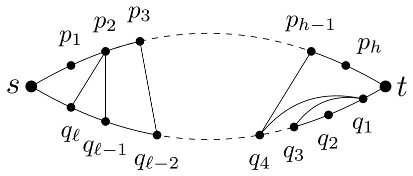


Figure 1 Example of an outerplanar graph: p_2q_ℓ, $p_2q_{\ell-1}$, $p_3q_{\ell-2}$, and $p_{h-1}q_4$ are vertical chords and q_1q_3, q_1q_4 are horizontal chords.

2.2 Problem definition and competitive analysis

Let $G = (V, E, \omega)$ be a graph and E_* represent a set of blocked edges. A pair (G, E_*) is a *road map* if s and t are connected in $G \setminus E_*$.

▶ **Definition 2.1** (k-CTP).

Input: *A graph $G = (V, E, \omega)$, two vertices $s, t \in V$, and a set E_* of blocked edges which are unknown such that $|E_*| \leq k$ and (G, E_*) is a road map.*

Objective: *Traverse graph G from s to t with minimum cost.*

A solution to the k-CTP is an (s,t)-walk. The set of blocked edges E_* is a hidden input at the beginning of the walk. We say an edge is *revealed* when one of its endpoints has already been visited. A *discovered blocked edge* is a revealed edge which is blocked. At any moment of the walk, we usually denote by $E'_* \subseteq E_*$ the set of discovered blocked edges, in other words the set of blocked edges for which we visited at least one endpoint. During the walk, we are in fact working on $G \setminus E'_*$ as discovered blocked edges can be removed from G.

We call a path *blocked* if one of its edges was discovered blocked; *apparently open* if no blocked edge has been discovered on it for now (it may contain a blocked edge which has not been discovered yet); *open* if we are sure that it does not contain any blocked edge (either all of its edges were revealed open, or it is apparently open and $|E'_*| = k$, or by connectivity considerations since s and t must stay connected in road maps).

For any $F \subseteq E_*$ and two vertices x, y of G, let $d_F\left(G, x, y\right)$ be the cost of the shortest (x,y)-path in graph $G \setminus F$. If the context is clear, we will use $d_F\left(x, y\right)$.

We denote by P_{opt} some *optimal offline path* of road map (G, E_*): it is one of the shortest (s,t)-paths in the graph $G \setminus E_*$. Its cost, the *optimal offline cost*, given by $d_{\text{opt}} = d_{E_*}\left(s, t\right)$, is the distance the traveller would have traversed if he had known the blockages in advance. Given a strategy A for the k-CTP, the *competitive ratio* [14] $c_A(G, E_*)$ over road map (G, E_*) is defined as the ratio between the cost $d_A^{\text{Tr}}\left(G, E_*\right)$ of the traversed walk and d_{opt}. Formally:

$$c_A(G, E_*) = \frac{d_A^{\text{Tr}}\left(G, E_*\right)}{d_{\text{opt}}}.$$

Given a monotone family of graphs $\mathcal{F}$ (i.e. closed under taking subgraph), we say that a strategy A admits a competitive ratio $c(k)$ for the family $\mathcal{F}$ if it is an upper bound for all values $c_A\left(G, E_*\right)$ over all k-CTP road maps (G, E_*) such that $G \in \mathcal{F}$. Conversely, we say that some ratio $c(k)$ cannot be achieved for family $\mathcal{F}$ if, for every strategy A, there is a road map (G, E_*) with $G \in \mathcal{F}$ such that $c_A(G, E_*) > c(k)$.

Westphal [26] identified, for any integer k, a relatively trivial family of graphs for which any deterministic strategy achieves ratio at least $2k + 1$. These graphs are made up of only $k + 1$ identical disjoint (s,t)-paths: they are planar and have treewidth 2. As those paths are indistinguishable, the traveller might have to traverse k of them before finding the open one. This outcome still works if we restrict ourselves to unit weights [15]. Conversely, there are two strategies in the literature achieving competitive ratio $2k + 1$ on general graphs: REPOSITION [26] and COMPARISON [27].

Note that articulation points allow a preliminary decomposition and simplification of any input graph, before even exploring:

▶ **Lemma 2.2.** *Let $\mathcal{F}$ be a monotone family of graphs, and assume that we have a strategy A achieving competitive ratio C on graphs of $\mathcal{F}$ that do not contain any articulation point. Then, there exists a strategy A' achieving the same competitive ratio C on all graphs of $\mathcal{F}$.*

Proof. The strategy A' goes as follows: let (G, E_*) be a road map with $G \in \mathcal{F}$. If G does not contain any articulation point, apply strategy A. Otherwise, let z be an articulation point of G. If $\{z\}$ is not an (s,t)-separator, then, recursively apply strategy A' on $R_G(s, \{z\})$, which is both the source and the target component, to reach t from s. Otherwise (so $\{z\}$ is an (s,t)-separator), recursively apply strategy A' on the source component $R_G(s, \{z\})$ to reach z from s, then recursively apply strategy A' on the target component $R_G(t, \{z\})$ to reach t from z. The procedure is illustrated in Figure 2.

We prove by induction on the number p of articulation points that A' terminates and achieves competitive ratio C. The base case $p = 0$ holds by property of A. For the inductive step, we distinguish two cases. If $\{z\}$ is not an (s,t)-separator, the walk we obtain is of

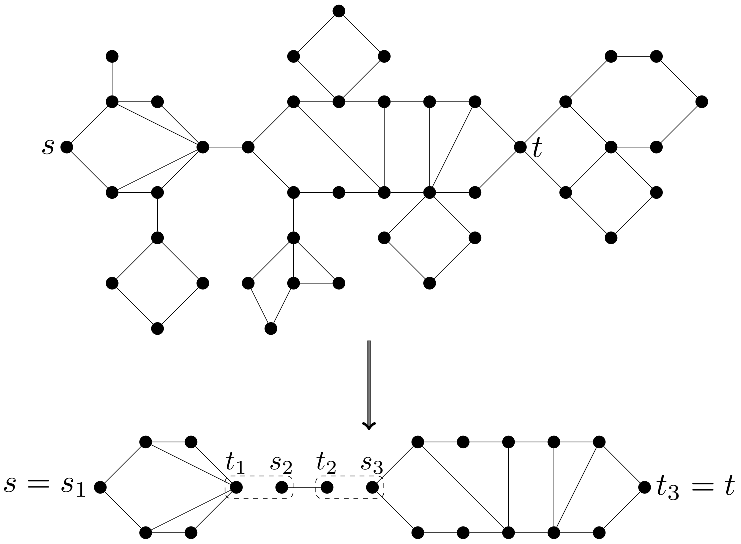


Figure 2 Decomposing the graph into components with no articulation points and removing the useless components (the vertices in a dashed rectangle are the same in the original graph).

length at most Cd_{opt}, which gives competitive ratio C. Otherwise, the length of the whole walk at most $Cd_{E_*}(s,z) + Cd_{E_*}(z,t)$. Since z is an (s,t)-separator, $z \in P_{\text{opt}}$ and we have $d_{\text{opt}} = d_{E_*}(s,z) + d_{E_*}(z,t)$, which concludes the proof. ◀

3 Optimal competitive ratio 9 for unit-weighted outerplanar graphs

We propose a polynomial-time strategy called ExpBalancing dedicated to unit-weighted outerplanar graphs. We show that it achieves competitive ratio 9 for this family of graphs, which we will later prove is optimal (see Theorem 1.2).

3.1 Presentation of the strategy

First, note that Lemma 2.2 allows us to work on outerplanar graphs without articulation points. The input is a unit-weighted 2-connected outerplanar graph G and two vertices s and t. We provide a detailed description of the strategy ExpBalancing that we follow to explore the graph G.

1. **Reaching t.** If, at any point in our exploration, we reach t, then we exit the algorithm and return the processed walk.
2. **Horizontal chords treatment.** If, at any point in our exploration, we visit a vertex $u \in S_i$, $i \in \{1,2\}$, incident with an open horizontal chord uv revealed for the first time, then we can remove all the vertices on side S_i that lie between u and v on the outer face. Said differently, we get rid of the vertices which are surrounded by the chord uv. If several horizontal chords incident with u are open, then it suffices to apply this rule to the chord which surrounds all others. This procedure comes from the observation that, due to both unit weights and planarity, the open horizontal chord uv with the rightmost v is necessarily the shortest way to go from u to t on side S_i, and thus visiting the vertices surrounded by it will occur an extra, useless cost.

3. **Exponential balancing.** The core *exponential balancing* principle of the strategy consists in alternately exploring sides within a given budget that doubles each time we switch sides. The budget is initially set to 1. Hence, we walk first on side S_1 with budget 1, second on side S_2 with budget 2, then on side S_1 with budget 4, and so on. We say each budget corresponds to an *attempt*. During each attempt, we traverse a path starting from the source s and stay exclusively on some side S_i, $i \in \{1, 2\}$. As evoked in the previous step, at each newly visited vertex, we use an open horizontal chord from our position which brings us as close as possible to t on our side. Either a horizontal chord is open and we use the one which surrounds all other open chords, or if no such chord is open, we pursue our walk on the outer face.

   This balancing process can be described on an automaton depicted in Figure 3 which will be particularly useful in the analysis of this strategy. Here, we assume that we neither are completely blocked on one side nor reveal an open vertical chord. We will handle these cases in Steps 4-6.

   We start our walk on s (state $\mathbf{E_1}$), make an attempt on an arbitrary side (say S_1) with budget 1 (state $\mathbf{E_2}$), and decide to come back to s if t was not reached. During our first attempt on side S_2 with budget 2, we cross a first edge and reach state **A**. Then, we cross a second edge if we are not blocked, but this part of the journey corresponds to the transition between states **A** and **B**. The automaton works as follows:

   - In state **A**, we have explored D vertices on each side (in the description above, $D = 1$ when we first arrive in state **A**). Call x and y the last explored vertices on each side, assume we are on x. The current budget is $2D$ and we pursue our attempt on the side of x.
   - We then explore at most D more vertices on the side of x. We reach state **B**.
   - We then go back to y through s, reaching state **C**.
   - We explore at most D more vertices on the side of y. We go back to state **A** with an updated value of D that is doubled, update x and y, and the sides are switched.

4. **Bypassing a blocked side.** If, during some attempt on side S_i, we are completely blocked (there is no open (s, t)-path on $G[S_i] \setminus E'_*$) before reaching the budget, hence exploring αD ($\alpha < 1$) instead of D (see Figures 4a and 4b), then we backtrack to s and pursue the balancing on the other side S_j ($j \in \{1, 2\}, j \neq i$). However, we forget any budget consideration: we travel until we either reach t or visit the endpoint u of some open vertical chord uv. In case there are several open vertical chords incident with u revealed at the same time, we consider the rightmost one. At this moment, we update the current graph $G \setminus E'_*$ by keeping only the target component of separator $\{u, v\}$ and considering u as a new source. Concretely, we concatenate the current walk computed before arriving at u with a recursive call of ExpBalancing on input $(G[R_G(t, \{u, v\})], u, t)$.

5. **Handling open vertical chords between states A and B.** If, during some attempt on side S_i, especially in the transition between states **A** and **B**, we reveal an open vertical chord uv, $u \in S_i$, after having explored distance αD (parameter α is rational, $0 < \alpha \leq 1$, but αD is an integer), then we go to the other side S_j, $j \neq i$, through uv and explore side S_j from v towards s until we:

   - either "see" a vertex y already visited after distance βD (we fix $\beta D \leq \alpha D - 1$, so $0 \leq \beta < \alpha$),
   - or explore distance $\alpha D - 1$ and do not see any already visited vertex,
   - or are completely blocked on S_j before we reach distance $\alpha D - 1$.

   By "see", we mean that we can reach - or not - a neighbor of y which reveals the status of the edge between them: in this way, we actually know the distance to reach y from v even if we did not visit v. Figure 4c describes this rule with an example.

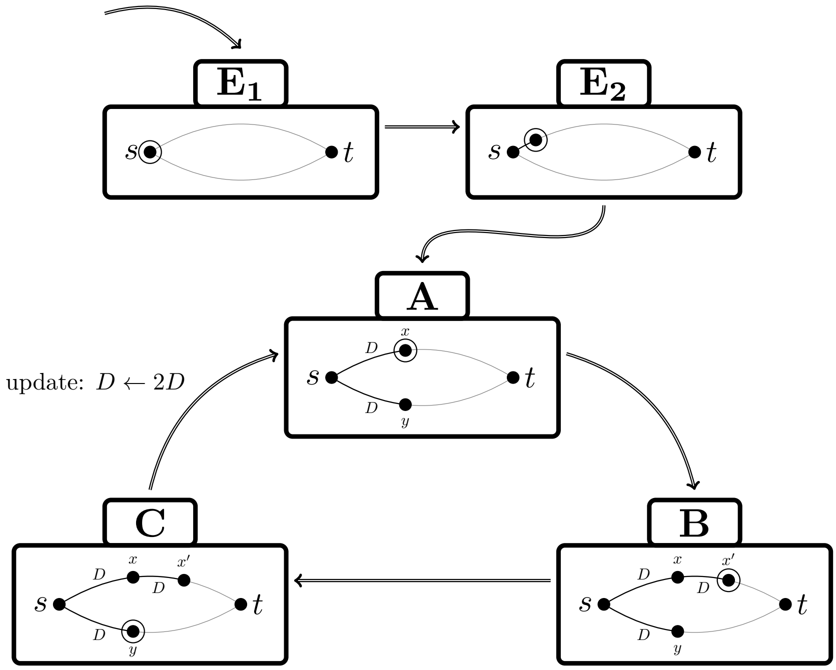


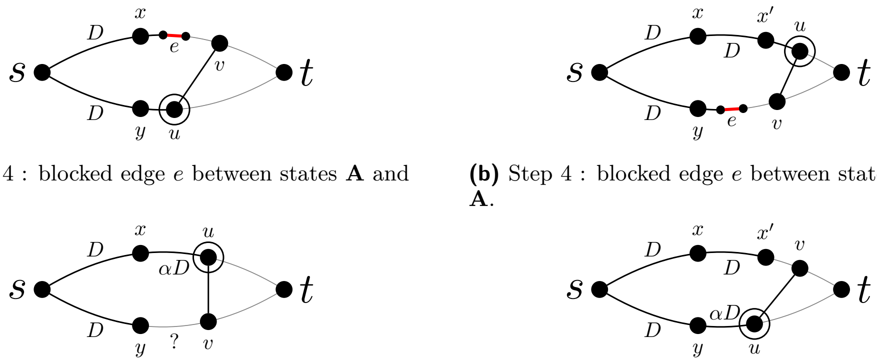
Figure 3 Representation of the exponential balancing divided into three different states. The circled vertex is the one we are currently exploring.

The role of this procedure is to know which endpoint of the chord is the closest to s. If we see, after distance $\beta D = \alpha D - 1$, an already visited vertex (denoted by y in Figure 4c) at distance αD from v, then, we continue the exponential balancing: we go back to v and thus to state **A** in the automaton, update the budget value D which becomes $D + \alpha D$.

Otherwise, we update G by keeping only the target component of separator $\{u, v\}$. The current graph becomes $G' = G[R(t, \{u, v\})]$. If we saw an already visited vertex $y \in S_j$ by exploring distance $\beta D < \alpha D - 1$, then the new source becomes $s' = v$. Otherwise, the new source is $s' = u$. We concatenate the current walk with the walk returned by applying ExpBalancing on input (G', s', t).

6. **Handling open vertical chords between states C and A.** If, during the transition between states **C** and **A** (when some attempt is launched on the side of y and the traversed distance on the other side is larger, see Figure 3), an open vertical chord uv is revealed (see Figure 4d), then we keep only the target component of $\{u, v\}$ and set u as the new source. More formally, we concatenate the current walk with the walk returned by applying ExpBalancing on input (G', u, t), where $G' = G[R_G(t, \{u, v\})]$.

Steps 4–6 can be summarized in this way: when we reveal an open vertical chord uv such that $d_{E_*}(s, v) = d_{E_*}(s, u) + 1$, we launch a recursive call on the target component of separator $\{u, v\}$ with source u and target t. Indeed, any optimal offline path must pass through separator $\{u, v\}$ and, as $d_{E_*}(s, v) = d_{E_*}(s, u) + 1$, we can say there is one optimal offline path P_{opt} such that $u \in P_{\text{opt}}$. Hence, it makes sense to select u as a new source, there is no interest in visiting vertices different from $\{u, v\}$ belonging to their source component.



(a) Step 4 : blocked edge e between states **A** and **B**.

(b) Step 4 : blocked edge e between states **C** and **A**.

(c) Step 5 : open vertical chord uv between states **A** and **B**.

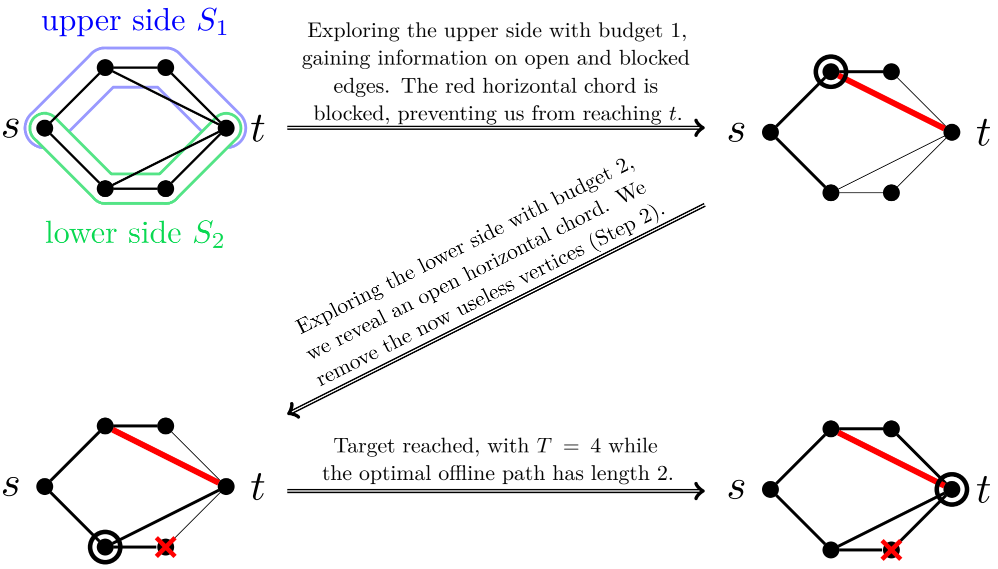
(d) Step 6 : open vertical chord uv between states **C** and **A**.

Figure 4 Four situations potentially met with ExpBalancing on some unit-weighted outerplanar graph.

3.2 Competitive analysis

We now show that the strategy ExpBalancing presented above has competitive ratio 9 on unit-weighted outerplanar graphs. We prove this statement by minimal counterexample. In this subsection, let G denote the smallest (by number of vertices, then number of edges) unit-weighted outerplanar graph on which ExpBalancing does not achieve competitive ratio 9. We will see that the existence of such a graph G necessarily implies a contradiction.

Examples of executions of ExpBalancing are given in Figures 5 and 6.



Exploring the upper side with budget 1, gaining information on open and blocked edges. The red horizontal chord is blocked, preventing us from reaching t.

Exploring the lower side with budget 2, we reveal an open horizontal chord. We remove the now useless vertices (Step 2).

Target reached, with $T = 4$ while the optimal offline path has length 2.

Figure 5 Application of ExpBalancing on the first graph of the decomposition of Figure 2. At each step, the circled vertex is the one we are currently exploring, and we know the status of the bold edges: black is open, red is blocked.

The two following technical lemmas prove that a recursive call has to happen when ExpBalancing is applied on G (Lemma 3.1) and that such a recursive call implies certain properties (Lemma 3.2).

▶ **Lemma 3.1.** *During the execution of* ExpBalancing, *let T be the distance travelled at a given point before the first recursive call (if any). Then, $T \leq 9d_{\text{opt}}$. Moreover, if we are in state* **A**, *let x and y be the last two vertices explored on each side during the exponential balancing. Then: (i) $d_{E_*}(s,x) = D$, (ii) $d_{E_*}(s,y) = D$ and (iii) $T \leq 5D$.*

Proof. Assume that we have applied ExpBalancing on G until a certain point and that no recursive call was launched so far. We first focus on the second part of the invariant we want to show:

$$\text{In state } \mathbf{A},\ (i)\ d_{E_*}(s,x) = D,\ (ii)\ d_{E_*}(s,y) = D \text{ and } (iii)\ T \leq 5D.$$

Items (i) and (ii) are true, since no shortcut between s and either x or y can exist: any open horizontal chord is used, and an open vertical chord opening up a shortcut leads to a recursive call (Steps 5 and 6).

Item (iii) is trivially true when we kick-start the exponential balancing: when entering **A** from $\mathbf{E_2}$, we have $T = 3$ and $d_{E_*}(s,x) = d_{E_*}(s,y) = 1$. Assume that it is true for a given $D \geq 1$, and let T_0 be the value of T at this point. When we reach state **B**, we have $T = T_0 + D \leq 6D$. When we reach state **C**, we have $T = T_0 + D + 3D \leq 9D$. In brief, from state **A** to **C**, we have $d_{\text{opt}} \geq D$ as distance D was explored on both sides without reaching t. The largest ratio of T by D on these phases is 9 at state **C**, where we have $T \leq 9d_{\text{opt}}$.

During the transition from **C** to **A**, if $D + \alpha D$ denotes the traversed distance on current side at any moment (see Figure 3), then $d_{\text{opt}} \geq D + \alpha D$ and $T = 9D + \alpha D$. The ratio $\frac{T}{d_{\text{opt}}}$ admits a decreasing upper bound, from 9 in state **C** to 5 in **A**. Indeed, when we are back to state **A**, we have $T = T_0 + D + 3D + D$, but the value of D is updated. Let $D' = 2D$. We have $T = T_0 + 5D \leq 5D + 5D = 5D'$, and so item (iii) remains true during the core loop.

We also have to check that it is true when we met an open vertical chord uv between states **A** and **B** which satisfies $d_{E_*}(s,v) = d_{E_*}(s,u)$ (case $\beta D = \alpha D - 1$ in Step 5). In this case, the new value of D is $D' = D + \alpha D$ and we have $T \leq 5D + \alpha D + 1 + 2\alpha D \leq 5(D + \alpha D) = 5D'$ (since $\alpha D \geq 1$), so item (iii) remains true.

Thus, conditions (i)-(iii) hold in state **A**, and we always (during all states and transitions between them) have $T \leq 9d_{\text{opt}}$, hence the statement holds. ◀

▶ **Lemma 3.2.** *Assume that we are currently executing* ExpBalancing *on G and that a recursive call is launched after revealing the vertical chord uv with new source u. Let T be the distance traversed before the recursive call. Then, either $T > 9d_{E_*}(s,u)$ or $d_{E_*}(s,v) < d_{E_*}(s,u) + 1$.*

Proof. If $d_{E_*}(s,v) \geq d_{E_*}(s,u) + 1$, following the rules established in Steps 4-6, we will launch a recursive call on the target component of $\{u,v\}$ with new source u. Hence, we will have $d^{\text{Tr}}_{\text{EXP}}(G, E_*) = T + T'$, where $T' \leq 9d_{E_*}(u,t)$ by minimality of G and EXP abbreviates ExpBalancing. By way of contradiction, suppose that $T \leq 9d_{E_*}(s,u)$. The optimal offline path P_{opt} necessarily goes through the separator $\{u,v\}$ in graph G and, since $d_{E_*}(s,v) = d_{E_*}(s,u) + 1$, u belongs to some optimal offline path. Consequently, $T + T' \leq 9(d_{E_*}(s,u) + d_{E_*}(u,t)) = 9d_{E_*}(s,t)$. ◀

We are now ready to prove the major contribution of this article.

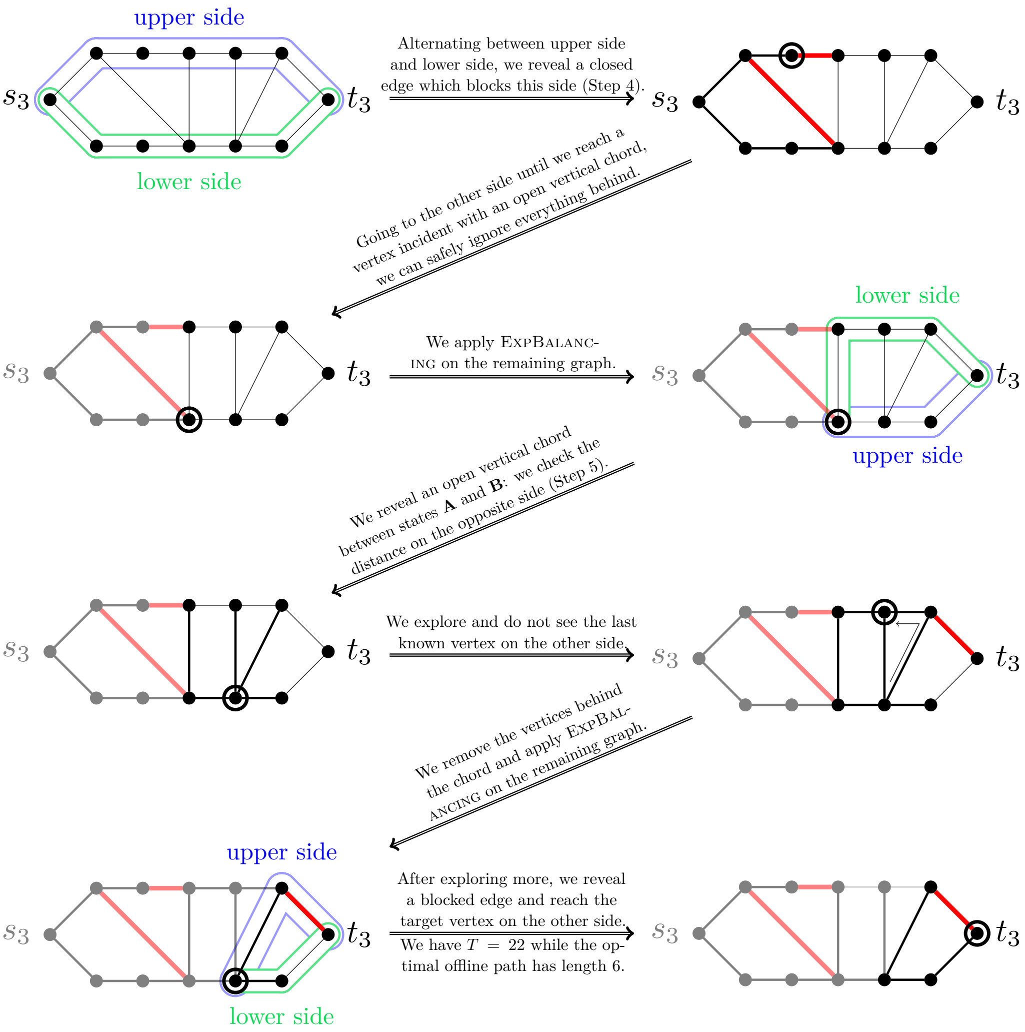

upper side
s_3
t_3
lower side
Alternating between upper side and lower side, we reveal a closed edge which blocks this side (Step 4).
s_3
t_3
Going to the other side until we reach a vertex incident with an open vertical chord, we can safely ignore everything behind.
s_3
t_3
We apply ExpBalancing on the remaining graph.
lower side
s_3
t_3
upper side
We reveal an open vertical chord between states **A** and **B**: we check the distance on the opposite side (Step 5).
s_3
t_3
We explore and do not see the last known vertex on the other side.
s_3
t_3
We remove the vertices behind the chord and apply ExpBalancing on the remaining graph.
upper side
s_3
t_3
lower side
After exploring more, we reveal a blocked edge and reach the target vertex on the other side.
We have $T = 22$ while the optimal offline path has length 6.
s_3
t_3

Figure 6 Application of ExpBalancing on the third graph of the decomposition of Figure 2. At each step, the circled vertex is the one we are currently exploring, and we know the status of the bold edges: black is open, red is blocked.

▶ **Theorem 1.1.** *There is a strategy with competitive ratio 9 for unit-weighted outerplanar graphs.*

Proof. A direct consequence of Lemma 3.1 is that, during some attempt, ExpBalancing will launch a recursive call on G (otherwise, it has competitive ratio 9, a contradiction). Let T be the distance traversed before the recursive call. Lemma 3.2 has an important consequence: if we launch a recursive call on the open vertical chord uv with new source u and can guarantee that both $d_{E_*}(s, v) = d_{E_*}(s, u) + 1$ and $T \leq 9d_{E_*}(s, u)$, then, we have a contradiction. According to the description of ExpBalancing, a recursive call is launched when we are sure that $d_{E_*}(s, v) = d_{E_*}(s, u) + 1$: this concerns Step 4, Step 5 when $\beta D < \alpha D - 1$ and Step 6.

Assume first that we are blocked on one side between states **A** and **B** in Step 4 (see Figure 4a). We know that $d_{E_*}(s,v) = d_{E_*}(s,u) + 1$ because u is an articulation point of $G \setminus E'_*$. Also, $d_{E_*}(s,u) = D + d_{E_*}(y,u)$. Using Lemma 3.1:

$$\begin{aligned} T &\leq (5D+\alpha D) + (\alpha D + 2D) + d_{E_*}(y,u) \\ &\leq (7+2\alpha)D + d_{E_*}(y,u) \\ &\leq 9(D + d_{E_*}(y,u)) && (\alpha \leq 1) \\ &\leq 9d_{E_*}(s,u) \end{aligned}$$

which, by Lemma 3.2, leads to a contradiction.

Assume now that we are blocked on one side between states **C** and **A** in Step 4 (see Figure 4b). Let x' be the last vertex reached at the end of state **A**, we know that $d_{E_*}(s,v) = d_{E_*}(s,u) + 1$ because u is an articulation point of $G \backslash E'_*$ and $d_{E_*}(s,u) = 2D + d_{E_*}(x',u)$. Using Lemma 3.1:

$$\begin{aligned} T &\leq (9D+\alpha D) + (\alpha D + 3D) + d_{E_*}(x',u) \\ &\leq (12+2\alpha)D + d_{E_*}(x',u) \\ &\leq 9(2D + d_{E_*}(x',u)) \leq 9d_{E_*}(s,u) && (\alpha \leq 1) \end{aligned}$$

which, by Lemma 3.2, leads to a contradiction.

Assume now that we reveal an open vertical chord uv between states **A** and **B** in Step 5 (see Figure 4c). Recall that $d_{E_*}(s,u) \leq D + \alpha D$, and we explore up to distance $\alpha D - 1$ towards y. There are two possibilities: either we see y by exploring distance βD (with $\beta D < \alpha D - 1$), or we do not see y even if we explore distance $\alpha D - 1$.

If we see y, then, we know that $d_{E_*}(s,u) = d_{E_*}(s,v) + 1$ since going to u through x will yield distance $D + \alpha D$ while going through y and v will yield distance at most $D + \beta D + 2$, and we know that $\beta D < \alpha D - 1$ and $\beta D \geq 0$. So, $d_{E_*}(s,v) = D + \beta D + 1$. Using Lemma 3.1:

$$\begin{aligned} T &\leq (5D+\alpha D) + (1 + 2(\beta D + 1)) \\ &\leq (5+\alpha+2\beta)D + 3 \\ &\leq 9(D + \beta D + 1) \leq 9d_{E_*}(s,v) && (\beta < \alpha \leq 1) \end{aligned}$$

which, by Lemma 3.2 leads to a contradiction (the roles of u and v are reversed here, since v is the new source).

If we do not reach y, either by blocked edges or because we have explored distance $\alpha D - 1$ without reaching it, then, we know that $d_{E_*}(s,v) = d_{E_*}(s,u) + 1$. Using Lemma 3.1:

$$\begin{aligned} T &\leq (5D+\alpha D) + (1 + 2(\alpha D - 1) + 1) \\ &\leq (5+3\alpha)D \\ &\leq 9(D + \alpha D) \leq 9d_{E_*}(s,u) && (\alpha \leq 1) \end{aligned}$$

which, by Lemma 3.2, leads to a contradiction.

Finally, assume that we reveal an open vertical chord uv between states **C** and **A** after having explored αD vertices in Step 6 (see Figure 4d). Since uv was not revealed before, this implies that the shortest path from s to v goes through u, and so $d_{E_*}(s,v) = d_{E_*}(s,u) + 1$. Using Lemma 3.1:

$$\begin{aligned} T &\leq 9D + \alpha D \\ &\leq 9(D + \alpha D) \leq 9d_{E_*}(s,u) && (\alpha \leq 1) \end{aligned}$$

which, by Lemma 3.2, leads to a contradiction.

All the possible cases lead to contradictions, and so such a G cannot exist. ExpBalancing thus achieves competitive ratio 9 on unit-weighted outerplanar graphs. ◀

3.3 Lower bound 9 for unit-weighted outerplanar graphs

In this subsection we prove that the competitive ratio achieved with the ExpBalancing strategy is optimal on unit-weighted outerplanar graphs.

▶ **Theorem 1.2.** *For any $\varepsilon > 0$, no deterministic strategy achieves competitive ratio $9 - \varepsilon$ on all road maps (G, E_*), where G is a unit-weighted outerplanar graph.*

This result can be obtained by a natural reduction from the linear search problem [9] (or, equivalently, the cow-path problem on two rays [21]). The *linear search problem* is defined as follows: an immobile hider is located on the real line. A searcher starts from the origin and wishes to discover the hider in minimal time. The searcher cannot see the hider until he actually reaches the point at which the hider is located and the time elapsed until this moment is the duration of the game.

This problem reduces to the k-CTP on specific road maps that we call *shell road maps*. The *shell graph* on $2n$ vertices, denoted by Sh_n (see Figure 7), is the graph obtained from a cycle on $2n$ vertices $\{v_0, v_1, \ldots, v_{2n-1}\}$ with all possible chords incident with vertex v_n, except v_0v_n. It is clearly outerplanar, and all edge weights are set to 1. In our setting, we shall consider v_0 as the source s and v_n as the target t. We call *shell road maps* the specific road maps (Sh_n, E_*) where E_* is made up only of edges incident with t. Said differently, the traveller cannot be blocked on the outer face on some edge v_iv_{i+1}.

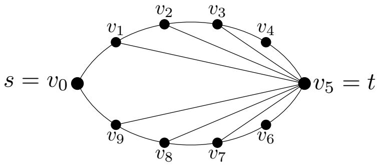


Figure 7 The shell graph on 10 vertices Sh_5.

The shell graph is 2-connected, so it contains an upper side S_1 and a lower side S_2 which can simulate the positive and the negative sides of the real line. The position of the hider will then intuitively correspond to the first encountered open chord to t : if the hider is at position $x > 0$ (resp. $x < 0$), then E_* will contain all $v_it \in E$ except $v_{\lceil x \rceil}t$ (resp. $v_{\lfloor 2n+x \rfloor}t$). In such a way, any strategy for the k-CTP with some competitive ratio r, will give a strategy for the linear search problem with asymptotic competitive ratio $r + \varepsilon$ for any $\varepsilon > 0$. However, it is known that the linear search problem has an optimal ratio of 9 [3] which gives the lower bound we want on the k-CTP. Note however that, in the sketched reduction, small details need to be cared of, for example the distance of the traveller has a unit additive term compared to the searcher on the line (cost of crossing the discovered chord to t). In order to remove any doubt related to these details, we provide in the full version [6] a complete proof of Theorem 1.2 (without reducing to the linear search, but sharing some features with the proof of [3]).

4 The case of arbitrarily weighted outerplanar graphs

Given our results on the unit-weighted case (which give as an easy corollary ratio $9S$ for fixed stretch S), a natural question is whether we can design a deterministic strategy achieving a constant competitive ratio for the more general family of arbitrarily weighted outerplanar graphs. In this section, we prove that this is impossible since, for any constant $C \geq 1$, there exists a weighted outerplanar graph on which the competitive ratio obtained is necessarily greater than C. Let us introduce a sub-family of outerplanar graphs that will be useful here.

▶ **Definition 4.1.** *An outerplanar graph G containing s and t is said to be* (s,t)-unbalanced *if either it is a single st edge or one of its sides contains all vertices V of the graph.*

In other words, an (s,t)-unbalanced outerplanar graph is such that s and t are neighbors on the outer face. While one side (say w.l.o.g. the lower side) contains all vertices, the upper one only contains s and t and simply consists of a single edge st. Thus, such a graph does not have any vertical chord. We show in the remainder that constant competitive ratio cannot be obtained even on weighted (s,t)-balanced outerplanar graphs.

We begin with the definition of a graph transformation $\mathcal{T}$ which takes as input a weighted (s,t)-unbalanced outerplanar graph $H = (V, E, \omega)$, three positive rational values α, C, and η, and an integer N. The construction of the output graph $\mathcal{T}(H, \alpha, C, \eta, N)$ works as follows:

- Create two vertices s and t with an edge st of weight C. This edge will stand as the upper side of the graph.
- Add N copies of the graph αH, where $\alpha H = (V_\alpha, E_\alpha, \omega_\alpha)$ is a graph such that $V_\alpha = V$, $E_\alpha = E$ and $\omega_\alpha(e) = \alpha\omega(e)$ for every edge $e \in E$. These copies are denoted by $\alpha H^{(1)}, \dots, \alpha H^{(N)}$ and the source/target pair of each $\alpha H^{(j)}$ is denoted by (s_j, t_j).
- Connect in series all copies $\alpha H^{(1)}, \dots, \alpha H^{(N)}$ from s to t in order to form the lower side of the graph, using their source/target as input/output vertices. In brief, merge s with s_1, t_i with s_{i+1} for $i \in \{1, \dots, N-1\}$, and t_N with t.
- Add all edges $t_j t$ for $1 \le j \le N-1$ with weight η.

Figure 8 illustrates the graph $\mathcal{T}(H, \alpha, C, \eta, N)$ obtained. Observe that it is an (s,t)-unbalanced outerplanar graph because the lower side of each αH contains all its own vertices. Therefore, all vertices of $\mathcal{T}(H, \alpha, C, \eta, N)$ lie on its lower side. We also set $t_0 = s$.

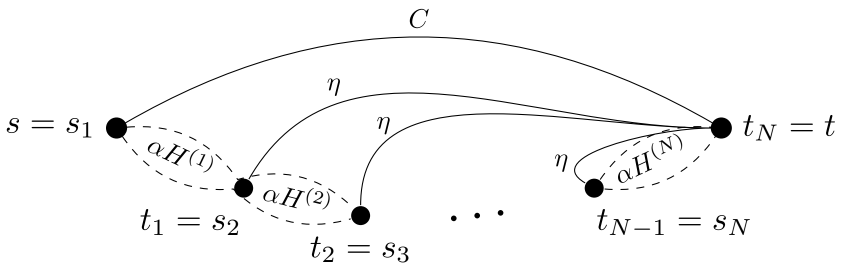


Figure 8 The graph $\mathcal{T}(H, \alpha, C, \eta, N)$ with its outerplanar embedding.

For the remainder, we define a trivial arithmetic sequence generating all positive half-integers: for any integer $i \ge 0$, let $C_i = \frac{1}{2} + i$. For any value C_i, we are able to construct a collection of road maps for which ratio C_i cannot be achieved by any deterministic strategy.

▶ **Proposition 4.2 (*).** *For any nonnegative integer i, there exists a family $\mathcal{R}_i$ of road maps which satisfies the following properties:*

- *all the road maps of $\mathcal{R}_i$ are defined on the same weighted (s,t)-unbalanced outerplanar graph,*
- *no deterministic strategy can achieve ratio C_i on family $\mathcal{R}_i$.*

Sketch of the proof. By induction, we assume that the property holds for some $i \ge 1$. We focus on some graph $H_{i+1} = \mathcal{T}(H_i, \alpha, C_i, \eta, N)$. The reasoning consists in a trade-off on the optimal distance between s and the last visited t_j vertex on the lower side, denoted by t_q. If $d_{E_*}(s, t_q)$ is at least C_i, then it appears that the st edge is the optimal offline path and we realize that exploring the lower side was too costly. We deduce that the obtained competitive ratio is necessarily greater than $C_i + 1$ using the induction hypothesis. Otherwise, if $d_{E_*}(s, t_q) < C_i$, then it means that we did not explore far enough on the lower side, hence the optimal offline path passes through it. We obtain the same conclusion that the ratio is greater than $C_i + 1$. See full version [6] for the proof.

Hence, no deterministic strategy can achieve constant competitive ratio on weighted outerplanar graphs since integer i can take arbitrarily large values:

▶ **Theorem 1.3.** *There is no constant C, independent from G and k, such that a deterministic strategy achieves competitive ratio C on all road maps (G, E_*) where G is a weighted outerplanar graph.*

Proof. By contradiction, for any $C \geq 1$, apply Proposition 4.2 on $i = \lceil C \rceil$. ◀

5 Perspectives

We highlighted a non-trivial unit-weighted family of graphs (outerplanar) for which there exists a deterministic strategy with constant competitive ratio 9, which is optimal. However, we proved that no constant competitive ratio can be achieved for arbitrarily weighted outerplanar graphs. Several questions arise.

Since some sub-families of outerplanar graphs have constant competitive ratio in the weighted case (trees and cycles, which imply cacti from Lemma 2.2) while a very close super-family admits the general bound $2k + 1$ in the unit-weighted case (planar of treewidth 2), a natural question is to investigate where the competitive gaps lie in both cases. For the unit-weighted case, future research could focus on the natural extension of p-outerplanar graphs [4], with p successive outer faces, in order to generalize constant competitiveness.

To achieve constant competitive ratio on arbitrarily weighted graphs, a good candidate could be graphs with bounded-sized minimal edge (s, t)-cuts, for which ratio $\sqrt{2}k + O(1)$ is known [12]. Observe that our construction $\mathcal{T}$ which disproves constant ratio increases the size of edge (s, t)-cuts. We conjecture that there exists a polynomial-time deterministic strategy achieving constant competitive ratio on graphs with edge (s, t)-cuts of bounded size.

References

1 V. Aksakalli, O. F. Sahin, and I. Ari. An AO* based exact algorithm for the canadian traveler problem. *INFORMS Journal on Computing*, 28(1):96–111, 2016. doi:10.1287/IJOC.2015.0668.

2 A. F. Alkaya, S. Yildirim, and V. Aksakalli. Heuristics for the Canadian Traveler Problem with neutralizations. *Comput. Ind. Eng.*, 159:107488, 2021. doi:10.1016/J.CIE.2021.107488.

3 R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993. doi:10.1006/INCO.1993.1054.

4 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

5 A. Bar-Noy and B. Schieber. The Canadian Traveller Problem. In *Proc. of ACM/SIAM SODA*, pages 261–270, 1991.

6 L. Beaudou, P. Bergé, V. Chernyshev, A. Dailly, Y. Gerard, A. Lagoutte, V. Limouzy, and L. Pastor. The Canadian Traveller Problem on outerplanar graphs. *CoRR*, abs/2403.01872, 2024. doi:10.48550/arXiv.2403.01872.

7 A. Beck and D. J. Newman. Yet more on the linear search problem. *Isr. J. Math.*, 8(4):419–429, 1970.

8 J. Becker and R. Batta. Canadian prize collection problem. *Military Operations Research*, 28(2):55–92, 2023.

9 R. Bellman. Problem 63-9, an optimal search. *SIAM review*, 5(3):274–274, 1963.

10 M. Bender and S. Westphal. An optimal randomized online algorithm for the k-Canadian Traveller Problem on node-disjoint paths. *J. Comb. Optim.*, 30(1):87–96, 2015. doi:10.1007/S10878-013-9634-8.

11 P. Bergé, J. Desmarchelier, W. Guo, A. Lefebvre, A. Rimmel, and J. Tomasik. Multiple canadians on the road: minimizing the distance competitive ratio. *J. Comb. Optim.*, 38(4):1086–1100, 2019. doi:10.1007/S10878-019-00438-6.

12 P. Bergé and L. Salaün. The influence of maximum $(s,\ t)$-cuts on the competitiveness of deterministic strategies for the Canadian Traveller Problem. *Theor. Comput. Sci.*, 941:221–240, 2023. doi:10.1016/J.TCS.2022.11.017.

13 Z. Bnaya, A. Felner, and S. E. Shimony. Canadian traveler problem with remote sensing. In *Proc. of IJCAI*, pages 437–442, 2009.

14 A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge Univ. Press, 1998.

15 H. Chan, J. Chang, H. Wu, and T. Wu. The k-Canadian Traveller Problem on Equal-Weight Graphs. *Proc. of WCMCT*, pages 135–137, 2015.

16 E. D. Demaine, Y. Huang, C. Liao, and K. Sadakane. Canadians Should Travel Randomly. *Proc. of ICALP*, pages 380–391, 2014. doi:10.1007/978-3-662-43948-7_32.

17 R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

18 P. Eyerich, T. Keller, and M. Helmert. High-quality policies for the Canadian Traveler's Problem. In *Proces. of AAAI*, pages 51–58. AAAI Press, 2010. doi:10.1609/AAAI.V24I1.7542.

19 D. Fried, S. E. Shimony, A. Benbassat, and C. Wenner. Complexity of canadian traveler problem variants. *Theor. Comput. Sci.*, 487:1–16, 2013. doi:10.1016/J.TCS.2013.03.016.

20 N. Hahn and M. Xefteris. The covering canadian traveller problem revisited. In *Procs. of MFCS*, volume 272 of *LIPIcs*, pages 53:1–53:12, 2023. doi:10.4230/LIPICS.MFCS.2023.53.

21 M.-Y. Kao, J. H. Reif, and S. T. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and computation*, 131(1):63–79, 1996. doi:10.1006/INCO.1996.0092.

22 C. Liao and Y. Huang. The covering canadian traveller problem. *Theor. Comput. Sci.*, 530:80–88, 2014. doi:10.1016/J.TCS.2014.02.026.

23 L. V. Lita, J. Schulte, and S. Thrun. A system for multi-agent coordination in uncertain environments. In *Procs. of AGENTS*, pages 21–22. ACM, 2001. doi:10.1145/375735.375806.

24 C. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991. doi:10.1016/0304-3975(91)90263-2.

25 D. Shiri and F. S. Salman. On the online multi-agent $O-D$ k-Canadian Traveler Problem. *J. Comb. Optim.*, 34(2):453–461, 2017. doi:10.1007/S10878-016-0079-8.

26 S. Westphal. A note on the k-Canadian Traveller Problem. *Inform. Proces. Lett.*, 106(3):87–89, 2008. doi:10.1016/J.IPL.2007.10.004.

27 Y. Xu, M. Hu, B. Su, B. Zhu, and Z. Zhu. The Canadian traveller problem and its competitive analysis. *J. Comb. Optim.*, 18(2):195–205, 2009. doi:10.1007/S10878-008-9156-Y.

Simple Qudit ZX and ZH Calculi, via Integrals

Niel de Beaudrap
University of Sussex, Brighton, UK

Richard D. P. East
Haiqu

Abstract

The ZX calculus and ZH calculus use diagrams to denote and compute properties of quantum operations, using "rewrite rules" to transform between diagrams which denote the same operator through a functorial *semantic map*. Different semantic maps give rise to different rewrite systems, which may prove more convenient for different purposes. Using discrete measures, we describe semantic maps for ZX and ZH diagrams, well-suited to analyse unitary circuits and measurements on qudits of any fixed dimension $D > 1$ as a single "ZXH-calculus". We demonstrate rewrite rules for the "stabiliser fragment" of the ZX calculus and a "multicharacter fragment" of the ZH calculus.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases ZX-calculus, ZH-calculus, qudits, string diagrams, discrete integrals

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.20

Related Version *Full Version*: https://arxiv.org/abs/2304.03310v2 [15]

Acknowledgements Thanks to Patrick Roy, Titouan Carette, John van de Wetering, and Robert Booth for technical discussions, and to the anonymous referees for suggestions on presentation.

1 Introduction

The ZX calculus [10, 2, 28, 42] and ZH calculus [3, 40] are systems using annotated graphs ("ZX diagrams" and "ZH diagrams"), to denote tensor networks for quantum computation, and other problems involving tensors over $\mathbb{C}^2$ [16, 19, 39, 18, 17, 26]. They include rewrite rules, to perform computations on diagrams without recourse to exponentially large matrices. Complicated procedures may involve diagrams of mounting complexity to analyse, but the ZX- and ZH-calculi often simplify the analysis of many-qubit procedures. It is also increasingly common to consider versions of the ZX- and ZH-calculi for qudits [20, 44, 48, 6, 41, 27, 33, 32], which promise similar benefits for the analysis of procedures on qudits.

Most treatments of these calculi [2, 3, 4, 5, 6, 11, 14, 21, 22, 23, 24, 25, 28, 29, 30, 33, 37, 38, 40, 42, 43, 45, 46, 47, 48, 49] are "scalar exact": equational theories, that do not introduce changes by scalar factors. Changes by scalar factors do not matter for some applications (e.g., testing equivalence of unitary transformations), but *are* important for probabilistic processes (e.g., postselection) or to compute specific numerical values [26]. But "scalar exact" treatments may involve frequent accumulation or deletion of *scalar gadgets*: disconnected sub-diagrams which obliquely denote normalisation factors. Presentations of these calculi which avoid such book-keeping, are simpler for instruction and practical use, and may also admit a unified rewrite system (a "ZXH calculus") incorporating the rules of each [14, 19, 18].

In previous work [14], one of us addressed this issue of bookkeeping of scalars for ZX- and ZH-diagrams on qubits through a carefully constructed semantic map. The result, described as "well-tempered" versions of these calculi, are scalar exact while avoiding the modifications of scalar gadgets for the most often-used rewrites. However, while the rewrite rules of this "well-tempered" notation are simple, the notational convention itself (i.e., the semantics of the generators of the calculi) is slightly unwieldly. Furthermore, it left open how to address similar issues with scalars for versions of these calculi on qudits of dimension $D > 2$.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 20; pp. 20:1–20:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this work, we consider how different normalisations of the ZX- and ZH-calculus may be expressed in a more uniform way, by representing operators on qudits (of any fixed dimension $D > 1$) through the use of integrals with respect to a discrete measure.

For a finite set S, let $\#S$ denote its cardinality. Consider a measure $\mu(S) = \#S \cdot \nu^2$ on subsets $S \subseteq \mathbb{Z}$, for $\nu > 0$ to be fixed later. This is a "measure" on sets S (see Section 2.1) which allows us to define a formal notion of integration of functions $f : \mathbb{Z} \to \mathbb{C}$,

$$\int\limits_{x \in S} f(x) \;:=\; \int\limits_{x \in S} f(x)\, \mathrm{d}\mu(x) \;:=\; \sum_{x \in S} f(x)\, \nu^2 \,. \tag{1}$$

(For the sake of brevity, we often use the standard convention of suppressing the differential $\mathrm{d}\mu$, as on left-hand expression below, when the measure of integration is understood.) Such integrals allow us to express sums with certain normalising factors more uniformly, by absorbing the factors into the measure μ by an appropriate choice of $\nu > 0$. For a finite-dimensional Hilbert space $\mathcal{H}$ with standard basis $|x\rangle := \mathbf{e}_x$ for some index set $x \in \mathbf{D}$, we may define the (not-necessarily normalised) *point-mass distributions* $|x\rangle\!\rangle = \nu^{-1} |x\rangle \in \mathcal{H}$, and their adjoints $\langle\!\langle x| = |x\rangle\!\rangle^\dagger$. Then, if we have some "state-function" $|f\rangle\!\rangle := \int_{x \in \mathbf{D}} f(x)\, |x\rangle\!\rangle$ for an arbitrary function $f : \mathbb{Z} \to \mathbb{C}$, it is easy to show that

$$\langle\!\langle z|f\rangle\!\rangle \;:=\; \int\limits_{x \in \mathbf{D}} \langle\!\langle z|x\rangle\!\rangle\, f(x) \;=\; f(z), \tag{2}$$

similar to how Dirac measures are used with integration over $\mathbb{R}$. While a similar result $\langle z|f\rangle = f(z)$ holds if we simply define $|f\rangle = \sum_{x \in D} f(x)\, |x\rangle$, couching this sort of analysis in terms of discrete integrals and point-mass functions $|x\rangle\!\rangle$ allows us to accommodate scalar factors which may arise when manipulating expressions involving operators such as $\sum_{x \in \mathbf{D}} |x\rangle^{\otimes n} \langle x|^{\otimes m}$ for $m, n > 1$. This is an example of the sort of operator, for which book-keeping of scalar factors frequently arises in most versions of the ZX- or ZH-calculi.

By introducing the additional layer of abstraction, provided by discrete integrals and their accompanying point-mass functions $|x\rangle\!\rangle$, we describe semantics for ZX- and ZH-diagrams which are simple, and which admits a system of rewrites which largely dispenses with the need for modifications to scalar gadgets in the diagrams. This approach to notation, and the rewrites which we demonstrate, are applicable for generators representing operators on qudits of *any* finite dimension, and enables the two calculi to be used interoperably as a single "ZXH-calculus". We present this approach in the hopes that it facilitates the development of practically useful extensions of these calculi beyond qubits.

Structure of this article. Section 2 sets out number-theoretic preliminaries, some background in string diagrams, and common approaches to defining ZX and ZH calculi. Section 3 introduces discrete measures and integrals on $\mathbf{D}$, including what little measure theory we require, and considers the constraints that follow from a particular treatment of discrete Fourier transforms. Section 4 demonstrates how using such discrete integrals as the basis for a semantic map for ZX and ZH diagrams, leads to convenient representations of particular unitary operators and convenient rewrites for both the ZX and ZH calculi. We frequently refer to the Appendices of the full version [15] of this work, where we provide complete proofs, more details about our constructions, and connections to related subjects.

Related work. As we note above, there is recent and ongoing work [20, 44, 48, 6, 41, 27, 33, 32, 34] on ZX, ZH, and related calculi on qudits of dimension $D > 2$ (though often restricted to the case of D an odd prime). Our work is influenced in particular by Booth and Carette [6], and Roy [33], and we are aware of parallel work by collaborations involving these authors [34, 31]. Our work is distinguished in presenting convenient semantics for both ZX and ZH diagrams for arbitrary $D > 1$, notably including the case where D is composite or even.

2 Preliminaries

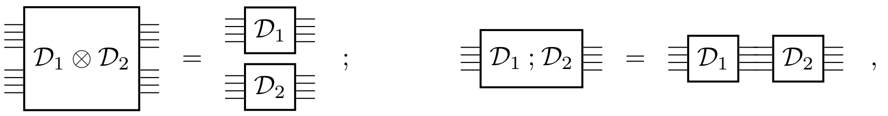
2.1 Number-theoretic preliminaries

Let $D > 1$ be a fixed integer, and $\omega = \mathrm{e}^{2\pi i/D}$. We assume basic familiarity with number theory, in particular with $\mathbb{Z}_D$, the integers modulo D. While it is common to associate $\mathbb{Z}_D$ with the set $\{0, 1, \ldots, D-1\}$ of non-negative "residues" of integers modulo D, one may associate them $\mathbb{Z}_D$ with any contiguous set of residues $\mathbf{D} = \{L, L+1, \ldots, U-1, U\}$ where $U - L + 1 = D$.[1] We may then occasionally substitute $\mathbb{Z}_D$ for $\mathbf{D}$ when this is unlikely to cause confusion: this will most often occur in the context of expressions such as ω^{xy}, which is well-defined modulo D in each of the variables x and y (i.e., adding any multiple of D to either x or y does not change the value of the expression). In such an expression, while we may intend for one of x or y or both may be an element of $\mathbb{Z}_D$ in principle, they would in practise be interpreted as a representative integer $x, y \in \mathbf{D} \subseteq \mathbb{Z}$.

2.2 String diagrams

ZX- and ZH-diagrams are examples of *string diagrams*, which can be described as diagrams composed of dots (or boxes) and wires, where the wires denote objects and the dots/boxes denote maps on those objects.

In the string diagrams which we consider in this article, diagrams are composed of dots or boxes, and wires. These diagrams can be described as being a composition of "generators", which typically consist of one (or zero) dots/boxes with some amount of meta-data, and any number (zero or more) directed wires, where the direction is usually represented by an orientation in the diagram. (In this article, wires are oriented left-to-right, though they are also allowed to bend upwards or downwards.) For any two diagrams $\mathcal{D}_1$ and $\mathcal{D}_2$, we may define composite diagrams $\mathcal{D}_1 \otimes \mathcal{D}_2$ and $\mathcal{D}_1 \,;\, \mathcal{D}_2$, represented schematically by


$\mathcal{D}_1 \otimes \mathcal{D}_2$
=
$\mathcal{D}_1$
$\mathcal{D}_2$
;
$\mathcal{D}_1 \,;\, \mathcal{D}_2$
=
$\mathcal{D}_1$
$\mathcal{D}_2$
,

(3)

which we call the "parallel" and "serial" composition of $\mathcal{D}_1$ and $\mathcal{D}_2$. In the latter case we require that the number of output wires of $\mathcal{D}_1$ (on the right of $\mathcal{D}_1$) equal the number of input wires of $\mathcal{D}_2$ (on the left of $\mathcal{D}_2$), for the composition to be well-defined.

String diagrams may be used to denote maps in a monoidal category $\mathbf{C}$ (in which objects can be aggregated to form composite objects through a parallel product, which we denote by "$\otimes$"). This is done through a *semantic map* $\llbracket \cdot \rrbracket$ which maps each generator to a map in $\mathbf{C}$. This semantic map is defined to be consistent with respect to composition, in the sense that

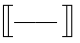
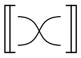
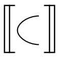
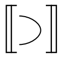
$$\Big\llbracket \mathcal{D}_1 \otimes \mathcal{D}_2 \Big\rrbracket = \llbracket \mathcal{D}_1 \rrbracket \otimes \llbracket \mathcal{D}_2 \rrbracket, \qquad \Big\llbracket \mathcal{D}_1 \,;\, \mathcal{D}_2 \Big\rrbracket = \llbracket \mathcal{D}_2 \rrbracket \circ \llbracket \mathcal{D}_1 \rrbracket. \tag{4}$$

[1] The reader may wonder why we do not simply adopt the conventional choice of $\mathbf{D} = \{0, 1, \ldots, D-1\}$. There are multiple reasons, the simplest of which being that allowing for the index to include negative integers may be useful for representing certain "quantum numbers" in application to physics.

Note the reversal of the order for sequential composition, which is just an artefact of the difference in orientation of diagrams (left-to-right), and the conventional right-to-left application order of functions that is common e.g. in quantum information theory.

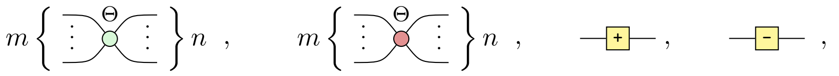
2.3 Preliminary remarks on ZX and ZH diagrams

ZX and ZH diagrams are string diagrams which denote multi-linear operators on some finite-dimensional vector space $\mathcal{H} \cong \mathbb{C}^{\mathbf{D}}$, equipped with a standard basis $|x\rangle$ for $x \in \mathbf{D}$ and functionals $\langle x| = |x\rangle^\dagger$. (For $\mathbf{D}$ a set of D consecutive integers, we may use arithmetic expressions, such as $|x+y\rangle$, to index basis vectors; specifically in the labels of "kets" and "bras", such expressions can be understood to be evaluated mod D.) The parallel product in this case is the usual tensor product $\otimes$, and the sequential product is composition of operators. For a generator $\mathcal{D}$ with m input wires and n output wires, one assigns an operator $[\![\mathcal{D}]\!] : \mathcal{H}^{\otimes m} \to \mathcal{H}^{\otimes n}$. To represent string diagrams to represent maps in which some of the "parallel" operands are being permuted or unaffected, we also consider generators consisting only of wires. We consider four such generators, to which we assign semantics as follows:

 $= \sum_{x\in\mathbf{D}} |x\rangle\langle x|\,,$  $= \sum_{x,y\in\mathbf{D}} |y,x\rangle\langle x,y|\,,$  $= \sum_{x\in\mathbf{D}} |x,x\rangle\,,$  $= \sum_{x\in\mathbf{D}} \langle x,x|\,.$ (5)

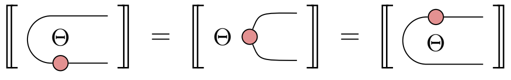
ZX and ZH diagrams are designed with different priorities, but have common features. ZX diagrams are effective for representing operations generated by single-qubit rotations and controlled-NOT gates; in most cases (excepting, e.g., Refs. [46, 6]) it rests on the unitary equivalence of two conjugate bases. ZH diagrams were developed to facilitate reasoning about quantum circuits over the Hadamard-Toffoli gate set [36, 1]. Both were originally defined so that the semantics is preserved by a change in the presentation of the underlying graph, which preserves the connectivity of the diagram [10, 12].

ZX Diagrams. We define the following ZX generators on qudits with state-space $\mathcal{H}$,



(6)

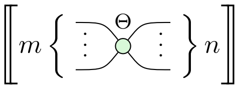
where $m, n \in \mathbb{N}$, and for any function $\Theta : \mathbb{Z} \to \mathbb{C}$. (Our approach of using functions mildly extends the approach of Wang [48], who prefers to parameterise the generators with vectors indexed from 1. For the constant function $\Theta(x) = 1$, we may omit the label Θ entirely.) We call these generators "green dots", "red dots", "Hadamard plus boxes", and "Hadamard minus boxes". The usual approach to assigning semantics to ZX generators is by considering the green and red dots to represent similar operations, subject to different (conjugate) choices of orthonormal basis, and a unitary "Hadamard" gate relating the two bases. One defines a semantic map $[\![\,\cdot\,]\!]$ in which the (lighter-coloured) "green" dots are mapped to an action on the basis $|x\rangle$, and the (darker-coloured) "red" are mapped to an action on the basis $|\omega^x\rangle$, where $|\omega^k\rangle = \frac{1}{\sqrt{D}} \sum_x \omega^{-kx} |x\rangle$ for $k \in \mathbf{D}$ (and where again $\omega = \mathrm{e}^{2\pi i/D}$).[2]The conventional choice would be, for a green dot, to assign an interpretation such as $\sum_{x\in\mathbf{D}} \Theta(x)\,|x\rangle^{\otimes n}\langle x|^{\otimes m}$; and for a red dot, to assign the interpretation $\sum_{x\in\mathbf{D}} \Theta(x)\,|\omega^x\rangle^{\otimes n}\langle\omega^x|^{\otimes m}$. Unfortunately, for $D > 2$, such a conventional interpretation does not yield a "flexsymmetric" [7] calculus, in effect because $\langle\omega^a|^{\mathsf{T}} = |\omega^a\rangle^* = |\omega^{-a}\rangle$. In particular, the conventional approach described just above would mean that the equality



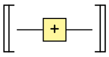
(7)

[2] In the notation of Ref. [6], we have $|\omega^k\rangle = |k{:}X\rangle$, up to a relabeling of the basis elements of $\mathcal{H}$.

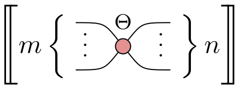
would not hold: the first diagram would denote $\sum_x \Theta(x)\,|\omega^{-x},\omega^{x}\rangle$, the second would denote $\sum_x \Theta(x)\,|\omega^{x},\omega^{x}\rangle$, and the third would denote $\sum_x \Theta(x)\,|\omega^{x},\omega^{-x}\rangle$. This represents a way in which such a calculus would fail to have the useful syntactic property that "only the connectivity matters" [10, 12]; and other inconveniences would also arise, which would make these diagrams more difficult to work with. To avoid this problem, we endorse the convention adopted by Refs. [6, 41] of involving a generator which is related to the green dot by *different* unitary transformations on the inputs and outputs, but which differ only by a permutation. We then interpret the generators of Eqn. (6) as operators using a model $[\![\,\cdot\,]\!]$ which satisfies



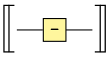
$$\propto \sum_{x\in\mathbf{D}} \Theta(x)\;|x\rangle^{\otimes n}\langle x|^{\otimes m}\;,$$



$$\propto \sum_{x,k\in\mathbf{D}}\sum \omega^{kx}\,|x\rangle\langle k| \qquad (8)$$



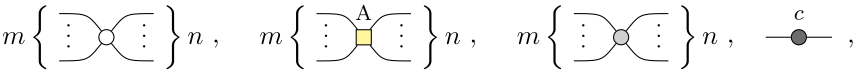
$$\propto \sum_{k\in\mathbf{D}} \Theta(k)\;|\omega^{-k}\rangle^{\otimes n}\langle \omega^{k}|^{\otimes m}\;,$$



$$\propto \sum_{x,k\in\mathbf{D}}\sum \omega^{-kx}\,|x\rangle\langle k|$$

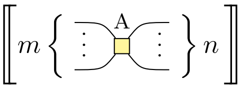
so that the "Hadamard" plus and minus boxes are proportional to the quantum Fourier transform $|\omega^{k}\rangle \mapsto |k\rangle$ (i.e., the inverse discrete Fourier transform), and its adjoint.

ZH Diagrams. We define the following ZH generators on qudits with Hilbert space $\mathcal{H}$,

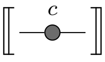


$$(9)$$

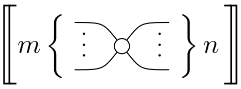
where $m,n\in\mathbb{N}$, $c\in\mathbb{Z}$, and for any function $\mathrm{A}:\mathbb{Z}\to\mathbb{C}$. (If $\mathrm{A}(t)=\alpha^{t}$ for some $\alpha\in\mathbb{C}^{\times}$, we may write the scalar α in place of A, consistent with the notation for ZH generators in Refs. [3, 14]. Following Roy [33], we later define a further short-hand notation for $\mathrm{A}(t)=\chi_c(t)=\exp(2\pi i c t/D)$ with $c\in\mathbb{Z}$.) We call these generators "white dots", "H-boxes", "gray dots", and "generalised-not dots".[3] We interpret the generators of Eqn. (9) as operators using a model $[\![\,\cdot\,]\!]$ which satisfies the following:



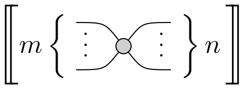
$$\propto \sum_{x\in\mathbf{D}^m,\,y\in\mathbf{D}^n}\sum \mathrm{A}(x_1\cdots x_m y_1\cdots y_n)\;|y\rangle\!\langle x|$$



$$\propto \sum_{x\in\mathbf{D}} |{-c}{-x}\rangle\langle x| \qquad (10)$$

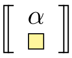


$$\propto \sum_{x\in\mathbf{D}} |x\rangle^{\otimes n}\langle x|^{\otimes m}$$



$$\propto \sum_{\substack{x\in\mathbf{D}^m,\,y\in\mathbf{D}^n\\ \sum_h x_h+\sum_k y_k\equiv 0}}\sum |y\rangle\!\langle x|\,,$$

where for the gray dots, we constrain the indices $x\in\mathbf{D}^m$ and $y\in\mathbf{D}^n$, so that the sum of their entries is 0 mod D; and for the not-dots we interpret the index of the vector $|-c-x\rangle$ modulo D. By contrast, note that for the H-boxes, we consider the products of the input and output labels $x_1,\ldots,x_m,y_1,\ldots,y_m$ as *integers*,[4] i.e., elements of $\mathbb{Z}$, whose product is the argument of A in the the expression $\mathrm{A}(x_1\cdots y_m)$. In particular, we fix the semantics so that



$$= \sum_{\text{(singleton)}} \alpha^{\text{(empty product)}}\cdot 1 \;=\; \alpha^{1} \;=\; \alpha, \qquad (11)$$

again using the short-hand that $\alpha\in\mathbb{C}^{\times}$ stands for the function $\mathrm{A}(t)=\alpha^{t}$.

[3] We follow Ref. [14] in considering the gray and not dots to be (primitive) generators, rather than gadgets or "derived generators", e.g., as in Refs. [3, 33].

[4] Note that our use of the index-set $x\in\mathbf{D}=\{L,L+1,\ldots,U-1,U\}$ means that the exponential function $t\mapsto\alpha^{t}$ for $\alpha=0$ is not well-defined if $L<0$. We may instead consider a function $\mathbf{X}_{\{0\}}:\mathbb{Z}\to\mathbb{C}$ given by $\mathbf{X}_{\{0\}}(t)=1$ for $t=0$, with $\mathbf{X}_{\{0\}}(t)=0$ otherwise: this is substitution is adequate, e.g., for applications to counting complexity [16, 26].

Remarks on semantics, and rewriting systems. Eqns. (8) and (10) describe not one semantic map $[\![\,\cdot\,]\!]$ for ZX diagrams or ZH diagrams, but rather the conventional approach (with minor elaborations) to choosing such semantic maps. A specific semantic map determines which pairs of diagrams have the same semantics, and therefore which *diagrammatic rewrites* are *sound* (i.e., which local transformations one may perform to a diagram without changing its semantics). We suggest that rewrite systems, in which the most commonly used diagrammatic rewrites can be expressed simply, are to be preferred over others. However, this depends on obtaining a semantic map $[\![\,\cdot\,]\!]$ for which such a rewrite system is sound.

- Some authors (e.g., [10, 12]) prefer to define semantics only up to proportionality, in which case Eqns. (8) and (10) suffice to determine when two diagrams are equivalent up to an neglected scalar factor. This has the virtue of simplicity, but does not provide the precision needed for all applications one might wish to consider for these calculi.
- Most "scalar exact" treatments of ZX and ZH fix a map $[\![\,\cdot\,]\!]$ by replacing the proportionalities in Eqns. (8) and (10) with equalities – except for the "Hadamard" boxes of Eqn. (8), where a factor of $1/\sqrt{D}$ is used to yield unitary operators. However, the rewrites in those systems often involve book-keeping of auxiliary sub-diagrams ("scalar gadgets").
- In the case $D = 2$, Ref. [14] presents a different, unified semantic map $[\![\,\cdot\,]\!]_\nu$ for both ZX and ZH diagrams, in order to support rewrites involving fewer scalar gadgets. However, the scalar factors involved in those semantics could be considered non-obvious, and does not provide insights into how one would achieve the same goal for arbitrary $D \geqslant 2$.

The aim of this work is to extend the results of Ref. [14], providing a simple approach to fixing a semantic map $[\![\,\cdot\,]\!]$ for both ZX and ZH diagrams for arbitrary $D \geqslant 2$, which supports a set of diagrammatic rewrites without much use of scalar gadgets.

3 Discrete integrals

Our main theoretical contribution is to demonstrate how discrete integrals provide a way to fix a semantic map for ZX and ZH diagrams, with favourable properties. In this section, we introduce discrete measures and discrete integrals independently of string diagrams, and consider the constraints on discrete measures obtained through a particular representation of discrete Fourier transforms.

3.1 Introducing discrete measures and discrete integrals

We begin by introducing more fully the concepts first described on page 2. For a set $\mathbf{X}$, let $\wp(\mathbf{X})$ be the power-set of $\mathbf{X}$. We may define a *σ-algebra on* $\mathbf{X}$ to be a set $\Sigma \subseteq \wp(\mathbf{X})$ which contains $\mathbf{X}$, which is closed under set complements ($S \in \Sigma \iff \mathbf{X} \setminus S \in \Sigma$), and which is closed under countable unions (if $S_1, S_2, \ldots \in \Sigma$, then $S_1 \cup S_2 \cup \cdots \in \Sigma$). – The purpose of defining Σ is to allow the notion of a *measure* $\mu : \Sigma \to \mathbb{R} \cup \{+\infty\}$ to be defined, where the sets $S \in \Sigma$ are the ones which have a well-defined measure. Such a function μ is a measure, if and only if $\mu(\varnothing) = 0$, $\mu(S) \geqslant 0$ for all $S \in \Sigma$, and if

$$\mu\bigl(S_1 \cup S_2 \cup \cdots\bigr) = \mu(S_1) + \mu(S_2) + \cdots \tag{12}$$

for any sequence of disjoint sets $S_j \in \Sigma$. An example is the σ-algebra Σ consisting of all countable unions of intervals over $\mathbb{R}$, with μ defined by assigning $\mu(J) = b - a$ to any interval $J = (a,b)$, $J = (a,b]$, $J = [a,b)$, or $J = [a,b]$ for $a \leqslant b$. A somewhat more exotic measure is the Dirac distribution μ_δ on $\mathbb{R}$, for which $\mu_\delta(S) = 1$ if $0 \in S$, and $\mu_\delta(S) = 0$ otherwise. (We remark on the Dirac distribution and related concepts in Appendix [15, Appendix D].) However, we are mainly interested in measures μ defined on subsets $S \subseteq \mathbf{D}$, for which $\mu(S) \propto \#S$.

For the set $\mathbf{D} = \{L, L+1, \ldots, U-1, U\}$, consider the σ-algebra $\mathcal{B} = \wp(\mathbf{D})$ consisting of all subsets of $\mathbf{D}$. Define the measure $\mu : \mathcal{B} \to \mathbb{R}$ on this σ-algebra given by $\mu(S) \;=\; \#S \cdot \nu^2$, where $\nu > 0$ can in principle be chosen freely. This presents $\mathbf{D}$ as a measure space, the purpose of which is to allow us to define (multi-)linear operators on $\mathcal{H}$ as arising from integrals with respect to that measure. For a function $f : \mathbb{Z} \to \mathbb{C}$, we may define a notion of integration of f over a subset $S \subseteq \mathbf{D}$:

$$\int\limits_{x \in S} f(x)\, \mathrm{d}\mu(x) \;=\; \sum_{x \in S} f(x)\, \mu(\{x\}) \;=\; \sum_{x \in S} f(x)\, \nu^2 \,. \tag{13}$$

We may apply this notion of integration to operator-valued functions, as is typical for wave-functions in quantum mechanics. For instance, one may define

$$\int\limits_{x \in S} f(x)\; |x\rangle\; \mathrm{d}\mu(x) \;=\; \nu^2 \sum_{x \in S} f(x)\; |x\rangle \,. \tag{14}$$

In the usual approach to describing wave-functions over $\mathbb{R}$, one takes $|x\rangle$ to represent a point-mass distribution (i.e., not a vector $\boldsymbol{v} \in \mathbb{C}^{\mathbb{R}}$ for which $v_x = 1$), so that the equality

$$\langle z| \left[\; \int\limits_{x \in \mathbb{R}} f(x)\; |x\rangle\; \mathrm{d}x \right] = \int\limits_{x \in \mathbb{R}} f(x)\, \delta_z(x)\, \mathrm{d}x \;=\; f(z), \tag{15}$$

holds. Here $\delta_z(x)$ is a shifted Dirac distribution (see Ref. [15, Appendix D.1] for more details).[5] To avoid notational confusion, we prefer to reserve the symbol "$|x\rangle$" to represent a unit-norm standard basis vector in $\mathcal{H}$ (i.e., a vector $\boldsymbol{v} \in \mathcal{H}$ such that $v_x = 1$), and introduce a symbol "$|x\rangle\!\rangle$" which denotes the vector $|x\rangle\!\rangle \;=\; \frac{1}{\nu}\, |x\rangle$, specifically so that

$$\langle\!\langle z| \left[\; \int\limits_{x \in \mathbf{D}} f(x)\; |x\rangle\!\rangle\; \mathrm{d}\mu(x) \right] = \int\limits_{x \in \mathbf{D}} f(x)\; \langle\!\langle z|x\rangle\!\rangle\; \mathrm{d}\mu(x) \;=\; \nu^2 \sum_{x \in \mathbf{D}} f(x) \frac{\langle z|x\rangle}{\nu^2} \;=\; f(z)\,, \tag{16}$$

and also

$$\int\limits_{x \in \mathbf{D}} |x\rangle\!\rangle\langle\!\langle x|\; \mathrm{d}\mu(x) \;=\; \nu^2 \sum_{x \in \mathbf{D}} \frac{|x\rangle\,\langle x|}{\nu^2} \;=\; \sum_{x \in \mathbf{D}} |x\rangle\,\langle x| \;=\; \mathbf{1}\,. \tag{17}$$

The notation "$|x\rangle\!\rangle$" provides us the flexibility to consider which measures $\mu : \mathcal{B} \to \mathbb{R}$ are best suited for defining convenient semantics for ZX and ZH generators, while retaining the features provided by Dirac distributions over $\mathbb{R}$, and without constraining ν.

For maps U and V described in this way, one may analyse compositions UV in the same way that one would do if U and V were given by sums of operators: by using the expressions for U and V in terms of discrete integrals, manipulating expressions within the integrals, and well-judged use of algebraic identities such as Eqns. (16) and (17). For instance, if we have

$$U \;=\; \iint\limits_{x,y \in \mathbf{D}} u_{x,y}\; |x\rangle\!\rangle\langle\!\langle y| \qquad\qquad V \;=\; \iint\limits_{w,z \in \mathbf{D}} v_{w,z}\; |w\rangle\!\rangle\langle\!\langle z| \tag{18}$$

then we may express the composite operation UV by

[5] While it is not necessary to understand our results, readers who are interested in connections between the integrals and measures presented here with integration over compact groups, may be interested in remarks which we make in Ref. [15, Appendix D.3].

$$\begin{aligned} UV \;&=\; \left[\;\iint\limits_{x,y\in\mathbf{D}} u_{x,y}\;|x\rangle\!\rangle\langle\!\langle y|\right]\left[\;\iint\limits_{w,z\in\mathbf{D}} v_{w,z}\;|w\rangle\!\rangle\langle\!\langle z|\right] \\ &=\; \iiiint\limits_{w,x,y,z\in\mathbf{D}} u_{x,y}v_{w,z}\;|x\rangle\!\rangle\langle\!\langle y|w\rangle\!\rangle\langle\!\langle z| \;=\; \iint\limits_{x,z\in\mathbf{D}}\left(\;\iint\limits_{w,y\in\mathbf{D}} u_{x,y}v_{w,z}\;\langle\!\langle y|w\rangle\!\rangle\right)|x\rangle\!\rangle\langle\!\langle z| \\ &\qquad\qquad\qquad\qquad\qquad\qquad\;=\; \iint\limits_{x,z\in\mathbf{D}}\left(\;\int\limits_{y\in\mathbf{D}} u_{x,y}v_{y,z}\right)|x\rangle\!\rangle\langle\!\langle z|\;. \end{aligned} \tag{19}$$

Composition of such integrals generalises straightforwardly for tensors of any signature: examples can be seen in Appendix A.1 of Ref. [15] (and Appendix A.3 makes heavy use of such analysis of compositions to prove the soundness of various diagrammatic rewrites of ZX and ZH diagrams.) The only distinction between this approach and one expressed directly in terms of summation, are the scalar factors which are subsumed in the integral notation and operators such as $|x\rangle\!\rangle\langle\!\langle z|$, both of which are governed by the choice of measure μ.

3.2 Constraints on normalisation motivated by the Fourier transform

Having defined the discrete measure (and discrete integrals) over $\mathbf{D}$, and the corresponding point-mass distributions $|x\rangle\!\rangle$ to satisfy Eqn. (2), we may consider how this might influence our approach to analysis of complex-valued functions over $\mathbf{D}$ (or $\mathbb{Z}_D$, using a similar measure).

In analogy to a common representation[6] of the Fourier transform of functions on $\mathbb{R}$, we may describe the (discrete) Fourier transform of a function $f : \mathbb{Z}_D \to \mathbb{C}$ by

$$\hat{f}(k) \;=\; \int\limits_{x\in\mathbb{Z}_D} \mathrm{e}^{-2\pi i k x/D}\, f(x)\;\mathrm{d}\mu(x). \tag{20}$$

In principle, the domain $\mathbb{Z}_D$ of $\hat{f}$ indexes a character $\chi_k(x) = \mathrm{e}^{-2\pi ikx/D}$ in the dual group $\widehat{\mathbb{Z}_D}$. The dual group $\widehat{\mathbb{Z}_D}$ can itself be assigned a measure $\tilde{\mu}$ which is in principle *independent* of μ. As $\mathbb{Z}_D$ is a finite abelian group, we use the fact that there is an isomorphism $\varepsilon : \widehat{\mathbb{Z}_D} \to \mathbb{Z}_D$ to describe $\hat{f}$ as a function $\mathbb{Z}_D \to \mathbb{C}$. The isomorphism ε induces a measure $\mu' = \tilde{\mu} \circ \varepsilon^{-1}$ on $\mathbb{Z}_D$, which may differ from μ and which would be relevant to any integrals involving the argument of $\hat{f}$.[7] – Note that there are different conventions for normalising the Fourier transform (over $\mathbb{R}$ or $\mathbb{Z}_D$): one might consider modifying Eqn. (20) to include a non-trivial scalar factor on the right-hand side. This is related to the questions of whether we take the Fourier transform $f \mapsto \hat{f}$ to preserve the ℓ_2-norm $\|f\|_2 = \Bigl(\int_x |f(x)|^2\,\mathrm{d}\mu(x)\Bigr)^{1/2}$, and whether we take μ' to differ from μ. We simply adopt the convention of defining the Fourier transform of $f : \mathbb{Z}_D \to \mathbb{C}$ as in Eqn. (20), and consider the constraints that this imposes on these other considerations.

[6] We emulate the presentation of the Fourier transform in terms of an oscillation frequency k (including the minus sign in the exponent, which for historical reasons is absent in the definition of the quantum Fourier transform). The main difference between Eqn. (20) and the usual Fourier transform over $\mathbb{R}$ is the factor of $1/D$ in the exponent: this can be shown to arise from a representation of functions $f : \mathbb{Z}_D \to \mathbb{C}$ in terms of discrete distributions on $\mathbb{R}$ (see Ref. [15, Appendix D.3.4]).

[7] The precise relationship between μ and μ', corresponds to the question in physics of the choice of units for x and k as continuous variables ranging over $\mathbb{R}$.

In analogy to standard practise in physics, we may use f to describe a "wave-function",[8]

$$|f\rangle\!\rangle \;:=\; \int\limits_{x\in\mathbb{Z}_D} f(x)\,|x\rangle\!\rangle\;\mathrm{d}\mu(x)\,. \tag{21}$$

A similar "wave function" for $\hat{f}$ would involve the measure μ', the measure on the argument of $\hat{f}$, which may in principle differ from μ:

$$|\hat{f}\,\rangle\!\rangle \;=\; \int\limits_{k\in\mathbb{Z}_D} \hat{f}(k)\,|k\rangle\!\rangle\;\mathrm{d}\mu'(k)\,, \tag{22}$$

integrating with respect to that different measure. Taking $\mu' \neq \mu$ would imply that the functions $f : (\mathbb{Z}_D, \mu) \to \mathbb{C}$, defined on $\mathbb{Z}_D$ considered as a space with measure μ, are strictly speaking not of the same type as their Fourier transforms $\hat{f} : (\mathbb{Z}_D, \mu') \to \mathbb{C}$ which are defined over a space with a different measure. String diagrams representing the transformations of such functions would then require wires of more than one type. While this is admissible in principle, we prefer to consider f and $\hat{f}$ to have the same measure space $(\mathbb{Z}_D, \mu)$ for their domains, so that we may treat them using string diagrams with wires of a single type, as we do in the ZX and ZH calculi. Identifying $\mu' = \mu$, we obtain

$$|\hat{f}\,\rangle\!\rangle \;=\; \int\limits_{k\in\mathbb{Z}_D} \hat{f}(k)\,|k\rangle\!\rangle\;\mathrm{d}\mu(k) \;=\; \iint\limits_{k,x\in\mathbb{Z}_D} \mathrm{e}^{-2\pi i k x/D} f(x)\;|k\rangle\!\rangle\;\mathrm{d}\mu(x)\;\mathrm{d}\mu(k)\,. \tag{23}$$

This motivates the definition of the discrete Fourier transform operator F over $\mathbb{Z}_D$, as

$$F \;=\; \iint\limits_{k,x\in\mathbb{Z}_D} \mathrm{e}^{-2\pi i k x/D}\;|k\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(x)\;\mathrm{d}\mu(k)\;, \tag{24}$$

so that $|\hat{f}\rangle\!\rangle = F|f\rangle\!\rangle$; this is the interpretation given to the "Hadamard minus box" in Eqn. (28). We adopt the convention that F is unitary, to allow it to directly represent a possible transformation of state-vectors over $\mathcal{H}$. This has the benefit that the inverse Fourier transform can be expressed similarly (now suppressing the differentials $\mathrm{d}\mu$, for brevity):

$$f(x) \;=\; \langle\!\langle x|F^\dagger|\hat{f}\rangle\!\rangle \;=\; \iiint\limits_{x,h,k\in\mathbb{Z}_D} \mathrm{e}^{2\pi i k x/D}\;|x\rangle\!\rangle\langle\!\langle k|\Bigl(\hat{f}(h)\,|h\rangle\!\rangle\Bigr) \;=\; \int\limits_{k\in\mathbb{Z}_D} \mathrm{e}^{2\pi i k x/D}\;\hat{f}(x)\;. \tag{25}$$

The definition of F in Eqn. (24) and the constraint that it should be unitary, imposes a constraint on the measure μ on $\mathbb{Z}_D$. We first prove a routine Lemma (which is used often in the Appendices of Ref. [15] in simplifying iterated integrals):

▶ **Lemma 1.** *Let $\omega = \mathrm{e}^{2\pi i/D}$ and $E \in \mathbf{D}$. Then* $\displaystyle\int\limits_{k\in\mathbb{Z}_D} \omega^{Ek}\;\mathrm{d}\mu(k) \;=\; \langle\!\langle E|0\rangle\!\rangle\,D\nu^4$.

Proof. This holds by reduction to the usual exponential sum:

$$\begin{aligned}\int\limits_{k\in\mathbf{D}} \mathrm{e}^{2\pi i E k/D}\;\mathrm{d}\mu(k) \;&=\; \nu^2 \sum_{k\in\mathbf{D}} \bigl(\omega^E\bigr)^k \;=\; \left\{\begin{array}{r} \nu^2\cdot\omega^{EL_D}\cdot\dfrac{(\omega^E)^D-1}{\omega-1}\,,\;\;\text{if } \omega^E\neq 1 \\ \nu^2\cdot D,\;\;\text{if } \omega^E = 1\end{array}\right\} \\ &=\; \delta_{E,0}\;D\nu^2 \;=\; \langle E|0\rangle\;D\nu^2 \;=\; \langle\!\langle E|0\rangle\!\rangle\;D\nu^4\;.\end{aligned}$$ ◀

[8] Note that $|f\rangle\!\rangle$ may not be a unit vector; whether this is the case depends on the values taken by f.

We may apply this in the case of the Fourier transform as follows. If F as expressed in Eqn. (24) is unitary, we have

$$\begin{aligned}
\mathbf{1} \;=\; F^\dagger F \;&=\; \Biggl[\;\iint\limits_{y,h\in\mathbf{D}} \mathrm{e}^{2\pi i h y/D}\; |y\rangle\!\rangle\langle\!\langle h|\;\mathrm{d}\mu(y)\;\mathrm{d}\mu(h)\Biggr]\Biggl[\;\iint\limits_{k,x\in\mathbf{D}} \mathrm{e}^{-2\pi i k x/D}\; |k\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(k)\;\mathrm{d}\mu(x)\Biggr]\\
&=\; \iiiint\limits_{y,h,k,x\in\mathbf{D}} \mathrm{e}^{2\pi i(hy-kx)/D}\; |y\rangle\!\rangle\langle\!\langle h|k\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(y)\;\mathrm{d}\mu(h)\;\mathrm{d}\mu(k)\;\mathrm{d}\mu(x)\\
&=\; \iiint\limits_{y,k,x\in\mathbf{D}} \mathrm{e}^{2\pi i k(y-x)/D}\; |y\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(y)\;\mathrm{d}\mu(k)\;\mathrm{d}\mu(x)\\
&=\; \iint\limits_{y,x\in\mathbf{D}} \Biggl[\;\int\limits_{k\in\mathbf{D}} \mathrm{e}^{2\pi i k(y-x)/D}\;\mathrm{d}\mu(k)\Biggr]\; |y\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(y)\;\mathrm{d}\mu(x)\\
&=\; \iint\limits_{y,x\in\mathbf{D}} \Bigl[D\nu^4\cdot\langle\!\langle y|x\rangle\!\rangle\Bigr]\; |y\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(y)\;\mathrm{d}\mu(x)\\
&=\; D\nu^4\int\limits_{x\in\mathbf{D}} |x\rangle\!\rangle\langle\!\langle x|\;\mathrm{d}\mu(x) \;=\; D\nu^4\cdot\mathbf{1}\,.
\end{aligned}\tag{26}$$

This implies that $\nu = D^{-1/4}$ (or equivalently, $N = \mu(\mathbb{Z}_D) = D\nu^2 = \sqrt{D}$).

As there are multiple conventions for representing the discrete Fourier transform, one might wish to consider how adopting a different convention to Eqn. (20) affects constraints on the measure μ; we consider this question in Ref. [15, Appendix B.5].

4 Semantics for ZX- and ZH-diagrams using discrete interals

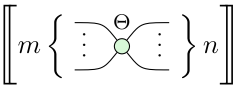
We present an approach to simply and systematically define semantic maps for ZX and ZH generators, which **(a)** yields simple diagrams for unitary transformations of interest, **(b)** admits scalar-exact diagrammatic rewrites involving few scalar gadgets, and **(c)** allows the two notational systems to be used seamlessly together.

Our approach is to subsume all considerations of normalising factors into the measure of a discrete integral, and its accompanying point-mass functions, as indicated on page 2. Our use of integrals and discrete measures in this way is standard, if somewhat uncommon in quantum information theory: see Ref. [35, 27] for comparable examples. Our intent is explicitly to draw attention to the freedom involved in the choice of measure, as a way forward to defining a semantic map $[\![\,\cdot\,]\!]$ for ZX and ZH diagrams that has desirable features.

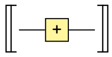
Defining a discrete integral on $\mathbf{D}$ with $\mu(\mathbf{D}) = \sqrt{D}$, as we do in the preceding Section, allows us to easily define semantic maps for ZX and ZH diagrams with a number of convenient properties. Let

$$|\omega^k\rangle\!\rangle \;=\; F|k\rangle\!\rangle \;=\; \int\limits_{x\in\mathbf{D}} \omega^{-kx}\,|x\rangle\!\rangle \tag{27}$$

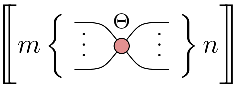
be the non-normalised point-mass distributions analogous to the Fourier basis states $|\omega^k\rangle$ introduced on page 4 (so that $|\omega^k\rangle\!\rangle$ is an ω^k-eigenvector of the cyclic shift operator X given by $X\,|a\rangle = |a{+}1\rangle$). We then define a semantic map $[\![\,\cdot\,]\!]$ on the ZX generators of Eqns. (6),



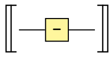
$$= \int_{x\in\mathbf{D}} \Theta(x)\; |x\rangle\!\rangle^{\otimes n} \langle\!\langle x|^{\otimes m}$$



$$= F^\dagger = \iint_{x,k\in\mathbf{D}} \mathrm{e}^{2\pi i k x/D} |k\rangle\!\rangle\langle\!\langle x| \tag{28}$$

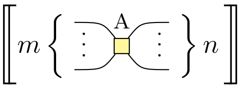


$$= \int_{k\in\mathbf{D}} \Theta(k)\; |\omega^{-k}\rangle\!\rangle^{\otimes n} \langle\!\langle\, \omega^{k}\, |^{\otimes m}$$

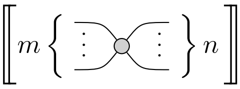


$$= F = \iint_{x,k\in\mathbf{D}} \mathrm{e}^{-2\pi i k x/D} |k\rangle\!\rangle\langle\!\langle x|$$

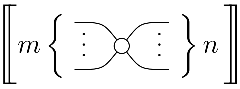
and the ZH generators of Eqn. (9):



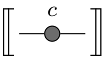
$$= \iint_{\substack{x\in\mathbf{D}^m\\ y\in\mathbf{D}^n}} \mathrm{A}(x_1\cdots x_m y_1\cdots y_n)\; |y\rangle\!\rangle\langle\!\langle x|\,,$$



$$= \iint_{\substack{x\in\mathbf{D}^m\\ y\in\mathbf{D}^n}} \Big\langle\!\!\Big\langle\, \sum_h x_h + \sum_k y_k \,\Big|\, 0 \,\Big\rangle\!\!\Big\rangle\;\; |y\rangle\!\rangle\langle\!\langle x|\,, \tag{29}$$



$$= \int_{x\in\mathbf{D}} |x\rangle\!\rangle^{\otimes n} \langle\!\langle x|^{\otimes m}\,,$$



$$= \int_{x\in\mathbf{D}} |{-c}{-x}\rangle\!\rangle\langle\!\langle x|\,,$$

These semantics are consistent with those set out in Eqns. (8) and (10), replacing the sums and the vectors $|x\rangle$ with discrete integrals and the corresponding point-mass distributions $|x\rangle\!\rangle$, and substitute proportionality relations with equalities. The discrete integrals (and point-mass functions) serve to specify specific scalar factors for the proportionalities.

These definitions are ones that we could choose to make, regardless of the measure μ that we consider for $\mathbf{D}$. Regardless of the choice made for ν, the above interpretations are certainly similar in their simplicity to the standard interpretations. By taking $\nu = D^{-1/4}$ as suggested in the preceding section, we not only obtain rewrite systems involving very few scalar gadgets – see Figs. 1 and 2 – but also, the most commonly considered states and unitary operations of qudit circuits admit simple presentations using these semantics. We may demonstrate this as follows.

4.1 The stabiliser sub-theory of ZX for arbitrary $D > 1$

We describe below a *stabiliser subtheory* of the ZX calculus, concerning ZX diagrams which suffice to represent stabiliser states [13] on systems of arbitrary dimension $D > 1$. These are characterised by ZX diagrams whose phase parameters are governed by restricted functions, for which arithmetic modulo D plays a central role.

We begin by describing the stabiliser sub-theory of quantum circuits. Following Ref. [13], define the complex unit $\tau = \mathrm{e}^{\pi i(D^2+1)/D}$, which is relevant to the analysis of stabiliser circuits on qudits of dimension D. The scalar τ is defined in such a way that $\tau^2 = \omega$, but also so that $\tau X^\dagger Z^\dagger$ is an operator of order D, where X and Z given by

$$X\,|t\rangle \;=\; |t+1\rangle\,, \qquad\qquad Z\,|t\rangle \;=\; \omega^t\,|t\rangle\,, \tag{30}$$

are the D-dimensional generalised Pauli operators. (As always, arithmetic performed in the kets are evaluated modulo D.) Choosing τ in this way makes it possible [13] to define a simple and uniform theory of unitary stabiliser circuits on qudits of dimension D, generated

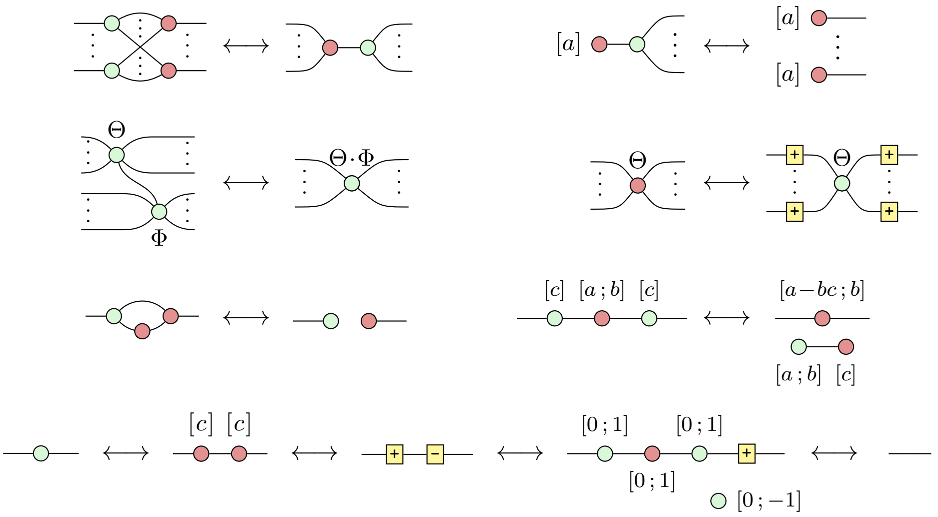


Figure 1 A sample of the (scalar-exact) rewrites which are sound for ZX diagrams with semantics as in Eqn. (28), when $\nu = D^{-1/4}$. Throughout, we have $\Theta, \Phi : \mathbb{Z} \to \mathbb{C}$, and $a, b, c \in \mathbb{Z}$. Node labels of the form $[a]$ or $[a\,;b]$ stand respectively for the amplitude functions $x \mapsto \tau^{2ax}$ and $x \mapsto \tau^{2ax+bx^2}$, where $\tau = \exp(\pi i(D^2{+}1)/D)$. A more complete list of rewrites, and proofs of their soundness, may be found in Ref. [15, Appendix A.3.3].

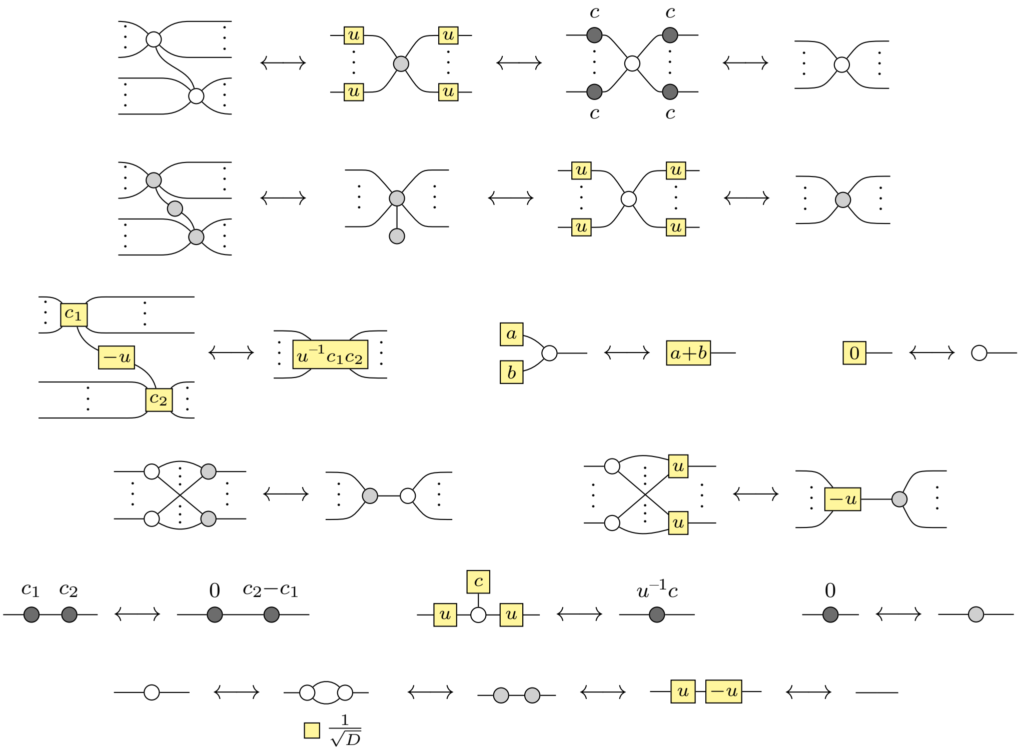


Figure 2 A sample of the (scalar-exact) rewrites which are sound for ZH diagrams with semantics as in Eqn. (29) when $\nu = D^{-1/4}$. H-boxes which are labeled inside with an integer parameter such as $c \in \mathbb{Z}$, indicate an amplitude of $\omega^c = \mathrm{e}^{2\pi i c/D}$; H-boxes labelled with "+" or "−" indicate $c = \pm 1$. Throughout, we have$a, b, c, c_1, c_2, u, v \in \mathbb{Z}$ (which may be evaluated modulo D), where in particular u and v are coprime to D. A more complete list of rewrites, and proofs of their soundness, may be found in Ref. [15, Appendix A.3.1].

by the single-qudit operators [9]

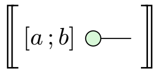
$$S \;=\; \int\limits_{x\in\mathbf{D}} \tau^{\,x^2}\;|x\rangle\!\rangle\langle\!\langle x|\;; \qquad F \;=\; \iint\limits_{k,x\in\mathbf{D}} \tau^{-2kx}\;|k\rangle\!\rangle\langle\!\langle x|\;; \qquad M_u \;=\; \int\limits_{x\in\mathbf{D}} |ux\rangle\!\rangle\langle\!\langle x|\;, \tag{31}$$

where in the case of M_u we restrict to $u \in \mathbb{Z}$ which is relatively prime to D; and either one of the two-qudit operators

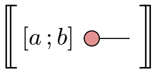
$$\mathrm{CX} \;=\; \iint\limits_{x,y\in\mathbf{D}} |x\rangle\!\rangle\langle\!\langle x| \otimes |x{+}y\rangle\!\rangle\langle\!\langle y|\;; \qquad\qquad \mathrm{CZ} \;=\; \iint\limits_{x,y\in\mathbf{D}} \tau^{2xy}\;|x,y\rangle\!\rangle\langle\!\langle x,y|\;. \tag{32}$$

Finally, the full range of stabiliser circuits also admit measurements in the standard basis (and bases which may be related to the standard basis by the above unitaries).

Booth and Carette [6] describe a version of the ZX calculus which is complete for this subtheory, for the special case of D an odd prime. Following them, we may describe how the semantics of Eqn. (28) allows a simplification of these rewrites, extending them in most cases to arbitrary $D > 1$. To this end, it will be helpful to use a slightly different notational convention to Booth and Carette [6], we may easily denote these with ZX diagrams using the semantics of Eqn. (28). For $a, b \in \mathbb{Z}$, when parameterising a green or red dot, let $[a\,;b]$ stand for the amplitude function $\Theta(x) = \tau^{2ax+bx^2}$, so that

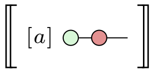


$$=\; \int\limits_{x\in\mathbf{D}} \tau^{\,2ax\,+\,bx^2}\;|x\rangle\!\rangle\;;$$

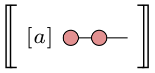


$$=\; \int\limits_{k\in\mathbf{D}} \tau^{\,2ak\,+\,bk^2}\;|\omega^{-k}\rangle\!\rangle\;; \tag{33}$$

generalising these to dots with multiple edges (or with none) similarly to Ref. [6]. When $b = 0$, we may abbreviate this function simply by $[a]$, so that we may represent the states $|a\rangle\!\rangle$ and $|\omega^a\rangle\!\rangle$ straightforwardly (albeit with the use of auxiliary red dots to represent an antipode operator, mapping $|\omega^a\rangle\!\rangle \mapsto |\omega^{-a}\rangle\!\rangle$ and $|a\rangle\!\rangle \mapsto |{-a}\rangle\!\rangle$ for $a \in \mathbb{Z}_D$):

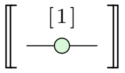


$$=\; \iint\limits_{k,x\in\mathbf{D}} \tau^{2ax}\,|\omega^{-k}\rangle\!\rangle\langle\!\langle \omega^k\,|x\rangle\!\rangle \;=\; |\omega^a\rangle\!\rangle\,; \tag{34a}$$

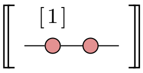


$$=\; \iint\limits_{h,k\in\mathbf{D}} \tau^{2ah}\,|\omega^{-k}\rangle\!\rangle\langle\!\langle \omega^k\,|\omega^{-h}\rangle\!\rangle \;=\; |a\rangle\!\rangle\,. \tag{34b}$$

We may also easily represent the operators Z, and X as $1 \to 1$ dots:

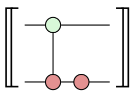


$$=\; \int\limits_{x\in\mathbf{D}} \tau^{2x}\,|x\rangle\!\rangle\langle\!\langle x| \;=\; Z\,;$$

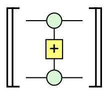


$$=\; \int\limits_{h\in\mathbf{D}} \tau^{2h}\,|\omega^h\rangle\!\rangle\langle\!\langle \omega^h| \;=\; X\,. \tag{35}$$

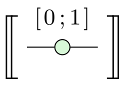
Regarding the unitary stabiliser operators on qudits, we may express them without any phases, using multi-edges between green and red dots, or using Hadamard boxes:



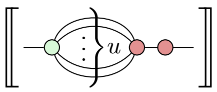
$= \mathrm{CX},$



$= \mathrm{CZ},$



$= S,$

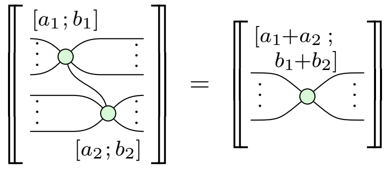


$= M_u\,.$ (36)

(The diagram shown for M_u also generalises to operators $M_u = \int_x |ux\rangle\!\rangle\langle\!\langle x|$ for u not a multiplicative unit modulo D, though in that case the operator will not be invertible.)

[9] Despite the different convention we adopt for the labeling of the standard basis, the definitions below are equivalent to those of Ref. [13]: the relative phases τ^{2ax+bx^2} remain well-defined on substitution of values $x < 0$ with $D + x$, as $\tau^{2a(D+x)+b(D+x)^2} = \tau^{2aD+2ax+bD^2+2bD+bx^2} = \tau^{2ax+bx^2}$ (using the fact that $\tau^{D^2} = \tau^{2D} = 1$ for both even and odd D).

The stabiliser subtheory of ZX may produce green or red dots *of degree zero* with phase parameter $[a\ ;\, b]$ for some $a, b \in \mathbb{Z}$. These may occur when evaluating the probability of measurement outcomes (e.g., in the standard basis) arising from a stabiliser qudit circuit. As we show in Ref. [15, Appendix A.3.3] (as a simple corollary of a more general fusion rule), we have



(37)

where each instance of "$\vdots$" denotes some number (zero or more) of incident wires. In the case where there are *no* other dots connected to two green dots as above, the right-hand side would be an isolated dot denoting a scalar, for which we define the notation $\Gamma(a, b, D)$:

$$\left[\!\!\left[\ [a\,;b]\,\bigcirc\ \right]\!\!\right] \;=\; \int\limits_{x\in\mathbf{D}} \tau^{2ax+bx^2} \;=:\; \Gamma(a,b,D)\ . \tag{38}$$

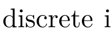
Evaluating such a discrete integral is connected with the subject of quadratic Gaussian sums, which is addressed in some detail in Ref. [15, Appendix C]. As a result of the normalisation convention for our discrete integrals, it is possible to show (see Ref. [15, Appendix C, Eqn. (103)]) that

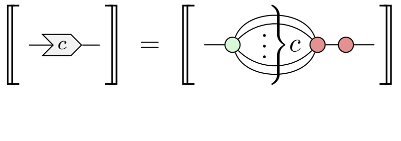
$$\Gamma(a,b,D) \;:=\; \int\limits_{x\in\mathbf{D}} \tau^{2ax+bx^2} \;=\; \begin{cases} \sqrt{t}\cdot \mathrm{e}^{i\gamma}, & \text{if } t = \gcd(b,D) \text{ and } a \text{ is divisible by } t; \\ \quad 0, & \text{otherwise,} \end{cases} \tag{39}$$

where γ is a phase parameter described in more detail in Ref. [15, Appendix C, Eqn. (103)]. In particular, if b is a multiplicative unit modulo D, this represents a global phase factor. (If we also have $a = 0$, then $\Gamma(a, b, D)$ is in fact a power of $\mathrm{e}^{\pi i/4}$.) More generally, $\Gamma(a, b, D)$ will either be 0, or have magnitude $\sqrt{t}$, where $t = \gcd(b, D)$.

In this way, we obtain a diagrammatic language which is capable of expressing the rewrites similar to those described by Ref. [6], while involving fewer scalar factors (see Ref. [15, Appendix A.3.3, Fig. 6] for a more complete list of sound rewrites).

4.2 Multipliers and multicharacters in qudit ZH

It would be cumbersome to reason about stabiliser multiplication operators M_u or iterated CX or CZ gates using parallel edges between dots. Booth and Carette [6] describe how these may be denoted using gadgets called "multipliers", denoted  for $c \in \mathbb{N}$, which represent a limited form of scalable ZX notation [9, 8]. Using discrete integrals and the semantics described in Eqn. (28), we would simply write



$$= \int\limits_{x\in\mathbf{D}} |cx\rangle\!\rangle\langle\!\langle x|\ . \tag{40}$$

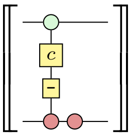
Using these multipliers, Booth and Carette [6, p. 24] then define "Fourier boxes" $-\boxed{c}-$:= $-\succ c\rangle-\boxed{+}-$ (using our notation for Hadamard boxes), whose interpretation coincides with the ones we assign using Eqn. (29) to an H-box with an amplitude parameter ω^c. Using this as a primitive, and composing this with the inverse $-\boxed{-}-$ of the positive Hadamard box $-\boxed{+}-$, we may directly describe multipliers instead as a ZH gadget, loosely following Roy [33]:

$$-\boxed{c}-\boxed{-}- \quad =: \quad -\succ c\rangle- \;. \tag{41}$$

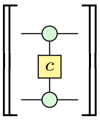
On the left, we employ a short-hand for H-boxes with an amplitude parameter ω^c. This is short-hand for a character function $\chi_c : \mathbb{Z} \to \mathbb{C}$ given by $\chi_c(x) = \omega^{cx}$, which is well-defined modulo D, and which we may then regard as a character on $\mathbb{Z}_D$. The function $\mathbb{Z} \times \mathbb{Z} \to \mathbb{C}$ given by $(x, y) \mapsto \chi_c(xy)$ is a *bicharacter*, which is also well-defined modulo D on each of its arguments; and more generally we may consider *multicharacters*, which are functions $\mathbb{Z}_D \times \cdots \times \mathbb{Z}_D \to \mathbb{C}$ given by $(x_1, \ldots, x_n) \mapsto \omega^{cx_1 \cdots x_n}$. We may call H-boxes with any number of edges, and with amplitude parameter ω^c for some $c \in \mathbb{Z}_D$, a ($\mathbb{Z}_D$-)*multicharacter box*.

We may use these ideas to define a *multicharacter subtheory* of ZH, consisting of the subtheory in which the H-boxes are indexed by paramters $c \in \mathbb{Z}_D$ in this way. Roy [33] has substantially investigated this fragment of ZH, in odd prime dimension. Our choice of semantic map allows us [15, Appendix A.3.1, Fig. 4] to reproduce many of the rewrites considered by Roy, while making minor simplifications and extending them to arbitrary dimensions $D > 1$.

We may use multiplier gadgets and multicharacter boxes to usefully describe unitary transformations, such as exponentiations of the qudit controlled-X and controlled-Z gates:

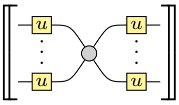


$$= \mathrm{CX}^c = \iint\limits_{x,y\in\mathbf{D}} |x, y{+}cx\rangle\!\rangle\langle\!\langle x, y| \;,$$

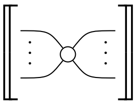


$$= \mathrm{CZ}^c = \iint\limits_{x,y\in\mathbf{D}} \omega^{cxy}\, |x, y\rangle\!\rangle\langle\!\langle x, y| \;. \tag{42}$$

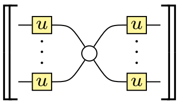
These degree-2 multicharacter boxes are effectively a Fourier transform over an isomorphic presentation of $\mathbb{Z}_D$ in some cases. This occurs in particular when $c = u \in \mathbb{Z}_D^\times$ is a multiplicative unit modulo D. We can witness this by rewrites which are valid for H-boxes parameterised by units $u \in \mathbb{Z}_D^\times$, such as ones which relate the white dots and the gray dots among the ZH generators (similar remarks apply for the green and red ZX generators):



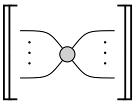
$=$



and



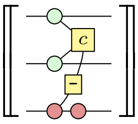
$=$



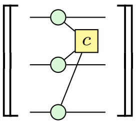
(43)

While there cannot be any perfect symmetry between the white and gray dots in general in the ZH calculus (as it involves the standard basis as a preferred basis), in this case a symmetry is recovered which one does not normally expect of presentations of the ZH calculusx.

We may also easily describe multi-qudit analogues of the qudit controlled-X and controlled-Z gates, using the fact that the H-boxes denote multi-characters. For example:



$$= \mathrm{CCX}^c = \iiint\limits_{x,y,z\in\mathbf{D}} |x, y, z{+}cxy\rangle\!\rangle\langle\!\langle x, y, z| \;, \tag{44a}$$



$$= \mathrm{CCZ}^c = \iiint\limits_{x,y,z\in\mathbf{D}} \omega^{cxyz}\, |x, y, z\rangle\!\rangle\langle\!\langle x, y, z| \;. \tag{44b}$$

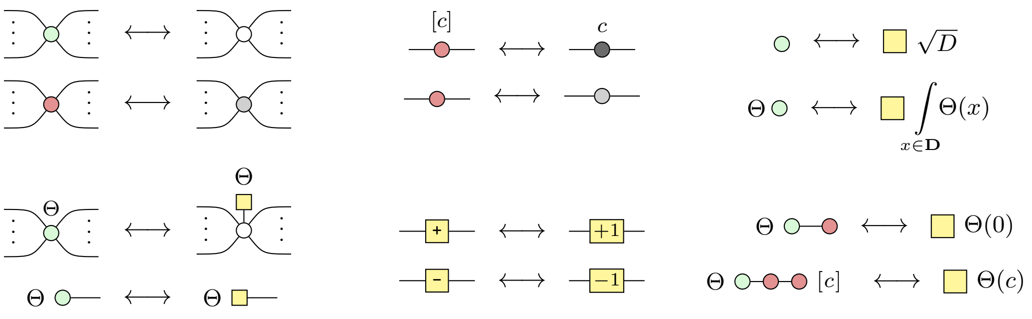
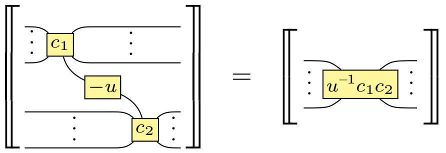


Figure 3 Sound rewrites between the ZX generators and the ZH generators, subject to the semantics of Eqns. (28) and (29) in the case $\nu = D^{-1/4}$. The proofs of the soundness of these rewrites are shown in Ref. [15, Appendix A.3.2].

While it would quickly become cumberson to represent each of the integrals in such an operation – this being a motivation for diagrammatic calculi in general – this demonstrates the genericity of the representation for these unitary transformations, and the relative lack of minor details to attend to in using them. Finally, we note the quasi-spider property that H-boxes are known for in the qubit case and in (and also shown in a more complicated form for odd prime D by Roy [33]), which can also be shown for a pair of multicharacter boxes connected to a common H-box with parameter $u \in \mathbb{Z}_D^\times$:



(45)

We do not claim to have a complete multicharacter subtheory for ZH over arbitrary qudits, but many of the rewrites which one may show in this case [15, Appendix A.3.1, Fig. 4] can be specialised in a useful way to the multicharacter case.

4.3 Compatibility and universality

In addition to the semantics of Eqns. (28) and (29) yielding ZX and ZH calculi which are each convenient in their own right, it also assigns the same semantics to certain ZX generators and certain ZH generators. This is illustrated in Fig. 3. This allows us to relate the two calculi to each other, to describe a “ZXH calculus” which has the features of both.

It is not necessary to consider such a united calculus to be able to denote arbitrary operators: see Ref. [15, Appendix A.2] for a sketch of a proof of universality of the ZH diagrams, in terms of a “normal form”-like diagram which mirrors the construction of Ref. [3]. However, while using both sets of generators may be redundant in principle, it should be expected to facilitate analysis, as the rewrite rules of each system effectively represents at least an important Lemma or Theorem of the other system.

We do not demonstrate any completeness results for these calculi; more rewrites may be necessary to prove completeness for arbitrary $D > 1$, for each of these two calculi, even in the stabiliser and multicharacter fragments described above.

5 Discussion

As well as providing an approach to define ZX and ZH calculi with simple rewrite systems on qudits, this approach is a simpler, and apparently independent, way to reproduce[10] the "well-tempered" semantic map $[\![\,\cdot\,]\!]_\nu$ described in Ref. [14] for $D = 2$. In this way, Eqns. (28) and (29) provide a more intuitive definition of those semantics, and extend them to arbitrary $D > 1$. It is possible to show that this is essentially down to the constraints imposed on the representation of the discrete Fourier transform in Section 3.2. Ref. [15, Appendix B] describes **(a)** the way that Eqns. (28) and (29) fail to constrain a generic "Ockhamic" interpretation of ZH diagrams while fixing a specific "Ockhamic" interpretations of ZX diagrams; and **(b)** to what extent this approach to fixing semantics actually differs from the approach of Ref. [14].

As well as discrete integrals, and amplitude functions $\Theta, \mathrm{A} : \mathbb{Z} \to \mathbb{C}$ in place of (vectors of) phases or amplitudes, we consider an index set for $\mathcal{H}$ which is not simply $\{0, 1, \ldots, D{-}1\}$, but instead $\{L, L+1, \ldots, U-1, U\}$ for some integers such that $U - L + 1 = D$. One conventional choice is $L = 0$ and $U = D - 1$, but most of out results (in particular: all those to do with the stabiliser / multicharacter fragments of ZX or ZH) hold equally well with any such set of labels for the standard basis. This less committal choice of index set demonstrates the flexibility of this system, which may prove useful for future applications (e.g., problems in physics where it may prove useful to consider negative index values).

We conclude with a highly speculative thought regarding discrete measures. One constraint which we imposed on the measure μ on $\mathbf{D}$ – interpreted as a measure on $\mathbb{Z}_D$ – was that the Fourier transform should be interpretable as an involution $\mathbb{C}^{(\mathbb{Z}_D,\mu)} \to \mathbb{C}^{(\mathbb{Z}_D,\mu)}$ on functions on the measure space $(\mathbb{Z}_D, \mu)$, rather than a bijection $\mathbb{C}^{(\mathbb{Z}_D,\mu)} \to \mathbb{C}^{(\mathbb{Z}_D,\mu')}$ between functions on distinct measure spaces $(\mathbb{Z}_D, \mu)$ and $(\mathbb{Z}_D, \mu')$. This may seem like a technical but necessary step; for a conventional presentation of ZX diagrams, it *is* necessary, if all of the wires are to have the same type. However, many quantum algorithms have a structure in which some classical operation with a distinguished control register, where that control operates on a state which is conceived as being in the Fourier basis. This structure is consistent with the control register having different "datatypes" at different stages of the algorithm. Could it be more appropriate to make a distinction on logical qudits of each dimension D, between a "standard" type and a "Fourier" type (possibly among others), than to have just a single "type" for each D? It would be interesting to consider what insights into the structure of quantum algorithms might arise by investigating along these lines; it is conceivable that this could give rise to new insights into structured quantum programming.

References

1 Dorit Aharonov. A simple proof that Toffoli and Hadamard are quantum universal, 2003. [arXiv:quant-ph/0301040]. `doi:10.48550/arXiv.quant-ph/0301040`.

2 Miriam Backens. Making the stabilizer ZX-calculus complete for scalars. In Chris Heunen, Peter Selinger, and Jamie Vicary, editors, *Proceedings of the 12th International Workshop on Quantum Physics and Logic (QPL 2015)*, Electronic Proceedings in Theoretical Computer Science, pages 17–32. Open Publishing Association, november 2015. See also [arXiv:1507.03854]. `doi:10.4204/EPTCS.195.2`.

[10] Strictly speaking, the calculi of Ref. [14] involve red and green dots with parameters $\theta \in \mathbb{R}$, H-boxes with parameters $\alpha \in \mathbb{C}$, only one type of Hadamard box instead of two, and a "nu box" which is missing from the calculus presented here. We may bridge these differences using the short-hand described for phases / amplitudes to parameterise these nodes, identifying both Hadamard plus and minus boxes with the single Hadamard box of Ref. [14], and replacing the nu-boxes with some suitable scalar gadgets (such as H-boxes parameterised by powers of $\nu = D^{-1/4}$).

3 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. *Electronic Proceedings in Theoretical Computer Science*, 287:23–42, January 2019. See also [arXiv:1805.02175]. doi:10.4204/eptcs.287.2.

4 Miriam Backens, Aleks Kissinger, Hector Miller-Bakewell, John van de Wetering, and Sal Wolffs. Completeness of the ZH-calculus. *Compositionality*, Volume 5 (2023), July 2023. See also [arXiv:2103.06610]. doi:10.32408/compositionality-5-5.

5 Miriam Backens, Simon Perdrix, and Quanlong Wang. A simplified stabilizer ZX-calculus. *Electronic Proceedings in Theoretical Computer Science*, 236:1–20, January 2017. See also [arXiv:1602.04744]. doi:10.4204/eptcs.236.1.

6 Robert I. Booth and Titouan Carette. Complete ZX-calculi for the stabiliser fragment in odd prime dimensions. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. See also [arXiv:2204.12531]. doi:10.4230/LIPIcs.MFCS.2022.24.

7 Titouan Carette. *Wielding the ZX-calculus, Flexsymmetry, Mixed States, and Scalable Notations*. Theses, Université de Lorraine, November 2021. URL: https://hal.science/tel-03468027.

8 Titouan Carette, Yohann D'Anello, and Simon Perdrix. Quantum algorithms and oracles with the scalable ZX-calculus. *Electronic Proceedings in Theoretical Computer Science*, 343:193–209, September 2021. See also [arXiv:2104.01043]. doi:10.4204/eptcs.343.10.

9 Titouan Carette, Dominic Horsman, and Simon Perdrix. SZX-Calculus: Scalable Graphical Quantum Reasoning. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. See also [arXiv:1905.00041]. doi:10.4230/LIPIcs.MFCS.2019.55.

10 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, April 2011. See also [arXiv:0906.4725]. doi:10.1088/1367-2630/13/4/043016.

11 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.

12 Bob Coecke and Quanlong Wang. ZX-rules for 2-qubit Clifford+T quantum circuits. In *International Conference on Reversible Computation*, pages 144–161. Springer, 2018. See also [arXiv:1804.05356]. doi:10.1007/978-3-319-99498-7_10.

13 Niel De Beaudrap. A linearized stabilizer formalism for systems of finite dimension. *Quantum Information & Computation*, 13(1–2):73–115, january 2013. See also [arXiv:1102.3354].

14 Niel de Beaudrap. Well-tempered ZX and ZH calculi. *Electronic Proceedings in Theoretical Computer Science*, 340:13–45, September 2021. See also [arXiv:2006.02557]. doi:10.4204/eptcs.340.2.

15 Niel de Beaudrap and Richard D. P. East. Simple ZX and ZH calculi for arbitrary finite dimensions, via discrete integrals, 2023. [arXiv:2304.03310v2] – version 2. doi:10.48550/arXiv.2304.03310.

16 Niel de Beaudrap, Aleks Kissinger, and Konstantinos Meichanetzidis. Tensor network rewriting strategies for satisfiability and counting. In Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, editors, *Proceedings 17th International Conference on Quantum Physics and Logic, Paris, France, June 2 – 6, 2020*, volume 340 of *Electronic Proceedings in Theoretical Computer Science*, pages 46–59. Open Publishing Association, 2021. See also [arXiv:2004.06455]. doi:10.4204/EPTCS.340.3.

17 Giovanni de Felice and Bob Coecke. Quantum linear optics via string diagrams. In Stefano Gogioso and Matty Hoban, editors, *Proceedings 19th International Conference on Quantum Physics and Logic, Wolfson College, Oxford, UK, 27 June – 1 July 2022*, volume 394 of *Electronic Proceedings in Theoretical Computer Science*, pages 83–100. Open Publishing Association, 2023. See also [arXiv:2204.12985]. doi:10.4204/EPTCS.394.6.

18 Richard D. P. East, Pierre Martin-Dussaud, and John van de Wetering. Spin-networks in the ZX-calculus, 2021. [arXiv:2111.03114]. doi:10.48550/arXiv.2111.03114.

19 Richard D.P. East, John van de Wetering, Nicholas Chancellor, and Adolfo G. Grushin. AKLT-states as ZX-diagrams: Diagrammatic reasoning for quantum states. *PRX Quantum*, 3:010302, january 2022. See also [arXiv:2012.01219. doi:10.1103/PRXQuantum.3.010302.

20 Xiaoyan Gong and Quanlong Wang. Equivalence of local complementation and Euler decomposition in the qutrit ZX-calculus, 2017. [arXiv:1704.05955].

21 Emmanuel Jeandel, Simon Perdrix, and Margarita Veshchezerova. Addition and differentiation of ZX-diagrams. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. [arXiv:2202.11386]. doi:10.4230/LIPIcs.FSCD.2022.13.

22 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the ZX-calculus for Clifford+T quantum mechanics. In *2018 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, LICS '18, pages 559–568, New York, NY, USA, 2018. Association for Computing Machinery. See also [arXiv:1705.11151]. doi:10.1145/3209108.3209131.

23 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of the ZX-calculus. *Logical Methods in Computer Science*, June 2020. See also [arXiv:1903.06035]. doi:10.23638/LMCS-16(2:11)2020.

24 Emmanuel Jeandel, Simon Perdrix, Renaud Vilmart, and Quanlong Wang. ZX-calculus: Cyclotomic supplementarity and incompleteness for Clifford+T quantum mechanics. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. See also [arXiv:1702.01945]. doi:10.4230/LIPIcs.MFCS.2017.11.

25 Stach Kuijpers, John van de Wetering, and Aleks Kissinger. Graphical Fourier theory and the cost of quantum addition, 2019. [arXiv:1904.07551].

26 Tuomas Laakkonen, Konstantinos Meichanetzidis, and John van de Wetering. A graphical #SAT algorithm for formulae with small clause density, 2022. [arXiv:2212.08048]. doi:10.48550/arXiv.2212.08048.

27 Shahn Majid. Quantum and braided ZX calculus. *Journal of Physics A: Mathematical and Theoretical*, 55(25):254007, june 2022. See also [arXiv:2103.07264]. doi:10.1088/1751-8121/ac631f.

28 Kang Feng Ng and Quanlong Wang. A universal completion of the ZX-calculus, 2017. [arXiv:1706.09877]. doi:10.48550/arXiv.1706.09877.

29 Kang Feng Ng and Quanlong Wang. Completeness of the ZX-calculus for pure qubit Clifford+T quantum mechanics, January 2018. doi:10.48550/arXiv.1801.07993.

30 Simon Perdrix and Quanlong Wang. Supplementarity is necessary for quantum diagram reasoning. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76:1–76:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. See also [arXiv:1506.03055]. doi:10.4230/LIPIcs.MFCS.2016.76.

31 Boldizsár Poór, Robert I. Booth, Titouan Carette, John van de Wetering, and Lia Yeh. The qupit stabiliser ZX-travaganza: Simplified axioms, normal forms and graph-theoretic simplification. *Electronic Proceedings in Theoretical Computer Science*, 384:220–264, August 2023. See also [arXiv:2306.05204]. doi:10.4204/eptcs.384.13.

32 Boldizsár Poór, Quanlong Wang, Razin A. Shaikh, Lia Yeh, Richie Yeung, and Bob Coecke. Completeness for arbitrary finite dimensions of ZXW-calculus, a unifying calculus. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14, 2023. See also [arXiv:2302.12135]. doi:10.1109/LICS56636.2023.10175672.

33 Patrick Roy. Qudit ZH-calculus. Master's thesis, University of Oxford, 2022.

34 Patrick Roy, John van de Wetering, and Lia Yeh. The qudit ZH-calculus: Generalised Toffoli+Hadamard and universality. *Electronic Proceedings in Theoretical Computer Science*, 384:142–170, August 2023. See also [arXiv:2307.10095]. doi:10.4204/eptcs.384.9.

35 Dirk Schlingemann. Cluster states, algorithms and graphs. *Quantum Information & Computation*, 4(4):287–324, july 2004. See also [arXiv:quant-ph/0305170].

36 Yaoyun Shi. Both Toffoli and controlled-NOT need little help to do universal quantum computing. *Quantum Information & Computation*, 3(1):84–92, january 2003. See also [arXiv:quant-ph/0205115].

37 Tobias Stollenwerk and Stuart Hadfield. Diagrammatic analysis for parameterized quantum circuits. In Stefano Gogioso and Matty Hoban, editors, *Proceedings 19th International Conference on Quantum Physics and Logic, Wolfson College, Oxford, UK, 27 June – 1 July 2022*, volume 394 of *Electronic Proceedings in Theoretical Computer Science*, pages 262–301. Open Publishing Association, 2023. See also [arXiv:2204.01307]. doi:10.4204/EPTCS.394.15.

38 Alexis Toumi, Richie Yeung, and Giovanni de Felice. Diagrammatic differentiation for quantum machine learning. In Chris Heunen and Miriam Backens, editors, *Proceedings 18th International Conference on Quantum Physics and Logic, Gdansk, Poland, and online, 7-11 June 2021*, volume 343 of *Electronic Proceedings in Theoretical Computer Science*, pages 132–144. Open Publishing Association, 2021. See also [arXiv:2103.07960]. doi:10.4204/EPTCS.343.7.

39 Alex Townsend-Teague and Konstantinos Meichanetzidis. Simplification strategies for the qutrit ZX-calculus, 2021. [arXiv:2103.06914]. doi:10.48550/arXiv.2103.06914.

40 John van de Wetering and Sal Wolffs. Completeness of the phase-free ZH-calculus, 2019. [arXiv:1904.07545]. doi:10.48550/arXiv.1904.07545.

41 John van de Wetering and Lia Yeh. Building qutrit diagonal gates from phase gadgets. *Electronic Proceedings in Theoretical Computer Science*, 394:46–65, November 2023. See also [arXiv:2204.13681]. doi:10.4204/eptcs.394.4.

42 Renaud Vilmart. A near-minimal axiomatisation of ZX-calculus for pure qubit quantum mechanics. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10, 2019. See also [arXiv:1812.09114]. doi:10.1109/LICS.2019.8785765.

43 Renaud Vilmart. A ZX-calculus with triangles for Toffoli-Hadamard, Clifford+T, and beyond. *Electronic Proceedings in Theoretical Computer Science*, 287:313–344, January 2019. See also [arXiv:1804.03084]. doi:10.4204/eptcs.287.18.

44 Quanlong Wang. Qutrit ZX-calculus is complete for stabilizer quantum mechanics. *Electronic Proceedings in Theoretical Computer Science*, 266:58–70, February 2018. See also [arXiv:1803.00696]. doi:10.4204/eptcs.266.3.

45 Quanlong Wang. On completeness of algebraic ZX-calculus over arbitrary commutative rings and semirings, 2019. [arXiv:1912.01003]. doi:10.48550/arXiv.1912.01003.

46 Quanlong Wang. Algebraic complete axiomatisation of ZX-calculus with a normal form via elementary matrix operations, 2020. [arXiv:2007.13739]. doi:10.48550/arXiv.2007.13739.

47 Quanlong Wang. An algebraic axiomatisation of ZX-calculus. In Benoît Valiron, Shane Mansfield, Pablo Arrighi, and Prakash Panangaden, editors, *Proceedings 17th International Conference on Quantum Physics and Logic, Paris, France, June 2 – 6, 2020*, volume 340 of *Electronic Proceedings in Theoretical Computer Science*, pages 303–332. Open Publishing Association, 2021. See also [arXiv:1911.06752]. doi:10.4204/EPTCS.340.16.

48 Quanlong Wang. Qufinite ZX-calculus: a unified framework of qudit ZX-calculi, 2021. [arXiv:2104.06429]. doi:10.48550/arXiv.2104.06429.

49 Quanlong Wang, Richie Yeung, and Mark Koch. Differentiating and integrating ZX diagrams with applications to quantum machine learning, 2022. [arXiv:2201.13250]. doi:10.48550/arXiv.2201.13250.

On Complexity of Confluence and Church-Rosser Proofs

Arnold Beckmann
Department of Computer Science, Swansea University, UK

Georg Moser
Department of Computer Science, University of Innsbruck, Austria

Abstract

In this paper, we investigate *confluence* and the *Church-Rosser property* – two well-studied properties of rewriting and the λ-calculus – from the viewpoint of proof complexity. With respect to confluence, and focusing on orthogonal term rewrite systems, our main contribution is that the size, measured in number of symbols, of the smallest rewrite proof is polynomial in the size of the peak. For the Church-Rosser property we obtain exponential lower bounds for the size of the join in the size of the equality proof. Finally, we study the complexity of proving confluence in the context of the λ-calculus. Here, we establish an exponential (worst-case) lower bound of the size of the join in the size of the peak.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases logic, bounded arithmetic, consistency, rewriting

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.21

Funding *Arnold Beckmann*: Royal Society International Exchanges Grant, IES\R3\223051
Georg Moser: Royal Society International Exchanges Grant, IES\R3\223051

1 Introduction

Confluence and the Church-Rosser property are two (very) well-known properties of rewriting that have been studied for several decades. *Confluence* expresses that if we have terms s, t, t', where s can be successively rewritten to t, as well as to t', then t and t' have a common descendent in the rewriting relation, cf. Figure 1 i). In short, if there is a *peak*: $t\ {}^*\!\!\leftarrow s \rightarrow^* t'$, we conclude the existence of a *rewrite proof*: $t \rightarrow^* \cdot\ {}^*\!\!\leftarrow t'$. The *Church-Rosser property* – illustrated in Figure 1 ii) – expresses that from the equality between t and t' ($t \leftrightarrow^* t'$), we conclude the existence of a rewrite proof: $t \rightarrow^* \cdot\ {}^*\!\!\leftarrow t'$. It is a folklore result that both properties are equivalent. And, as indicative in the name, their intensive study goes back to work by Church and Rosser [8].

Despite the large body of work on confluence and the Church-Rosser property, it seems that the, to us, natural question about the inherent proof complexities has only received scarce attention. A noteworthy exception is work by Ketema and Grue Simonsen [11]. Focusing on orthogonal term rewrite systems and employing the number of reductions as measure of proof complexity, they obtain in the context of confluence optimal exponential upper bounds on the size of the rewrite proof in relation to the size of the peak. With respect to the Church-Rosser property only a non-elementary upper bound can be shown. Related results have been obtained for the λ-calculus, where again non-elementary bounds are obtained for both properties, cf. [10].

If, however, proof complexity is measured more in the tradition of computational complexity, that is, as the number of symbols occurring in a proof, then more tractable results are possible. For example for orthogonal term rewrite systems, we prove that for confluence the size of the least rewrite proof is always polynomially bounded in the size of the peak.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 21; pp. 21:1–21:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

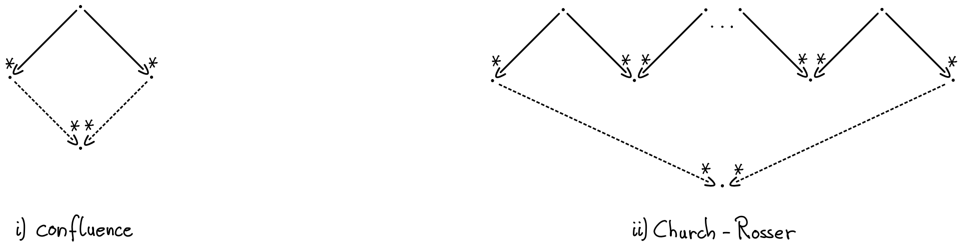


Figure 1 Confluence and Church-Rosser property.

Motivation. These results may open the way for the application of rewriting techniques in complexity theoretic studies, in particular in the context of Bounded Arithmetic [6]. A major open problem in Bounded Arithmetic is the separation of its fragments, which has deep connections to similar questions about the separation of computational complexity classes like the Polynomial Time Hierarchy, including the P vs. NP problem. Consider equational theories, restricted to term equations that define functions symbols exclusively by recursion. As established in [5] by the first author, consistency of such equational theories can be proved in the fragment of Bounded Arithmetic S^1_2. This is remarkable, as it disproves the general impression in Bounded Arithmetic, that consistency statements cannot be used for separation arguments - consistency of equational theories with a richer set of axioms are usually unprovable in Bounded Arithmetic [7].

In the proof in [5], the given equational proof is reconstructed in S^1_2 using a technically involved process of "approximation" and "calculation". An alternative, much more elegant, proof could employ the Church-Rosser property of the induced term rewrite system. To our best knowledge it is, however, unclear whether this property (or confluence) is formalisable in S^1_2. The results of this paper are conceivable as a first step towards this direction.

Contributions. In summary, we make the following contributions, where we are only concerned with *orthogonal* term rewrite systems.

1) Our main result, Theorem 17, shows that the size – measured in the number of symbols – of the smallest possible rewrite proofs is in the worst-case polynomially bounded in the size of the peak, cf. Figure 1. This shows that confluence properties are polynomial time computable, hence are formalisable in Bounded Arithmetic.
The polynomial (in fact biquadratic) upper bound stems from a quadratic bound on the number of reductions in the rewrite proof in the size of the peak, and a quadratic bound on the size of each term in the rewrite proof.

2) For the Church-Rosser property we give an exponential worst-case lower bound to the size of the join in the size of the equality proof, cf. Theorem 19. This shows that it is not possible to formalise Church-Rosser properties directly in Bounded Arithmetic. The (worst-case) bound is precise.

3) We give matching (worst-case) upper and lower bounds based on different complexity measures. For confluence, we show that the size of the join is linear in the size of the product of the end terms in the peak, cf. Corollary 15 and Proposition 10. For the Church-Rosser property, we show that the size of the join is polynomial in the product of the sizes of the intermediary terms in the equational proof, cf. Theorem 22 and Proposition 21.

4) Finally, we study the complexity of proving confluence in the context of the λ-calculus. We obtain that the size of the join is at least exponential in the size of the peak. Hence, confluence is also not formalisable directly in Bounded Arithmetic.

Outline

The next section introduces basic notions and results. In Section 3 we establish the mentioned lower bound results for rewriting. Section 4 introduces technical notions that underly the methodology of our main results, to be presented in Section 5. In Section 6 we study lower and upper bounds on the complexity of Church-Rosser proofs. The lower bound of confluence proofs is established in Section 7. Section 8 discusses related works. Finally, in Section 9, we conclude and present future work.

2 Preliminaries

We assume (at least nodding) acquaintance with term rewriting [2, 12], however recall basic definitions and notations for ease of readability.

General. Let R be a binary relation. We write R^n for the n-fold iteration of R and R^* for the reflexive and transitive closure of R. Let $\mathcal{V}$ denote a countable infinite set of variables, and $\mathcal{F}$ a countable infinite set of function symbols (also called signature). The set of terms over $\mathcal{F}$ and $\mathcal{V}$ is denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$.

Let t be a term (over $\mathcal{F}$ and $\mathcal{V}$). A *position* p is a finite sequence of positive integers. Via positions, we uniquely identifying subterms of t, denoted as $t|_p$. We write $p \| q$ to indicate parallel positions, generalising the notions suitably to sets of positions. We write $\mathsf{Var}(t)$ to denote the set of variables occurring in t, ie. $\mathsf{Var}\, t = \{x \mid t|_p \text{ is a variable for some position } p\}$ and we write $\mathsf{rt}(t)$ to denote its root symbol. For example, for $\{x, y\} \subseteq \mathcal{V}$, $\mathsf{Var}(x + y) = \{x, y\}$ and $\mathsf{rt}(x + y) = +$. The *size* $|t|$ of term t is defined as the number of symbol occurrences in t, for example, $|x + y|= 3$. A term t is *linear* if every variable in t occurs only once.

Term Rewriting. A *rewrite rule* is a pair $l \to r$ of terms, such that (i) the left-hand side l is not a variable and (ii) $\mathsf{Var}(l) \supseteq \mathsf{Var}(r)$. A *term rewrite system* (TRS) over $\mathcal{F}$ is a finite set of rewrite rules $\mathcal{R}$; it will be denoted by the pair $(\mathcal{F}, \mathcal{R})$. If the signature $\mathcal{F}$ is clear from context, we simply denote a TRS by its set of rules $\mathcal{R}$. If $l \to r$ is a rewrite rule and σ a renaming, then the rule $l\sigma \to r\sigma$ is called a *variant* of $l \to r$. A TRS is said to be *variant-free*, if it does not contain rewrite rules that are variants. In the following we assume that TRSs are variant-free.

The rewrite relation based on $\mathcal{R}$ is denoted as $\to_{\mathcal{R}}$ and its transitve and reflexive closure as $\to_{\mathcal{R}}^*$. If the TRS is clear from context, we will simply write $\to$ and $\to^*$ respectively. Let s be a redex in term t. Here a *redex* is an occurrence of a term s that is an instance of the left-hand side l of a rule $l \to r \in \mathcal{R}$. We write $t \xrightarrow{s}_{\mathcal{R}} t'$ to indicate that redex s is contracted in the rewrite step. A term t over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is in *normal form* with respect to a TRS $\mathcal{R}$, if t does not contain any redex. We call a substitution σ *normalised* (with respect to $\mathcal{R}$), if all terms in the range of σ are in normal form. The *innermost rewrite relation* $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ of a TRS $\mathcal{R}$ is defined as follows: $s \xrightarrow{\mathrm{i}}_{\mathcal{R}} t$ if there exists a rewrite rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $s = C[l\sigma]$, $t = C[r\sigma]$, and all proper subterms of $l\sigma$ are normal forms of $\mathcal{R}$.

An *overlap* for $\mathcal{R}$ is a triple $\langle l \to r, p, l' \to r' \rangle$, such that (i) $l \to r$, $l' \to r'$ are rules in $\mathcal{R}$, whose variables are disjoint, (ii) p is not a variable position in l', (iii) l and $l'|_p$ are unifiable, (iv) if $p = \varepsilon$, then $l \to r$, $l' \to r'$ are not variants. A TRS is *left-linear* if the left-hand sides of all rules are linear. A TRS $\mathcal{R}$ without overlap is called *non-ambiguous*; a left-linear, non-ambiguous TRS is called *orthogonal*.

Let s and t be terms. Then an *(innermost) derivation* $D\colon s \to_{\mathcal{R}}^* t$ with respect to a TRS $\mathcal{R}$ is a finite sequence of (innermost) rewrite steps. Given an equational system $\mathcal{E}$, we can define, as usual, a TRS $\mathcal{R}$ such that

$$s =_{\mathcal{E}} t \quad \text{iff} \quad s \leftrightarrow_{\mathcal{R}}^* t \;.$$

(See [2, 12] for the straightforward construction.) A finite sequence of equational steps: $t_1 \leftrightarrow_{\mathcal{R}} t_2 \cdots \leftrightarrow_{\mathcal{R}} t_n$ is called an *equational proof.*

A term $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is *confluent*, if for all $t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $t \;{}^*\!\!\leftarrow s \to^* t'$, there exists a common reduct v, that is, $t \to^* v \;{}^*\!\!\leftarrow t'$. A TRS $(\mathcal{F}, \mathcal{R})$ is *confluent* if all terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ are confluent. We call the equational proof $t \;{}^*\!\!\leftarrow s \to^* t'$ a *peak*, the term v the *join* and the derivations $t \to^* v \;{}^*\!\!\leftarrow t'$ a *rewrite proof.* A peak is *local*, if it consists of one step each: $t \leftarrow s \to t'$. Confluence is equivalent to the *Church-Rosser property*, which states that for any equational proof $t \leftrightarrow^* t'$ there is a rewrite proof $t \to^* v \;{}^*\!\!\leftarrow t'$. A rewrite relation $\to$ has the *diamond property*, if any local peak over $\to$ can be joined immediately, that is, if $\leftarrow \cdot \to \subseteq \to \cdot \leftarrow$ holds.

Descendants and Residuals. Let $(\mathcal{F}, \mathcal{R})$ be a TRS and let L be a set of labels. The *labelled TRS* $(\mathcal{F}^L, \mathcal{R}^L)$ is defined by setting (i) $\mathcal{F}^L := \mathcal{F} \cup \{f^\ell \mid f \in \mathcal{F} \text{ and } \ell \in L\}$, (ii) the projection $\langle t \rangle$ of a term $t \in \mathcal{T}(\mathcal{F}^L, \mathcal{V})$ removes all labels, and (iii) $\mathcal{R}^L := \{l \to r \mid \langle l \rangle \to r \in \mathcal{R}\}$. The next proposition is from Terese [12, Proposition 4.2.3].

▶ **Proposition 1.** *Consider a left-linear TRS $(\mathcal{F}, \mathcal{R})$ and a set of labels L. Let $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and let s' be a labelled term such that $\langle s' \rangle = s$. Then each reduction step $s \to t$ can be lifted to a reduction step $s' \to t'$ in the labelled TRS $(\mathcal{F}^L, \mathcal{R}^L)$ such that $\langle t' \rangle = t$.*

In the following, we write $\mathcal{R}^L$ in short for the labelled TRS $(\mathcal{F}^L, \mathcal{R}^L)$, if the (labelled) signature is clear from context.

▶ **Definition 2.** *Let t be a term in a TRS $\mathcal{R}$, let s be a redex and let f be a function symbol occurring at position p in t, ie. $\mathsf{f} = \mathsf{rt}(t|_p)$. Let t_f denote the term that results from t by labelling this occurrence of f with label $\ell \in L$. Then the reduction step $t \xrightarrow{s} t'$ (contracting redex s) is lifted to a reduction step $t_\mathsf{f} \to t''$ in $\mathcal{R}^L$.*

The occurrences of f in t' that have label ℓ in t'' are the descendants *of the original symbol occurence of f in t. Conversely, the original f is called the* ancestor *of its descendants.*

The descendant/ancestor relation is extended to subterm occurrences via their root symbols. The descendant of a redex is called a *residual.* For a set of redexes S, we call the set of residuals of redexes in S simply the set of residuals of S. The descendant/ancestor relation naturally generalises to sequence of rewrite steps, that is, derivations. Note that the ancestor relation is unique, that is, for any derivation $D\colon s \to^* t$ the ancestor of a subterm u in t is given as a unique occurrence of a subterm u' in s, if it exists, cf. [12, Chapter 4].

Orthogonality. It is well-known that every orthogonal TRS is confluent, which can for example be verified by repeated applications of the Parallel Moves Lemma, cf. [12, Lemma 4.3.3].

▶ **Lemma 3** (Parallel Moves Lemma). *In an orthogonal TRS, let $t \to^* t_2$ be given. Let $t \xrightarrow{s} t_1$ be a one-step reduction by contraction of redex s. Then a common reduct t_3 of t_1 and t_2 can be found by contracting in t_2 of all residuals of redex s. Observe that all residuals will be pairwise disjoint.*

In order to prove the Parallel Moves Lemma, one makes use of the parallel rewriting relation, formalising the notion of contraction of pairwise disjoint redexes.

▶ **Definition 4.** *Let $\mathcal{R}$ be a TRS. We define the* parallel rewriting *relation $\Rightarrow_{\mathcal{R}}$ as follows*

1. *$x \Rightarrow_{\mathcal{R}} x$ for any variable x,*
2. *$f(\vec{s}) \Rightarrow_{\mathcal{R}} f(\vec{t})$ for any function symbol f, if for all i $s_i \Rightarrow_{\mathcal{R}} t_i$, and*
3. *$l\sigma \Rightarrow_{\mathcal{R}} r\sigma$, if $l \to r \in \mathcal{R}$ and σ a substitution.*

We often omit $\mathcal{R}$ and simply write $s \Rightarrow t$, if the TRS is clear from context.

Note that $\to_{\mathcal{R}} \subseteq \Rightarrow_{\mathcal{R}} \subseteq \to^*_{\mathcal{R}}$, in particular we have that $\to^*_{\mathcal{R}} = \Rightarrow_{\mathcal{R}}{}^*$. Making use of parallel rewriting, we can state the Parallel Moves Lemma succinctly as follows. A strengthening of the lemma has been stated and proven in [11].

▶ **Lemma 5.** *Parallel rewriting has the diamond property for every orthogonal TRS $\mathcal{R}$, that is, if $t \Leftarrow_{\mathcal{R}} s \Rightarrow_{\mathcal{R}} t'$, then there exists a join t'' such that $t' \Rightarrow_{\mathcal{R}} t'' \Leftarrow_{\mathcal{R}} t$.*

Let TRS $\mathcal{R}$ be fixed and let $s \Rightarrow t$ denote a paralel rewriting step with respect to $\mathcal{R}$. Suppose the (occurrences of) disjoint redexes contracted are collected in set S. Then we succinctly write $s \overset{S}{\Longrightarrow} t$. Due to the Parallel Moves Lemma, we obtain the following proposition, cf. [12, Proposition 4.5.6].

▶ **Proposition 6.** *Let $\mathcal{R}$ be an orthogonal TRS, and let $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Let S, T be sets of pairwise disjoint redexes in t and let $t \overset{S}{\Longrightarrow} t'$. Then the set of residuals of T in t' is unique, that is, independent of the order in which redexes in S are contracted.*

Proof. This is a direct consequence of the diamond property of $\Rightarrow$. Actually a stronger results holds. The single parallel rewriting step employed, is generalisable to a complete development step, without affecting the validity of the proposition, cf. [12, Proposition 4.5.6]. ◀

Based on Proposition 6 we denote with T/S the (unique) set of residuals of T in t' that are obtained by the parallel rewriting step $t \overset{S}{\Longrightarrow} t'$. With Lemma 3 we observe that T/S consists of pairwise disjoint redexes in t'.

Following the definition of the functions $\mathsf{cvs}_{\mathcal{R}}$ and $\mathsf{vs}_{\mathcal{R}}$ in [11], we define functions that compute the worst case of joining derivations based on peaks, resp. equation proofs, of a given size in the most effective way. Let $\|D\|$ denote the number of symbol occurrences in D.

▶ **Definition 7.** *Let $\mathcal{R}$ be an orthogonal term rewrite system. With $\mathsf{j}_{\mathcal{R}}(t, t')$ we denote the minimal size of a joining derivation of terms t and t', if it exist:*

$$\mathsf{j}_{\mathcal{R}}(t,t') = \begin{cases} \min\{\|D'\| \colon D' : t \to^*_{\mathcal{R}} \cdot {}^*\!\!\leftarrow_{\mathcal{R}} t'\} & \text{if } t \text{ and } t' \text{ have a joining derivation} \\ \infty & \text{otherwise} \end{cases}$$

The worst case join complexities *for confluence* Conf *and Church-Rosser* CR *are defined as*

$$\begin{aligned} \mathsf{Conf}(n) &= \max\{\mathsf{j}_{\mathcal{R}}(t,t') \colon \exists D; \|D\| = n,\ D : t \,{}^*_{\mathcal{R}}\!\!\leftarrow \cdot \to^*_{\mathcal{R}} t',\ \mathcal{R} \text{ orthogonal TRS}\} \\ \mathsf{CR}(n) &= \max\{\mathsf{j}_{\mathcal{R}}(t,t') \colon \exists D; \|D\| = n,\ D : t \leftrightarrow^*_{\mathcal{R}} t',\ \mathcal{R} \text{ orthogonal TRS}\}\ . \end{aligned}$$

In the following we will give some (worst-case) upper and (worst-case) lower bounds to those functions. Our main result will be a polynomial upper bound to Conf in Corollary 18. We also provide an exponential lower bound to CR in Corollary 20.

For the remainder of the paper, we restrict to *orthogonal* TRSs.

3 Lower Bounds for Confluence

For our lower bound considerations we use the following big-O facts, which follow easily from definitions.

▶ **Lemma 8.**
1. *If* $e_1(n) = \mathsf{O}(e(n))$ *and* $e_2(n) = \Omega(e(n))$ *then* $e_2(n) = \Omega(e_1(n))$.
2. *If* $e_1(n) = e(n)^{\mathsf{O}(1)}$ *and* $e_2(n) = e(n)^{\Omega(1)}$, *then* $e_2(n) = e_1(n)^{\Omega(1)}$.

We first give a linear lower bound to the number of steps for joining a peak in the size of the splitting sequence. We will provide a corresponding upper bound in Corollary 16.

▶ **Proposition 9.** *There is an orthogonal TRS $\mathcal{R}$ satisfying the following: Let $D_1\colon a \to^* b$ and $D_2\colon a \to^* c$ be derivations over $\mathcal{R}$, such that $b \to^k d$, and $c \to^l d$ holds for numbers k, l, and term d. Then $k + l = \Omega(\|D_1\| + \|D_2\|)$, that is, $k + l$ is at least linear in the number of symbols in D_1 and D_2 together.*

Proof. Consider the TRS $\mathcal{R}_1$ given by

$$\mathsf{f}(x) \to \mathsf{g}(x, x) \qquad \mathsf{a}(x) \to \mathsf{b}(x, x)\ . \tag{1}$$

We define meta term symbols via $A(T) := \mathsf{a}(T)$, $B(T) := \mathsf{b}(T, T)$, $F(T) := \mathsf{f}(T)$, $G(T) := \mathsf{g}(T, T)$. For a meta term symbol T let $T^{(n)}$ denote its n-fold iteration.

We define

$$S_n = F^{(n)}(A^{(n)}(0)) \qquad U_n = F^{(n)}(B^{(n)}(0))$$
$$V_n = G^{(n)}(A^{(n)}(0)) \qquad W_n = G^{(n)}(B^{(n)}(0))\ ,$$

and compute

$$|S_n| = \mathsf{O}(n) \qquad |U_n| = \mathsf{O}(2^n) \qquad |V_n| = \mathsf{O}(n2^n)\ .$$

Consider the following peak in $\mathcal{R}_1$, rewriting innermost redexes first.

$$D_1 :\quad S_n \xrightarrow{\mathsf{a}} F^{(n)}(A^{(n-1)}(B(0))) \xrightarrow{\mathsf{a}} F^{(n)}(A^{(n-2)}(B^{(2)}(0))) \xrightarrow{\mathsf{a}} \cdots \xrightarrow{\mathsf{a}} U_n$$
$$D_2 :\quad S_n \xrightarrow{\mathsf{f}} F^{(n-1)}(G(A^{(n)}(0))) \xrightarrow{\mathsf{f}} F^{(n-2)}(G^{(2)}(A^{(n)}(0))) \xrightarrow{\mathsf{f}} \cdots \xrightarrow{\mathsf{f}} V_n\ .$$

To discern ambiguity, we have identified the root symbol of the redex above the rewrite relation.

The size of each term in the first derivation is $\mathsf{O}(2^n)$, hence the overall size of D_1 is $\mathsf{O}(n2^n)$. The size of the k-th term in the second derivation is $\mathsf{O}(n2^k)$, so adding them up for $k \leqslant n$ gives a bound of $\mathsf{O}(n2^n)$ for the overall derivation length of D_2 as well. Hence $(\|D_1\| + \|D_2\|) = \mathsf{O}(n2^n)$.

The 'fastest' join of U_n and V_n is given by rewriting innermost redexes first:

$$U_n \xrightarrow{\mathsf{f}}^1 F^{(n-1)}(G(B^{(n)}(0))) \xrightarrow{\mathsf{f}}^1 F^{(n-2)}(G^{(2)}(B^{(n)}(0))) \xrightarrow{\mathsf{f}}^1 \cdots \xrightarrow{\mathsf{f}}^1 W_n$$
$$V_n \xrightarrow{\mathsf{a}}^{2^n} G^{(n)}(A^{(n-1)}(B(0))) \xrightarrow{\mathsf{a}}^{2^n} G^{(n)}(A^{(n-2)}(B^{(2)}(0))) \xrightarrow{\mathsf{a}}^{2^n} \cdots \xrightarrow{\mathsf{a}}^{2^n} W_n\ .$$

The length of the first derivation is n, and of the second $n2^n$, respectively.

Thus, a lower bound to the number of steps S_{join} of any derivations that join U_n and V_n is $n2^n$: $S_{\text{join}} = \Omega(n2^n)$. Together with $(\|D_1\| + \|D_2\|) = \mathsf{O}(n2^n)$ and Lemma 8.(1), we obtain $S_{\text{join}} = \Omega(\|D_1\| + \|D_2\|)$. Hence, S_{join} must be at least linear in the size of the derivations D_1 and D_2 constituting the peak. ◀

We also give a linear lower bound to the size of the join of the diamond in the product of the sizes of meet-able terms in a peak. The corresponding upper bound will be given in Corollary 15.

▶ **Proposition 10.** *There is an orthogonal TRS $\mathcal{R}$ satisfying the following: Let $b \ {}^*\!\!\leftarrow a \rightarrow^* c$ be a peak over $\mathcal{R}$ with consequent join d such that $b \rightarrow^* d$ and $c \rightarrow^* d$. Then $|d|= \Omega(|b|\cdot|c|)$, that is, the size $|d|$ of d is at least linear in $|b|\cdot|c|$.*

Proof. Fix n. We will basically follow the example from the proof of Proposition 9, with a slight modification to obtain optimal bounds.

With the notation from the proof of Proposition 9, expand TRS $\mathcal{R}_1$, cf. (1), with the rule $\mathsf{h} \rightarrow A^{(n)}(\mathsf{0})$. Let the resulting TRS be denoted as $\mathcal{R}_2$. We define

$$S'_n = F^{(n)}(\mathsf{h}) \qquad U_n = F^{(n)}(B^{(n)}(\mathsf{0}))$$
$$V'_n = G^{(n)}(\mathsf{h}) \qquad W_n = G^{(n)}(B^{(n)}(\mathsf{0})) ,$$

and compute

$$|U_n|= \mathsf{O}(2^n) \qquad |V'_n|= \mathsf{O}(2^n) \qquad |W_n|= \Omega(2^{2n}) .$$

Consider the following peak:

$$S'_n \xrightarrow{\mathsf{h}} F^{(n)}(A^{(n)}(\mathsf{0})) \xrightarrow{\mathsf{a}}{}^* U_n$$
$$S'_n \xrightarrow{\mathsf{f}} F^{(n-1)}(G(\mathsf{h})) \xrightarrow{\mathsf{f}} F^{(n-2)}(G^{(2)}(\mathsf{h})) \xrightarrow{\mathsf{f}}{}^* V'_n .$$

The 'smallest' join of U_n and V_n is given by rewriting only residuals:

$$U_n \xrightarrow{\mathsf{f}}{}^* W_n$$
$$V'_n \xrightarrow{\mathsf{h}}{}^* G^{(n)}(A^{(n)}(\mathsf{0})) \xrightarrow{\mathsf{a}}{}^* W_n .$$

We compute $|U_n|\cdot|V'_n|= \mathsf{O}(2^{2n})$. Together with $|W_n|= \Omega(2^{2n})$ and (1) we obtain $|W_n|= \Omega(|U_n|\cdot|V'_n|)$. Hence, the size of any join must be at least linear in the product of the sizes of U_n and V'_n. ◀

4 Injectivity

For the sequel, we fix an orthogonal TRS $\mathcal{R}$. Let $t' \ {}^*\!\!\leftarrow s \rightarrow^* t$ denote a peak over $\mathcal{R}$.

Consider the tiling diagramme in Figure 2 obtained by repeated applications of Lemma 5. We assume that $H_{0,\nu}$ denotes a singleton set of one redex in $s_{0,\nu}$, for $\nu = 0\dots,i{-}1$, and that $V_{\mu,0}$ denotes a singleton set of one redex in $s_{\mu,0}$, for $\mu = 0\dots,j{-}1$. Note that this implies $|H_{0,\nu}|= 1$ and $|V_{\mu,0}|= 1$. Further, we obtain

$$V_{\mu,\nu+1} = V_{\mu,\nu}/H_{\mu,\nu} \qquad H_{\mu+1,\nu} = H_{\mu,\nu}/V_{\mu,\nu} ,$$

as sets of residuals using Proposition 6. Moreover, using Proposition 6, we have that $H_{\mu,\nu}$ and $V_{\mu,\nu}$ are sets of pairwise disjoint redexes in $s_{\mu,\nu}$, for all $\mu = 0\dots,j{-}1$, $\nu = 0\dots,i{-}1$. Recall that a *redex* is an occurrence of a term t that is an instance of the left-hand side l of a rule $l \rightarrow r \in \mathcal{R}$.

$$\begin{tikzcd}
s_{0,0} = s \arrow[r, Rightarrow, "H_{0,0}"] \arrow[d, Rightarrow, "V_{0,0}"] & s_{0,1} \arrow[r, Rightarrow, "H_{0,1}"] \arrow[d, Rightarrow, "V_{0,1}"] & s_{0,2} \arrow[r, Rightarrow, "H_{0,2}"] \arrow[d, Rightarrow, "V_{0,2}"] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & s_{0,i-1} \arrow[r, Rightarrow, "H_{0,i-1}"] \arrow[d, Rightarrow, "V_{0,i-1}"] & s_{0,i} = t \arrow[d, Rightarrow, "V_{0,i}"] \\
s_{1,0} \arrow[r, Rightarrow, "H_{1,0}"] \arrow[d, Rightarrow, "V_{1,0}"] & s_{1,1} \arrow[r, Rightarrow, "H_{1,1}"] \arrow[d, Rightarrow, "V_{1,1}"] & s_{1,2} \arrow[r, Rightarrow, "H_{1,2}"] \arrow[d, Rightarrow, "V_{1,2}"] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & s_{1,i-1} \arrow[r, Rightarrow, "H_{1,i-1}"] \arrow[d, Rightarrow, "V_{1,i-1}"] & s_{1,i} \arrow[d, Rightarrow, "V_{1,i}"] \\
s_{2,0} \arrow[r, Rightarrow, "H_{2,0}"] \arrow[d, Rightarrow, "V_{2,0}"] & s_{2,1} \arrow[r, Rightarrow, "H_{2,1}"] \arrow[d, Rightarrow, "V_{2,1}"] & s_{2,2} \arrow[r, Rightarrow, "H_{2,2}"] \arrow[d, Rightarrow, "V_{2,2}"] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & s_{2,i-1} \arrow[r, Rightarrow, "H_{2,i-1}"] \arrow[d, Rightarrow, "V_{2,i-1}"] & s_{2,i} \arrow[d, Rightarrow, "V_{2,i}"] \\
\cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & \cdot \arrow[d, dotted, no head] \\
s_{j-1,0} \arrow[r, Rightarrow, "H_{j-1,0}"] \arrow[d, Rightarrow, "V_{j-1,0}"] & s_{j-1,1} \arrow[r, Rightarrow, "H_{j-1,1}"] \arrow[d, Rightarrow, "V_{j-1,1}"] & s_{j-1,2} \arrow[r, Rightarrow, "H_{j-1,2}"] \arrow[d, Rightarrow, "V_{j-1,2}"] & \cdot \arrow[r, dotted, no head] \arrow[d, dotted, no head] & s_{j-1,i-1} \arrow[r, Rightarrow, "H_{j-1,i-1}"] \arrow[d, Rightarrow, "V_{j-1,i-1}"] & s_{j-1,i} \arrow[d, Rightarrow, "V_{j-1,i}"] \\
s_{j,0} = t' \arrow[r, Rightarrow, "H_{j,0}"] & s_{j,1} \arrow[r, Rightarrow, "H_{j,1}"] & s_{j,2} \arrow[r, Rightarrow, "H_{j,2}"] & \cdot \arrow[r, dotted, no head] & s_{j,i-1} \arrow[r, Rightarrow, "H_{j,i-1}"] & s_{j,i} = t''
\end{tikzcd}$$

Figure 2 The tiling situation.

Generalised Ancestors

Given a sequence of rewrite steps

$$t \to_{s'} t' \to_{s''} t'' \to \ldots \to_{s^{(n-1)}} t^{(n-1)} \to_{s^{(n)}} t^{(n)}$$

we generalise the notion of ancestor to trace any subterm in the sequence back to t – we denote this *generalised ancestor*, or short *g.-ancestor*.

Ancestors are also g.-ancestors. Consider a subterm u_j in $t^{(j)}$, and its ancestors u_{j-1} in $t^{(j-1)}$, etc., until u_i in $t^{(i)}$ cannot be extended any further. Let f denote the root symbol of u_i in $t^{(i)}$. As f does not have an ancestor in $t^{(i-1)}$, we must be in the following situation: There exist a context $C[*]$, substitution σ, and rule $l \to r$ in $\mathcal{R}$, such that $t^{(i-1)} = C[l\sigma]$, $t^{(i)} \equiv C[r\sigma]$, and f occurs in r. We now define the *generalised ancestor* of f in $t^{(i)}$ as the root symbol of l in $C[l\sigma] = t^{(i-1)}$. Continue until t is reached.

▶ **Proposition 11.** *In the tiling diagramme in Figure 2, the generalised ancestors of any symbol occurrence are unique, that is, independent of the path chosen to compute them.*

Proof. Arguing inductively, it suffices to prove the statement for a single square:

$$\begin{tikzcd}
s_{\mu,\nu} \arrow[r, Rightarrow, "H_{\mu,\nu}"] \arrow[d, Rightarrow, "V_{\mu,\nu}"] & s_{\mu,\nu+1} \arrow[d, Rightarrow, "V_{\mu,\nu+1}"] \\
s_{\mu+1,\nu} \arrow[r, Rightarrow, "H_{\mu+1,\nu}"] & s_{\mu+1,\nu+1} .
\end{tikzcd}$$

Recall that using Proposition 6, we have that $H_{\mu,\nu}$ and $V_{\mu,\nu}$ are sets of disjoint redexes in $s_{\mu,\nu}$, for all $\mu = 0 \ldots, j-1$, $\nu = 0 \ldots, i-1$. Thus, in proof of the claim, we can assume without loss of generality that $|H_{\mu,\nu}| = |V_{\mu,\nu}| = 1$.

Let u be a subterm of $s_{\mu+1,\nu+1}$. First, suppose u has an *ancestor* in $s_{\mu,\nu}$. Then, this ancestor is unique, as mentioned above.

Second, suppose u has only *generalised ancestors* in $s_{\mu,\nu}$. Then, we distinguish cases on the relative positioning of redexes in $H_{\mu,\nu}$ and $V_{\mu,\nu}$, respectively. Recall, that by assumption the redexes in $H_{\mu,\nu}$ and $V_{\mu,\nu}$ are pairwise disjoint.

Case. Suppose $H_{\mu,\nu} \| V_{\mu,\nu}$, that is, the redexes in $H_{\mu,\nu} \cup V_{\mu,\nu}$ are all pairwise disjoint. Then the claim is obvious.

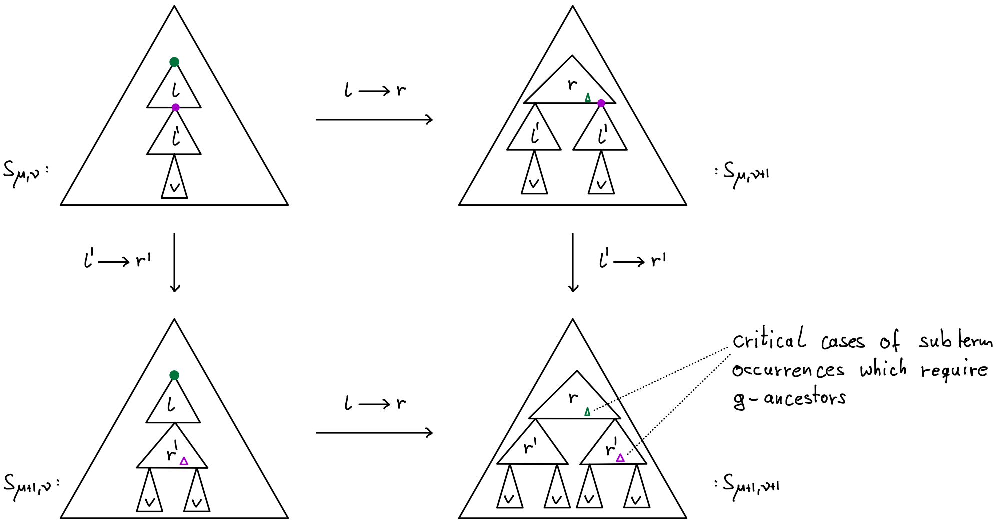


Figure 3 Critical cases where generalised ancestors occur.

Case. Suppose there exists rules $l \to r, l' \to r' \in \mathcal{R}$, and substitutions σ, σ' such that $l\sigma \in H_{\mu,\nu}$ and $l'\sigma' \in V_{\mu,\nu}$. Further $l'\sigma' \lhd l\sigma$. (The case $l\sigma = l'\sigma$ is trivial, because we must have $(l \to r) = (l' \to r')$ due to orthogonality of $\mathcal{R}$.) As u does not have an ancestor in $s_{\mu,\nu}$, $\mathsf{rt}(u)$ either occurs in r or in r'. The situation of this case is depicted in Figure 3.

Wlog. $\mathsf{rt}(u)$ occurs in r' and thus u occurs in any of the occurrences of $r'\sigma'$ in $s_{\mu+1,\nu+1}$. By assumption on $l\sigma$ and $l'\sigma'$, u has an ancestor in $s_{\mu+1,\nu}$ and a generalised ancestor in $s_{\mu,\nu+1}$, which are both unique and consequently their join in $s_{\mu,\nu}$ is unique, too. ◀

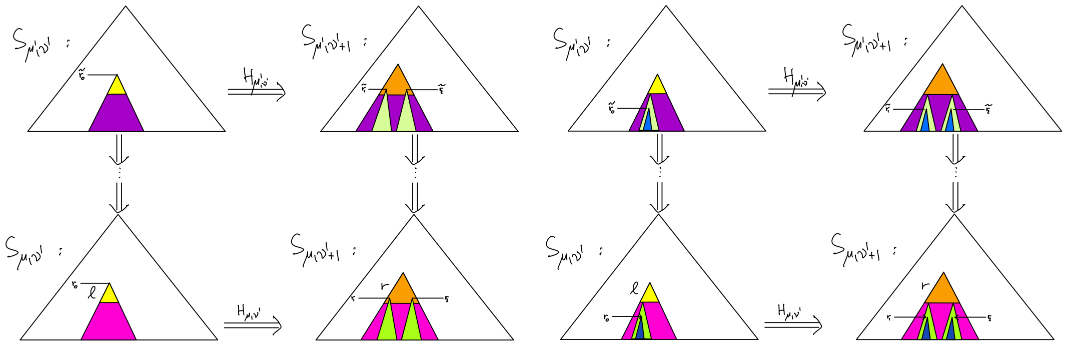
▶ **Definition 12.** *Let the tiling diagramme in Figure 2 be given, and let $\mu < j$, $\nu < i$. Let f be a function symbol occurrence in $s_{\mu,\nu}$, and let $\mu' \leqslant \mu$, $\nu' \leqslant \nu$. We define $\mathrm{ga}^{\mu,\nu}_{\mu',\nu'}(\mathsf{f})$ as the g.-ancestor of f in $s_{\mu',\nu'}$.*

We now formulate the main result of this section.

▶ **Lemma 13.** *Let the tiling diagramme in Figure 2 be given, and let $\mu < j$, $\nu < i$, and $\mu' \leqslant \mu$, $\nu' \leqslant \nu$. The mapping of function symbol occurrences f in $s_{\mu,\nu}$ to the pair $(\mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{f}), \mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{f}))$ is an injection.*

Proof. This claim can be proven by induction on $\nu - \nu'$. The case for $\nu = \nu'$ is obvious, because $\mathrm{ga}^{\mu,\nu}_{\mu,\nu}$ is the identity, which is injective.

For the induction step from $\nu' + 1$ to ν' we can assume by induction hypothesis that the claim is true for $(\mu', \nu' + 1)$. We then show the claim for (μ', ν'), depicted as follows.



(a) Case A. **(b)** Case B.

Figure 4 Cases A and B in proof of Lemma 13.

$$\begin{tikzcd}
s_{\mu',\nu'} \arrow[r, Rightarrow] \arrow[d, Rightarrow, "\vdots"] & s_{\mu',\nu'+1} \arrow[r, Rightarrow] \arrow[d, Rightarrow, "\vdots"] & \cdots \arrow[r, Rightarrow] & s_{\mu',\nu} \arrow[d, Rightarrow, "\vdots"] \arrow[r, phantom, "\ni"] & \mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{f}) & \mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{g}) \\
s_{\mu,\nu'} \arrow[r, Rightarrow] \arrow[d, phantom, sloped, "\ni"] & s_{\mu,\nu'+1} \arrow[r, Rightarrow] \arrow[d, phantom, sloped, "\ni"] & \cdots \arrow[r, Rightarrow] & s_{\mu,\nu} \arrow[dr, phantom, "\ni"] & & \\
\mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{f}) & \mathrm{ga}^{\mu,\nu}_{\mu,\nu'+1}(\mathsf{f}) \arrow[l, mapsto] & & & \mathsf{f} \arrow[lll, mapsto] \arrow[uu, mapsto] & \\
\mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{g}) & \mathrm{ga}^{\mu,\nu}_{\mu,\nu'+1}(\mathsf{g}) \arrow[l, mapsto] & & & & \mathsf{g} \arrow[llll, mapsto] \arrow[uuu, mapsto]
\end{tikzcd}$$

For sake of contradiction assume the claim is wrong for (μ',ν'). That is, there are f,g occurring in $s_{\mu,\nu}$ with f,g different symbol occurrences, such that $\mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{f}) = \mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{g})$ and $\mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{f}) = \mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{g})$. By i.h. we must have $\mathrm{ga}^{\mu,\nu}_{\mu,\nu'+1}(\mathsf{f}) \neq \mathrm{ga}^{\mu,\nu}_{\mu,\nu'+1}(\mathsf{g})$. Let $r_1 = \mathrm{ga}^{\mu,\nu}_{\mu,\nu'+1}(\mathsf{f})$, $r_2 = \mathrm{ga}^{\mu,\nu}_{\mu,\nu'+1}(\mathsf{g})$, and $r_0 = \mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{f}) = \mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{g})$. This situation is depicted below.

$$\begin{tikzcd}
s_{\mu',\nu'} \arrow[r, Rightarrow, "H_{\mu',\nu'}"] \arrow[d, Rightarrow, "\vdots"] & s_{\mu',\nu'+1} \arrow[r, Rightarrow] \arrow[d, Rightarrow, "\vdots"] & \cdots \arrow[r, Rightarrow] & s_{\mu',\nu} \arrow[d, Rightarrow, "\vdots"] \arrow[r, phantom, "\ni"] & \hat{r} \\
s_{\mu,\nu'} \arrow[r, Rightarrow, "H_{\mu,\nu'}"] \arrow[dd, phantom, sloped, "\ni"] & s_{\mu,\nu'+1} \arrow[r, Rightarrow] \arrow[d, phantom, sloped, "\ni"] & \cdots \arrow[r, Rightarrow] & s_{\mu,\nu} \arrow[dr, phantom, "\ni"] & \\
 & r_1 \arrow[dl, mapsto] & & & \mathsf{f} \arrow[lll, mapsto] \arrow[uu, mapsto] \\
r_0 & & & & \\
 & r_2 \arrow[ul, mapsto] & & & \mathsf{g} \arrow[lll, mapsto] \arrow[uuuu, mapsto]
\end{tikzcd}$$

We must be in the following situation: There are rule $l \to r$ in $\mathcal{R}$, substitution ρ, terms $u_1,\ldots,u_k$, context $C[*_1,\ldots,*_k]$, such that $H_{\mu,\nu'} = \{u_1,\ldots,u_k\}$, $u_1 = l\rho$, $s_{\mu,\nu'} = C[u_1,\ldots,u_k]$, and r_1 and r_2 occur in $r\rho$ in $s_{\mu,\nu'+1} = C[r\rho,\ldots]$, and either

- the roots of r_1 and r_2 occur already in r in $C[r\rho,\ldots]$, hence their joint g.-ancestor r_0 is the root of l in $C[l\rho,u_2,\ldots,u_k]$, see Figure 4a;
- or we have a variable x occuring in l which occurs multiple times in r, e.g. as $C_r[*_1,*_2]$ with $r = C[x,x]$ – hence $r\rho = C_r\rho[x\rho,x\rho]$ – and r_1 occurs in the first $x\rho$, r_2 occurs in the second $x\rho$, and their joint ancestor r_0 occurs in $x\rho$ in $l\rho$ in $s_{\mu,\nu'}$, see Figure 4b.

Let $\hat{r} = \mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{f}) = \mathrm{ga}^{\mu,\nu}_{\mu',\nu}(\mathsf{g})$ be the g.-ancestor of f and g in $s_{\mu',\nu}$. $H_{\mu,\nu'}$ are residuals of $H_{\mu',\nu'}$, hence the ancestors $\tilde{r}_0$ of r_0 in $s_{\mu',\nu'}$ and $\tilde{r}_1,\tilde{r}_2$ of r_1,r_2 in $s_{\mu',\nu'+1}$ will occur in $l\rho'$ and $r\rho'$ for some ρ'. In particular in A), the roots of $\tilde{r}_1$ and $\tilde{r}_2$ are in r, and $\tilde{r}_0$ is at the root of l. In case B) we have that $r\rho' = C_r\rho'[x\rho',x\rho']$ with $\tilde{r}_1$ occuring in 1st and $\tilde{r}_2$ in 2nd of $x\rho'$.

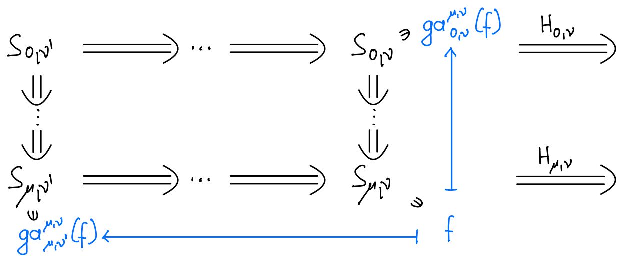
In both cases we have that $\tilde{r}_1$ and $\tilde{r}_2$ are two distinct g.-ancestors of f and g in $s_{\mu',\nu'+1}$, resp., by following from $s_{\mu,\nu}$ the derivation first to $s_{\mu,\nu'+1}$ and then to $s_{\mu',\nu'+1}$. However, by following from $s_{\mu,\nu}$ the derivation to $s_{\mu',\nu}$, f and g have a joint ancestor $\hat{r}$, hence can only have one joint ancestor in $s_{\mu',\nu'+1}$ when following the derivation from $s_{\mu',\nu}$ to $s_{\mu',\nu'+1}$ to the left. This contradicts Proposition 11 that g.-ancestors are unique. ◀

▶ **Lemma 14.** *Let the tiling diagramme in Figure 2 be given, and let $\mu < j$, $\nu < i$.*

Assuming $|H_{0,\nu}|= 1$, the mapping of each redex in $H_{\mu,\nu}$ to their generalised ancestors in $s_{\mu,\nu'}$ for $\nu' < \nu$ is an injection.

Similar for $V_{\mu,\nu}$: Assuming $|V_{\mu,0}|= 1$, the mapping of each redex in $V_{\mu,\nu}$ to their generalised ancestors in $s_{\mu',\nu}$ for $\mu' < \mu$ is an injection.

Proof. We only consider the first assertion, the second is dual. Ie., we are in the following situation.



Let s be a term, H a set of redexes in s, and f a function symbol occurrence in s. We succinctly write $\mathsf{f} \in H$ to indicate that f is the occurrence of the root symbol of some redex in H.

By Lemma 13 we have that the mapping

$$\mathsf{f} \in H_{\mu,\nu} \mapsto (\mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{f}), \mathrm{ga}^{\mu,\nu}_{0,\nu}(\mathsf{f}))$$

is an injection. By assumption we have that $|H_{0,\nu}|= 1$, hence $H_{0,\nu} = \{\hat{r}\}$ for some $\hat{r}$. This implies that $\mathrm{ga}^{\mu,\nu}_{0,\nu}(\mathsf{f}) = \hat{r}$ for all $\mathsf{f} \in H_{\mu,\nu}$. Hence

$$\mathsf{f} \in H_{\mu,\nu} \mapsto \mathrm{ga}^{\mu,\nu}_{\mu,\nu'}(\mathsf{f})$$

must be injective. ◀

5 Upper Bounds on Confluence

In this short section, we state and prove our main result that the size, that is, the number of symbols, of a rewrite proof is polynomial in the size of the peak, cf. Figure 1. First, we draw two easy corollaries from Lemma 13 and Lemma 14, respectively.

▶ **Corollary 15.** *Consider the tiling diagramme in Figure 2. The size of the join t'' is bounded by the product of the sizes of t and t':*

$$|t''| \quad \leqslant \quad |t|{\cdot}|t'| \ .$$

Proof. This is a direct consequence of Lemma 13. ◀

▶ **Corollary 16.** *Consider the tiling diagramme in Figure 2, assuming $|H_{0,\nu}|= 1$ and $|V_{\mu,0}|= 1$. In this situation, the number of (sequential) reduction steps needed to join t and t' via t'', is bounded by the square of the size of the initial sequence. More precisely:*

$$\sum_{\nu=0}^{i-1}|H_{j,\nu}| \quad + \quad \sum_{\mu=0}^{j-1}|V_{\mu,i}| \;\leqslant\; i\cdot|t'|+j\cdot|t| \;\leqslant\; \Big(\sum_{\mu=0}^{j}|s_{\mu,0}|+\sum_{\nu=1}^{i}|s_{0,\nu}|\Big)^2 .$$

Proof. For the first inequality, observe that by Lemma 14, we have that $|H_{j,\nu}|\leq |s_{j,0}|$ for $\nu < i$ and $|V_{\mu,i}|\leq |s_{0,i}|$ for $\mu < j$. Thus, $|H_{j,\nu}|\leq |t'|$ and $|V_{\mu,i}|\leq |t|$ by definition. Then, the second inequality follows by elementary calculations. Finally, observe that if the set of redex S is disjoint then $\overset{S}{\Longrightarrow} \subseteq \to^S$, from which the claim follows. ◀

Now, our main result follows with ease.

▶ **Theorem 17.** *Let $\mathcal{R}$ be an orthogonal TRS and assume the existence of a peak $D\colon t' \;{}^*\!\!\leftarrow s \to^* t$. Then there exists a rewriting proof $D'\colon t' \to^* t'' \;{}^*\!\!\leftarrow t$ whose size is polynomially bounded in the size of D. In fact, the size of D' is biquadratic in the size of D.*

Proof. This is a consequence of Corollaries 15 and 16. Let D' be the joining derivation given by the tiling diagram in Figure 2, where $s_{0,0} = s$, $s_{0,\nu}$ is the ν-th term in $s \to^i t$, and $s_{\mu,0}$ the μ-th term in $s \to^j t'$. Employing the notation of that figure, we obtain

$$\|D\| \;=\; \sum_{\mu=0}^{j}|s_{\mu,0}|+\sum_{\nu=1}^{i}|s_{0,\nu}| .$$

Recall that $\|D\|$ denotes the number of symbol occurrences in D. Due to Corollary 15, we have, for each μ, ν ($0 \leqslant \mu \leqslant j$, $0 \leqslant \nu \leqslant i$), that

$$|s_{\mu,\nu}| \;\leqslant\; |s_{\mu,0}|\cdot|s_{0,\nu}| \;\leqslant\; \|D\|^2 . \tag{2}$$

Moreover, due to Corollary 16, the number of joining steps in D' is bounded by $\|D\|^2$:

$$\text{number of joining steps} \quad\leqslant\quad \sum_{\nu=0}^{i-1}|H_{j,\nu}| + \sum_{\mu=0}^{j-1}|V_{\mu,i}| \quad\leq\quad \|D\|^2 . \tag{3}$$

Combining (2) and (3), we conclude that $\|D'\| \leqslant \|D\|^4$. ◀

▶ **Corollary 18.** Conf *is biquadratically bounded, i.e.* $\mathsf{Conf}(n) = \mathsf{O}(n^4)$.

A closer inspection of the example in the proof of Proposition 10 establishes a cubic lower bound, i.e. $\mathsf{Conf}(n) = \Omega(n^3)$.

6 Lower and Upper Bounds for the Church-Rosser Property

In the case of the Church-Rosser property, we first give an exponential lower bound to the size of the join, which in particular gives an exponential lower bound to the join complexity CR.

▶ **Theorem 19.** *There is an orthogonal TRS $\mathcal{R}$ satisfying the following: Let D be a derivation of $a \leftrightarrow^* b$ over $\mathcal{R}$, such that $a \to^* c$ and $b \to^* c$ holds, then $|c|$ is exponential in $\|D\|$ in general, i.e. $|c|= 2^{\|D\|^{\Omega(1)}}$.*

Proof. Consider the TRS $\mathcal{R}_3$ given by

$$\mathsf{f}_i(x) \to \mathsf{a}_i(x,x) \quad \mathsf{g}_i(x) \to \mathsf{a}_i(x,x) \qquad (i = 1,\dots,k)\ . \tag{4}$$

We define meta term symbols via $A_i(T) := \mathsf{a}_i(T,T)$, define

$$S_i^k = \mathsf{g}_1(\dots \mathsf{g}_{i-1}(\mathsf{g}_i(\mathsf{f}_{i+1}(\dots \mathsf{f}_k(\mathsf{0})\dots)))\dots) \qquad U^k = A_1(\dots A_k(\mathsf{0})\dots)$$
$$T_i^k = \mathsf{g}_1(\dots \mathsf{g}_{i-1}(A_i(\mathsf{f}_{i+1}(\dots \mathsf{f}_k(\mathsf{0})\dots)))\dots)\ ,$$

and compute

$$|S_i^k| = \mathsf{O}(k) \qquad |T_i^k| = \mathsf{O}(k) \qquad S_i^k \xrightarrow{g_i} T_i^k \qquad S_i^k \xrightarrow{f_{i+1}} T_{i+1}^k\ .$$

Consider the following derivation:

$$D \quad := \quad T_1^k \leftarrow S_1^k \to T_2^k \leftarrow S_2^k \to T_3^k \ \dots\ T_{k-1}^k \leftarrow S_{k-1}^k \to T_k^k$$

The unique Church-Rosser join is given by $T_i^k \to^* U$ for all $i = 1,\dots,k$. From now on we drop the superscript k.

Let $S_D = \|D\|$ and $S_U = |U|$. We compute $S_D = \mathsf{O}(n^2)$ and $S_U = \Omega(2^n)$. Thus $S_D \leqslant ck^2$ for some $c > 0$, hence $k \geqslant \sqrt{\frac{1}{c}S_D} \geqslant {S_D}^{\epsilon}$ for small $\epsilon > 0$. Thus $S_U \geqslant 2^k \geqslant 2^{{S_D}^{\epsilon}}$. ◀

▶ **Corollary 20.** $\mathsf{CR}(n)$ *is exponential in* n*, i.e.* $\mathsf{CR}(n) = 2^{n^{\Omega(1)}}$.

Inspecting our upper bounds, Corollaries 15 and 16, establishes that this bound is optimal up to the degree, i.e. $\mathsf{CR}(n) = 2^{n^{\mathsf{O}(1)}}$.

We now show that the size of the join in the case of Church-Rosser is polynomially related to the product of the sizes of the terms in the starting derivation. We first state the lower bound.

▶ **Proposition 21.** *There is an orthogonal TRS* $\mathcal{R}$ *satisfying the following: Let* $a_1 \leftrightarrow a_2 \leftrightarrow \dots \leftrightarrow a_k$ *be a derivation over* $\mathcal{R}$ *such that* $a_1 \to^* b$ *and* $a_k \to^* b$ *for some* b*. Then* $|b|$ *is polynomial in* $|a_1|{\cdot}|a_2|{\cdot}\dots{\cdot}|a_k|$ *in general, i.e.* $|b|= (|a_1|{\cdot}|a_2|{\cdot}\dots{\cdot}|a_k|)^{\Omega(1)}$.

Proof. We modify the TRS from the previous proof so that the starting terms are of constant size: Expand the TRS from the proof of Theorem 19 by

$$\bar{\mathsf{f}}_i^k \to \mathsf{f}_i(\bar{\mathsf{f}}_{i+1}^k) \quad \bar{\mathsf{g}}_i^k(x) \to \bar{\mathsf{g}}_{i-1}^k(\mathsf{g}_i(x)) \qquad (i = 1,\dots,k) \tag{5}$$

where $\bar{\mathsf{f}}_{k+1}^k$ represents 0. We define

$$\bar{S}_i^k = \bar{\mathsf{g}}_i^k(\bar{\mathsf{f}}_{i+1}^k) \qquad \bar{T}_i^k = \bar{\mathsf{g}}_{i-1}^k(A_i(\bar{\mathsf{f}}_{i+1}^k))\ ,$$

and compute

$$|\bar{S}_i^k| = \mathsf{O}(1) \qquad \bar{S}_i^k = \bar{\mathsf{g}}_i^k(\bar{\mathsf{f}}_{i+1}^k) \xrightarrow{\bar{\mathsf{g}}_i^k} \bar{\mathsf{g}}_{i-1}^k(\mathsf{g}_i(\bar{\mathsf{f}}_{i+1}^k)) \xrightarrow{\mathsf{g}_i} \bar{\mathsf{g}}_{i-1}^k(A_i(\bar{\mathsf{f}}_{i+1}^k)) \ = \ \bar{T}_i^k$$
$$|\bar{T}_i^k| = \mathsf{O}(1) \qquad \bar{S}_i^k = \bar{\mathsf{g}}_i^k(\bar{\mathsf{f}}_{i+1}^k) \xrightarrow{\bar{\mathsf{f}}_{i+1}^k} \bar{\mathsf{g}}_i^k(\mathsf{f}_{i+1}(\bar{\mathsf{f}}_{i+2}^k)) \xrightarrow{\mathsf{f}_{i+1}} \bar{\mathsf{g}}_i^k(A_{i+1}(\bar{\mathsf{f}}_{i+2}^k)) \ = \ \bar{T}_{i+1}^k\ .$$

From now on we will drop the superscript k. Consider the following derivation:

$$\bar{D} \quad := \quad \bar{T}_1 \leftarrow^2 \bar{S}_1 \to^2 \bar{T}_2 \leftarrow^2 \bar{S}_2 \to^2 \bar{T}_3 \ \dots\ \bar{T}_{k-1} \leftarrow^2 \bar{S}_{k-1} \to^2 \bar{T}_k\ .$$

The unique Church-Rosser join is again given by $\bar{T}_i \to^* r$ for all $i = 1,\dots,k$.

Let $\bar{S} = \Pi_{t\in\bar{D}}|t|$ and $S_r = |r|$. We compute $\bar{S} = c^{2k}$ for some $c = \mathsf{O}(1)$ which is an upper bound on the size of terms occurring in $\bar{D}$. Hence $\bar{S} = (2^k)^{\mathsf{O}(1)}$. We also have $S_r = (2^k)^{\Omega(1)}$. Hence $S_r = \bar{S}^{\Omega(1)}$ using Lemma 8(2). ◀

We also have a corresponding upper bound.

▶ **Theorem 22.** *Let $\mathcal{R}$ be an orthogonal TRS. Given a derivation $a_1 \leftrightarrow a_2 \leftrightarrow \cdots \leftrightarrow a_k$ over $\mathcal{R}$, there is a join $a_1 \to^* b \;{}^*\!\!\leftarrow a_k$ for some b, such that $|b|$ is bounded by $|a_1|{\cdot}|a_2|{\cdot}\cdots{\cdot}|a_k|$.*

Proof. The upper bound is obtained by induction on k using the related upper bound for confluence, Corollary 15: Assume $a_1 \leftrightarrow \cdots \leftrightarrow a_k \leftrightarrow a_{k+1}$. By induction hypothesis there are some b, $a_1 \to^* b$ and $a_k \to^* b$ such that $|b|$ is bounded by $|a_1|{\cdot}|a_2|{\cdot}\cdots{\cdot}|a_k|$. If $a_{k+1} \to a_k$ then b is also the join for a_1 and a_{k+1} and we are already done. Otherwise, $a_k \to a_{k+1}$. Using that $a_k \to^* b$, we can join this peak with some c of size $\leqslant |b|{\cdot}|a_{k+1}|$ using Corollary 15. Thus $|c| \leqslant |b|{\cdot}|a_{k+1}| \leqslant |a_1|{\cdot}|a_2|{\cdot}\cdots{\cdot}|a_k|{\cdot}|a_{k+1}|$. ◀

7 A Lower Bound for the Lambda Calculus

For this section, we assume(at least nodding) acquaintance with the (untyped) λ-calculus [3, 4]. While we refrain from re-stating (hopefully) well-known notions, the result should be easy to understand.

We show that for confluence in λ-calculus, the size of the join is exponential in the product of the sizes of the starting terms in general.

▶ **Proposition 23.** *Given a peak $D\colon b \leftarrow_\lambda^* a \to_\lambda^* c$, and a joining derivation $b \to_\lambda^* d \leftarrow_\lambda^* c$. Then $|d|$ is exponential in $\|D\|$ as well as in $|b|{\cdot}|c|$ in general: $|d| = 2^{\|D\|^{\Omega(1)}}$ and $|d| = 2^{(|b|\cdot|c|)^{\Omega(1)}}$.*

Proof. Let f, g, h, x, y be variables. Let $A := \lambda x.((\lambda y.hyy)(gx))$ and $B := \lambda x.(h(gx)(gx))$. We have $A \xrightarrow{\lambda y}_\lambda B$, $|A|= \Theta(1)$, $|B|= \Theta(1)$.

Define terms T^k, U^k, V^k, W^k as follows: Let $T^0 = U^0 = V^0 = W^0 = f$, and inductively

$$T^{k+1} = (A\,T^k), \quad U^{k+1} = (B\,U^k), \quad V^{k+1} = (\lambda y.hyy)(gV^k), \quad W^{k+1} = h(gW^k)(gW^k) \ .$$

Then $|T^k|= \mathsf{O}(k)$, $|U^k|= \mathsf{O}(k)$, $|V^k|= \mathsf{O}(k)$, and $|W^k|= \Omega(2^k)$. We have

$$T^k \xrightarrow{\lambda y}{}_\lambda^k U^k \qquad T^k \xrightarrow{\lambda x}{}_\lambda^k V^k \qquad U^k \xrightarrow{\lambda x}{}_\lambda^k W^k \qquad V^k \xrightarrow{\lambda y}{}_\lambda^k W^k$$

by induction on k. Let D be $U^k \leftarrow_\lambda^* T^k \to_\lambda^* V^k$. Then $\|D\| = \mathsf{O}(k^2)$, hence $k \geqslant (\|D\|)^\epsilon$ for some $\epsilon > 0$, hence $|W^k| = \Omega(2^k) = \Omega(2^{(\|D\|)^\epsilon})$. As $|b|{\cdot}|d|= \mathsf{O}(k^2)$ as well, the same calculation applies in this case as well. ◀

8 Related Works

Ketema and Grue Simonsen have studied similar properties in [11]. For a given TRS $\mathcal{R}$, they define functions $\mathsf{cvs}_\mathcal{R}$ and $\mathsf{vs}_\mathcal{R}$, estimating the least number of reduction steps necessary in a rewrite proof, assuming an equational proof or a peak, respectively. More precisely, $\mathsf{cvs}_\mathcal{R}(m, n)$ denotes the least number of reduction steps required to complete a rewrite proof, given an equational proof involving at most n steps between two terms t, t' of size at most m. Likewise, $\mathsf{vs}_\mathcal{R}(m, n)$ denotes the least number of reduction steps in a rewrite proof, given a peak $t \;{}^*\!\!\leftarrow s \to^* t'$, where the size of s is at most m and the reduction lengths are at most of size n. For orthogonal TRSs $\mathcal{R}$ they obtain optimal exponential upper bound on $\mathsf{vs}_\mathcal{R}$ and an upper bound on $\mathsf{cvs}_\mathcal{R}$ that belongs to the 4^{th}-level of the Grzegorczyk hierarchy. I.e. the upper bound on $\mathsf{cvs}_\mathcal{R}$ is at least non-elementary. Wrt. the λ-calculus, confluence already requires an non-elementary upper bound. In subsequent work, Fujita proved that for the λ-calculus $\mathsf{cvs}_\mathcal{R}$ is upper bounded in the 4^{th}-level of the Grzegorczyk hierarchy, cf. [10]. Only optimality of the bound on $\mathsf{vs}_\mathcal{R}$ for orthogonal rewrite systems has been established.

We emphasise that these results are orthogonal to our contributions, as we make use of a different notion of proof complexity: the number of symbols, rather than the number of reduction steps. While this measure is natural in the context of rewriting (or even the λ-calculus), it is less so in the context of computational complexity, from our point of view. In short, for orthogonal TRSs, this change allows us to provide (optimal) polynomial upper bounds on confluence proofs and (optimal) exponential upper bounds on Church-Rosser proofs, while we establish an exponential lower bound on confluence proofs for the λ-calculus. Note that our changed notion of size not only allows tractable upper bounds, but also differentiates precisely between the expressivity of (first-order) term rewrite systems and (higher-order) λ-calculus, a difference that got somewhat blurred in related works.

To the best of our knowledge, confluence or Church-Rosser properties in term-rewriting have not been studied in general in Bounded Arithmetic (though they have been used as tools in the analysis of related artefacts, as in work by Das [9]). The closest we are aware of are the results by the first author [5] that formalises a restricted and very involved property the resembles elements of Church-Rosser, and which are used to prove the consistency of any equational theory that exclusively is based on recursive defining equations, in a weak theory of bounded arithmetic. These results were improved by Yamagata [13] by also allowing rules for substituting terms into equations in the equational reasoning while proving consistency in a weak theory of bounded arithmetic. However, Yamagata formalised ideas from programming semantics with no connection to rewriting.

9 Conclusion

In this paper, we have investigated two well-studied properties of rewriting and the λ-calculus, namely confluence and the Church-Rosser property, through the lens of proof complexity. In particular, for orthogonal TRSs, we have shown that the shortest rewrite proof obtained in a confluence argument is polynomially related to the size of the peak.

This is in contrast to earlier results on upper bounds on the size of confluence and Church-Rosser proofs that used the number of steps as size measure. While this measure is natural in the context of rewriting (or even the λ-calculus), it is less so in the context of computational complexity, from our point of view. We emphasise that our changed notion of size not only allows tractable upper bounds, but also differentiates precisely between the expressivity of (first-order) term rewrite systems and (higher-order) λ-calculus, a difference, that got somewhat blurred in related works.

We have established preliminary steps towards our motivation to study consistency proofs in weak theories of arithmetic through the lens of rewriting technologies. In future work we want to expand this direction. It seems natural to us to employ techniques from graph rewriting [12, Chapter 13] (see also [1]) to overcome the exponential lower bound on the size of the join that we have established for the Church-Rosser property. Due to the succinct encoding of multiple occurrences in graph rewriting it could be possible to allow an alternative encoding of the join and of the rewrite proof, altogether. The latter could potentially give rise to a polynomial encoding. These investigations are left to future work.

References

1 Martin Avanzini and Georg Moser. Complexity of Acyclic Term Graph Rewriting. In *Prof. 1st FSCD*, volume 52 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.FSCD.2016.10.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

3 Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

4 Henk Barendregt and Giulio Manzonetto. *A Lambda Calculus Satellite*. College Publications, 2022.

5 Arnold Beckmann. Proving Consistency of Equational Theories in Bounded Arithmetic. *J. Symb. Log.*, 67(1):279–296, 2002. doi:10.2178/JSL/1190150044.

6 Samuel R. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, Italy, 1986.

7 Samuel R. Buss and Aleksandar Ignjatović. Unprovability of consistency statements in fragments of bounded arithmetic. *Ann. Pure Appl. Logic*, 74(3):221–244, 1995.

8 Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Transaction of the American Mathematical Society*, 39:472–482, 1936.

9 Anupam Das. From positive and intuitionistic bounded arithmetic to monotone proof complexity. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 126–135, New York, NY, USA, 2016. Association for Computing Machinery.

10 Ken-etsu Fujita. The Church-Rosser theorem and quantitative analysis of witnesses. *Inf. Comput.*, 263:52–56, 2018. doi:10.1016/J.IC.2018.09.002.

11 Jeroen Ketema and Jakob Grue Simonsen. Least upper bounds on the size of confluence and church-rosser diagrams in term rewriting and λ-calculus. *ACM Trans. Comput. Log.*, 14(4):31:1–31:28, 2013. doi:10.1145/2528934.

12 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

13 Yoyuki Yamagata. Consistency proof of a fraement of pv with substitution in bounded arithmetic. *The Journal of Symbolic Logic*, 83(3):1063–1090, 2018. doi:10.1017/jsl.2018.14.

Graph Search Trees and the Intermezzo Problem

Jesse Beisegel
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Ekkehard Köhler
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Fabienne Ratajczak
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Robert Scheffler
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Martin Strehler
Department of Mathematics, Westsächsische Hochschule Zwickau, Germany

Abstract

The last in-tree recognition problem asks whether a given spanning tree can be derived by connecting each vertex with its rightmost left neighbor of some search ordering. In this study, we demonstrate that the last-in-tree recognition problem for Generic Search is NP-complete. We utilize this finding to strengthen a complexity result from order theory. Given a partial order π and a set of triples, the NP-complete intermezzo problem asks for a linear extension of π where each first element of a triple is not between the other two. We show that this problem remains NP-complete even when the Hasse diagram of the partial order forms a tree of bounded height. In contrast, we give an XP-algorithm for the problem when parameterized by the width of the partial order. Furthermore, we show that – under the assumption of the Exponential Time Hypothesis – the running time of this algorithm is asymptotically optimal.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Problems, reductions and completeness; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases graph search trees, intermezzo problem, algorithm, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.22

Related Version *Full Version*: `https://arxiv.org/abs/2404.18645` [3]

Acknowledgements The authors would like to thank Matjaž Krnc, Martin Milanič, and Nevena Pivač for fruitful discussions about first-in and last-in trees as well as about Paul and Mary.

1 Introduction

In the realm of computational combinatorics, one of the primary challenges is to determine a feasible configuration based on incomplete information. This paper aims to elucidate the relationships between two notable instances of this problem category: recognition of search trees of graph searches and total ordering with constraints. Specifically, our focus will be on exploring the last-in-tree recognition in the context of generic search and the intermezzo problem.

Graph Searches. Graph searches like *Breadth First Search* (BFS) or *Depth First Search* (DFS) are among the most basic algorithms in computer science. Their simplicity belies their significance as they form the backbone of more complex algorithms used to compute key properties of graphs. For instance, DFS can be employed to test for planarity as demonstrated by Hopcraft and Tarjan [17] and *Lexicographic Breadth First Search* (LBFS) aids in the recognition and minimum coloring of chordal graphs through a perfect elimination ordering [22]. Notably, all the above mentioned algorithms operate in linear time, underscoring their efficiency.

In this context, *Generic Search* (GS) represents the most general form of a graph search, with connectivity being its sole constraint: To elaborate, starting from a root vertex r, every subsequently visited vertex merely needs to be adjacent to a previously visited vertex. Consequently, GS can yield any total order of the vertices, provided each prefix is connected. A search methodology that bears a close resemblance to GS is the *Maximum Neighborhood Search* (MNS) [8], which can be perceived as a lexicographic variant of GS. Similarly, BFS and DFS can be implemented by using a queue and a stack, respectively, to store vertices that have not yet been visited.

Recognizing the significance of basic graph search algorithms such as BFS or DFS, recent efforts have been directed towards a deeper understanding of these algorithms. The primary focus of these studies revolves around two structures: end vertices and search trees (for a summary of known results see [26, Tables 1 and 2]). Given a graph G and a specific search rule (e.g., BFS or DFS), the *End Vertex Problem* aims to identify potential final vertices of the search. For GS, solving the end vertex problem is relatively straightforward. As long as a vertex v is not an articulation point, i.e., $G - v$ remains connected, v can serve as an end vertex of GS [5]. However, the end vertex problem is NP-complete for all other common search rules on general graphs [1, 5, 9, 31]. By restricting to special graph classes, linear-time algorithms have been developed to solve this problem, e.g., for BFS on split graphs [5], for DFS on interval graphs [1], and for MNS on chordal graphs [1].

Given a graph G and a spanning tree T, the *Tree Recognition Problem* seeks to determine whether T can be derived as a search tree. In essence, it questions the feasibility of reconstructing a linear order of vertices from the tree. This problem is typically studied in two variants: first-in-trees and last-in-trees [2]. In first-in-trees, each vertex is connected to its neighbor that appears first in the search order. Conversely, in last-in-trees (or $\mathcal{L}$-trees), each vertex v is a child of its neighbor that appears last before v in the search order. Normally, first-in-trees are used for BFS and last-in-trees for DFS, with existing linear-time algorithms capable of recognizing the corresponding trees in both cases [15, 16, 18, 20]. Interestingly, the problem becomes NP-complete when the search-tree paradigms are swapped between these searches, i.e., using last-in-trees for BFS and first-in-trees for DFS [24]. Furthermore, Scheffler [26] shows that the first-in-tree recognition problem of GS can be solved in linear time.

Total Ordering. A well-known theorem in order theory states that any partial order can be extended to a linear order. This holds true even for infinite sets, as demonstrated by Szpilrajn (Marczewski) through the use of the axiom of choice [28].[1] The process simplifies considerably for finite sets, where topological sorting algorithms can determine such an extension in linear time [7].

While partial orders are typically defined by a binary relation, total order problems offer a more general perspective. Here, one is given a set A, a family $\mathcal{B}$ of subsets $A_i \subseteq A$, and for each $A_i \in \mathcal{B}$ one or more valid orderings of the elements within A_i. The objective is to ascertain a total order of the elements in A that adheres to all these constraints.

Among the problems, the *Betweenness Problem* and the *Cyclic Ordering Problem* are particularly noteworthy. These two problems have already been discussed in the seminal textbook by Garey and Johnson [13]. In the betweenness problem, we are presented with

[1] He also references unpublished proofs by Banach, Kuratowski, and Tarski.

triples (a, b, c), and the only valid configurations are $a < b < c$ or $c < b < a$. In simpler terms, b must be positioned between a and c. The cyclic ordering problem involves given triples (a, b, c) for which there are three feasible orderings: $a < b < c$, $b < c < a$, or $c < a < b$. As the appearance in Garey and Johnson's book already suggests, both of these problems are indeed NP-complete.

In [14], Guttmann and Maucher systematically categorized total ordering problems based on pairs and triples. They also introduced the term *Intermezzo* to describe a specific variant: given pairs (b, c) where $b < c$, and triples (a, b, c) where either $a < b < c$ or $b < c < a$, implying that a is not placed between b and c. Note that a partial order is defined by both pairs and triples through the relation $b < c$. This problem has been proven to be NP-complete.

Interconnections. In this context, the problems of identifying end vertices and search trees are interconnected with the total ordering problem, given the underlying vertex order. The end vertex problem asks if a vertex can be the maximal element within this order. On the other hand, the correct search order offers a certificate for the search tree problem that can be checked in linear time. However, the constraints, which include all valid search orders and could potentially be exponential in number, are not explicitly given. Instead, they are implicitly defined by the underlying search paradigm.

Recently, Scheffler [23] introduced the more general problem of linearizing partial orders where the resultant total order must serve as a search order of a specified graph G. He presents polynomial-time algorithms for this problem for several searches and graph classes. In particular, he shows that the problem can be solved for GS on general graphs using a simple greedy algorithm. These results generalize the polynomial-time algorithms for the end vertex problem, given that the partial order can be selected to determine the end vertex.

Our Contribution. After providing the necessary notation, we prove NP-completeness of the $\mathcal{L}$-tree problem for Generic Search (GS) in Section 3. It is worth noting that two aspects of this result may appear surprising: Firstly, for GS all other problems considered so far can be solved in polynomial time with straightforward methods. Secondly, until now, for any given combination of a search rule (such as BFS, DFS, etc.) and a graph class (like chordal, interval, split, etc.), both tree recognition problems have not been harder than the end vertex problem. Thus, GS on general graphs represents the first known instance where the end vertex problem is simpler than a tree-recognition problem. We use the NP-completeness of the $\mathcal{L}$-tree problem of GS in Section 4 to show that the Intermezzo Problem is also NP-complete even if the partial order π is a cs-tree or the height of π is bounded. In contrast, we give an XP-algorithm for the problem when parameterized by the width of π. Under the assumption of the Exponential Time Hypothesis, we show that the running time of this algorithm is asymptotically optimal. Proofs omitted due to space constraints can be found in the full version [3].

2 Preliminaries

All the graphs that we consider are simple, finite, non-empty and undirected. Given a graph G, we denote by $V(G)$ the set of *vertices* and by $E(G)$ the set of *edges*.

A path P of G is a non-empty subgraph of G with $V(P) = \{v_1, \ldots, v_k\}$ and $E(P) = \{v_1v_2, \ldots, v_{k-1}v_k\}$, where $v_1, \ldots, v_k$ are all distinct. We will sometimes denote such a path by $v_1 - v_2 - \ldots - v_{k-1} - v_k$. A graph G is called a *tree* if it is connected and does not contain a cycle. A *spanning tree* T is a subgraph of a graph G which is a tree with $V(T) = V(G)$. A tree together with a distinguished *root vertex* r is said to be *rooted.* In such a rooted tree a vertex v is an *ancestor* of vertex w if v is an element of the unique path from w to the root r. In particular, if v is adjacent to w, it is called the *parent* of w. Furthermore, a vertex w is called the *descendant (child)* of v if v is the ancestor (parent) of w. We define the *height of a rooted tree* as the maximum number of edges of a path from the root r to any other vertex. A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set.

Given a set X, a *(binary) relation* $\mathcal{R}$ on X is a subset of the set $X^2 = \{(x, y) \mid x, y \in X\}$. The set X is called the *ground set of* $\mathcal{R}$. The *reflexive and transitive closure* of a relation $\mathcal{R}$ is the smallest relation $\mathcal{R}'$ such that $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{R}'$ is reflexive and transitive. A *partial order* π on a set X is a reflexive, antisymmetric and transitive relation on X. The tuple (X, π) is then called a *partially ordered set.* We also denote $(x, y) \in \pi$ by $x \prec_\pi y$ if $x \neq y$. A *minimal element* of a partial order π on X is an element $x \in X$ for which there is no element $y \in X$ with $y \prec_\pi x$. A *chain* of a partial order π on a set X is a set of elements $\{x_1, \ldots, x_k\} \subseteq X$ such that $x_1 \prec_\pi x_2 \prec_\pi \ldots \prec_\pi x_k$. The *height* of π is the number of elements of the largest chain of π. An *antichain* of π is a set of elements $\{x_1, \ldots, x_k\} \subseteq X$ such that $x_i \not\prec_\pi x_j$ for any $i, j \in \{1, \ldots, k\}$. The *width* of π is the number of elements of the largest antichain of π.

A *linear ordering* of a finite set X is a bijection $\sigma : X \rightarrow \{1, 2, \ldots, |X|\}$. We will often refer to linear orderings simply as orderings. Furthermore, we will denote an ordering by a tuple $(x_1, \ldots, x_n)$ which means that $\sigma(x_i) = i$. Given two elements x and y in X, we say that x is *to the left* (resp. *to the right*) of y if $\sigma(x) < \sigma(y)$ (resp. $\sigma(x) > \sigma(y)$) and we denote this by $x \prec_\sigma y$ (resp. $x \succ_\sigma y$).

A *vertex ordering* of a graph G is a linear ordering of the vertex set $V(G)$. A vertex ordering $\sigma = (v_1, \ldots, v_n)$ is called *connected* if for any $i \in \{1, \ldots, n\}$ the graph $G[v_1, \ldots, v_i]$ is connected. In this paper, a *graph search* is an algorithm that, given a graph G as input, outputs a connected vertex ordering of G. The graph search that is able to compute any such ordering is called *Generic Search (GS).*

3 Complexity of the $\mathcal{L}$-tree Recognition Problem

The definition of the term *search tree* varies between different paradigms. However, typically, it consists of the vertices of the graph and, given the search ordering $(v_1, \ldots, v_n)$, for each vertex v_i exactly one edge to a $v_j \in N(v_i)$ with $j < i$. By specifying to which of the previously visited neighbors a new vertex is adjacent in the tree, we can define different types of graph search trees. For example, in DFS trees a vertex v is adjacent to the rightmost neighbor to the left of v. This motivates the following definition.

▶ **Definition 3.1.** *Given a search ordering $\sigma := (v_1, \ldots, v_n)$ of a graph search on a connected graph G, we define the* last-in tree *(or $\mathcal{L}$-tree) to be the tree consisting of the vertex set $V(G)$ and an edge from each vertex v_i to its rightmost neighbor v_j in σ with $j < i$.*

As explained above, for a classical DFS the tree T is an $\mathcal{L}$-tree with respect to σ. Given this definition, we can state the following decision problem.

▶ **Problem 1** ($\mathcal{L}$-Tree Recognition Problem of graph search $\mathcal{A}$).
Instance: *A connected graph G and a spanning tree T of G.*
Question: *Is there a graph search ordering of $\mathcal{A}$ such that T is its $\mathcal{L}$-tree of G?*

Note that we have defined the $\mathcal{L}$-tree recognition problem without a given start vertex for the search. It is also possible to define this problem with a fixed start vertex and we call this the *rooted $\mathcal{L}$-tree recognition problem.* Obviously, a polynomial-time algorithm for the rooted tree recognition problem yields a polynomial-time algorithm for the general problem by simply repeating the procedure for all vertices. The other direction, however, is not necessarily true.

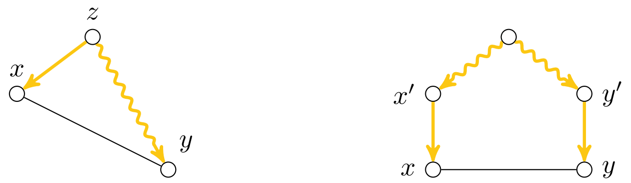


Figure 1 On the left is an example of a *hook configuration*. On the right is an example of a *U-bend*. The yellow edges symbolize edges of the spanning tree, the black edges are non-tree edges of the graph and the wavy line represents a directed path in the spanning tree.

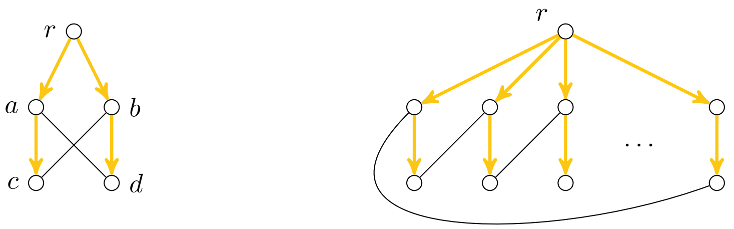


Figure 2 Family of graphs where the rooted spanning trees (yellow edges) are not $\mathcal{L}$-trees of GS.

The $\mathcal{L}$-tree recognition problem of GS raised in [24] and [27] is an open problem and in the following we will show that it is in fact NP-complete. This result will also answer another open question, as Scheffler showed in [24] that the $\mathcal{L}$-tree recognition problem of BFS for split graphs is at least as hard as that of GS.

An important property of $\mathcal{L}$-trees of GS can be derived from the non-tree edges that connect vertices of different branches of the tree.

▶ **Lemma 3.2.** *Let T be a spanning tree of a graph G rooted in r. Let xy be an edge in $E(G) \setminus E(T)$ and let x' and y' be the parents of x and y in T, respectively. If T is an $\mathcal{L}$-tree of a GS ordering σ starting with r, then it either holds that $x' \prec_\sigma x \prec_\sigma y' \prec_\sigma y$ or $y' \prec_\sigma y \prec_\sigma x' \prec_\sigma x$.*

The configuration described in Lemma 3.2 will be called a *U-bend configuration* (see Figure 1 on the right).

Before we begin with our main results, we should analyze examples of some rooted spanning trees that cannot be $\mathcal{L}$-trees of GS. One of the smallest examples can be found to the left in Figure 2. The example on the right is a generalization with arbitrary many branches of the spanning tree. These examples can be easily described using a concept called *hook configuration*. This is a special case of a U-bend where the parent of one vertex is an ancestor of the others.

▶ **Definition 3.3.** *Let T be a spanning tree of a graph G rooted in $r \in V(G)$. We say that a triple of vertices x, y, and z forms a* hook configuration *or a* hook *if z is the parent of x in T, $xy \in E(G) \setminus E(T)$ and y is a descendant of z but y is not a descendant of x (see Figure 1 on the left). We call x the* point *and y the* eye *of the hook.*

These hook configurations have a strong a priori effect on the sequence of any search ordering corresponding to that tree.

▶ **Lemma 3.4.** *Let x and y be part of a hook configuration of T rooted in $r \in V(G)$ with point x and eye y. Then for any GS ordering σ starting in r with $\mathcal{L}$-tree T it holds that $x \prec_\sigma y$.*

For the examples shown in Figure 2, it is possible to see that the hook configurations create something like a cycle in the ordering using Lemma 3.4: We see that $a \prec d$ and $b \prec c$ and these contradict each other because of the tree edges.

In the special case that the graph together with its spanning tree does not contain any hooks, it is trivial to decide the $\mathcal{L}$-Tree Recognition Problem.

▶ **Theorem 3.5.** *Let T be a spanning tree of a graph G rooted in $r \in V(G)$. If there is no hook configuration, then any DFS ordering of T starting in r is a GS ordering of G with $\mathcal{L}$-tree T. Therefore, any such tree together with G is a* Yes*-instance for the $\mathcal{L}$-Tree Recognition Problem of Generic Search.*

Proof. Let σ be a DFS ordering of T starting in r. This ordering σ fulfills the following property also called *four point condition* (see for example [8]): If $a \prec_\sigma b \prec_\sigma c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $a \prec_\sigma d \prec_\sigma b$ such that $db \in E$.

Suppose that σ does not induce the $\mathcal{L}$-tree T for G. Let w be the leftmost vertex in σ such that there exist u and v with $u \prec_\sigma v \prec_\sigma w$ with $uw \in E(T)$ and $vw \in E(G) \setminus E(T)$. If uv is an edge in the tree T, then u, v and w form a hook configuration; a contradiction to the assumption. Therefore, we can assume that $uv \notin E(T)$. Now we can apply the four point condition to vertices u, v and w (note that the DFS was executed on T). A result of the four point condition, is the fact that there must exist a u-v-path $u = d_1 - \cdots - d_k = v$ with $u \prec_\sigma d_2 \prec_\sigma \cdots \prec_\sigma d_{k-1} \prec_\sigma v$ (Corollary 2.6 in [8]). In particular, we see that v is a descendant of u. Together with the fact that $uw \in T$ and $vw \in E(G) \setminus E(T)$, we see that u, v and w form a hook configuration; a contradiction to the assumptions of the theorem. This implies that each vertex in the search is connected to its correct parent in T, proving that T is an $\mathcal{L}$-tree of GS in G. ◀

This theorem could lead to the assumption that deciding whether a given spanning tree is an $\mathcal{L}$-tree of GS only amounts to an analysis of all the hook configurations. In fact, it is easy to see that we can find all hook configurations in polynomial time. However, it is not always so simple. In fact, we show in the following that in general the $\mathcal{L}$-Tree Recognition Problem of GS is NP-complete. We describe a reduction from 3-SAT, i.e., we are given an instance $\mathcal{I}$ of 3-SAT and derive an instance $(G(\mathcal{I}), T(\mathcal{I}))$ for the $\mathcal{L}$-Tree Recognition Problem of GS.

Let $\mathcal{I}$ be an instance of 3-SAT with variable set $\{X_1, \ldots, X_n\}$ and clause set $\{C^1, \ldots, C^m\}$. For ease of notation, we define the positive literal X_j as $X_j(1)$ and the negative literal $\neg X_j$ as $X_j(0)$. First we define the spanning tree $T(\mathcal{I})$ and then we add the edges missing to give the full graph $G(\mathcal{I})$. For each variable X_j of $\mathcal{I}$ we add two vertices $x_j(0)$ and $x_j(1)$ (representing the two literals of X_j) to $V(G(\mathcal{I}))$. These vertices are all adjacent to a common root vertex r. Furthermore, r is adjacent to the clause-hub-vertex C. Now we add a vertex c^i for each clause C^i and connect it to C (see Figure 3 for a depiction of this setup.). Furthermore, for each occurrence of a literal associated with vertex $x_j(p)$ in a clause C^i we add a vertex x^i_j. As we may assume that only one of the literals appears in a clause, we do not have to use an index to mark whether the vertex belongs to a negated variable or not. This sums up the basic setup concerning the variables and clauses. However, for technical reasons we need several more vertices (see Figure 4):

- For each c^i we add the vertices $a^i_0, a^i_1, a^i_2, b^i_0, b^i_1, b^i_2$ in $T(\mathcal{I})$. These are called the *technical vertices of c^i*.
- These vertices form the paths $c^i - a^i_0 - b^i_0$, $c^i - a^i_1 - b^i_1$ and $c^i - a^i_2 - b^i_2$ in $T(\mathcal{I})$.
- For each x^i_j we add vertices d^i_j, e^i_j, f^i_j. These are called the *technical vertices of x^i_j*.
- These vertices form the path $x^i_j - d^i_j - e^i_j - f^i_j$ in $T(\mathcal{I})$.

This concludes the definition of the tree $T(\mathcal{I})$, to which we will now add the remaining edges for $G(\mathcal{I})$ (see Figure 3 and Figure 4):

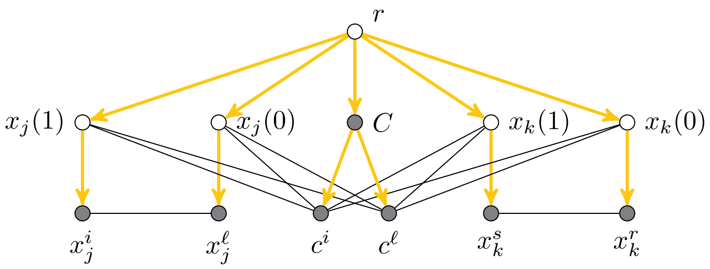


Figure 3 This figure illustrates the variable gadget. The yellow edges are the tree edges, and the vertices marked in gray appear again in the clause gadget. The literal X_j appears in clause C^k, the literal $\neg X_j$ appears in clause C^ℓ. Note that for the x_j^i vertices we do not need to denote whether they belong to X_j or its negation, as each clause only contains either X_j or $\neg X_j$.

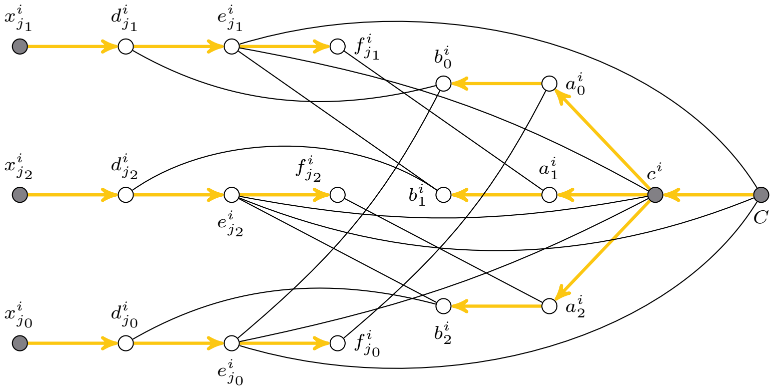


Figure 4 This figure illustrates the clause gadget. The tree edges are colored yellow and the gray vertices mark the vertices that can be found in the variable gadget. Note that we have drawn this figure horizontally to make the embedding cleaner. The directions of the tree edges denote the direction from the root of the tree.

- For each X_j we add edges $x_j(0)c^i$ and $x_j(1)c^i$ for $i \in \{1, \dots, m\}$.
- For each x_j^i adjacent to x_j we add an edge $x_j^i x_j^k$ if both X_j occurs in clause C^i and $\neg X_j$ occurs in literal C^k.
- For any clause $C^i = \{X_{j_0}(p_0), X_{j_1}(p_1), X_{j_2}(p_2)\}$ we add the edges:
  - $Ce^i_{j_p}$ for $p \in \{0, 1, 2\}$.
  - $c^i e^i_{j_p}$ for $p \in \{0, 1, 2\}$.
  - $a^i_p f^i_{j_p}$ for $p \in \{0, 1, 2\}$.
  - $b^i_p e^i_{j_p}$ for $p \in \{0, 1, 2\}$.
  - $b^i_p d^i_{j_{(p+1) \bmod 3}}$ for $p \in \{0, 1, 2\}$.

The modulo operation applied to the indices to define the edges in the clause gadget illustrates the circularity inherent in that gadget. Visiting one of the branches below c^i effectively unlocks one of the branches below an x^i_j. Conversely, visiting an x^i_j before c^i blocks a corresponding branch of c^i. This effect is what leads to the property that we will show in Lemma 3.7.

The first step in our reduction is to check whether we can use a search ordering that achieves $T(\mathcal{I})$ to construct an assignment of the 3-SAT instance $\mathcal{I}$. For each variable we will choose the assignment that corresponds to the literal vertex chosen *second*, i.e., if $x_j(1)$ is chosen first we assign X_j the value 0 and if $x_j(0)$ is chosen first we assign X_j the value 1. The following lemma shows that the children of the literal vertex that is chosen first are visited before the clause vertices.

▶ **Lemma 3.6.** *If $T(\mathcal{I})$ is an $\mathcal{L}$-tree of some GS ordering σ of $G(\mathcal{I})$ starting in r, then it holds for every variable x_j that all the children of $x_j(1)$ or all the children of $x_j(0)$ are to the left of vertex C in σ. In particular, if $x_j(p) \prec_\sigma x_j(q)$, then the children of $x_j(p)$ are all to the left of $x_j(q)$.*

Proof. The sets of vertices $\{x_j(1), c^i, r\}$ and $\{x_j(0), c^i, r\}$ form hook configurations with eye c^i and point $x_j(1)$ or $x_j(0)$, respectively. By Lemma 3.4, it holds that $x_j(1) \prec_\sigma c^i$ and $x_j(0) \prec_\sigma c^i$. Furthermore, with Lemma 3.2 we see that $x_j(1) \prec_\sigma C$ and $x_j(0) \prec_\sigma C$, as for example $r, x_j(1), c^i$ and C form a U-bend. W.l.o.g., we may assume that $x_j(1) \prec_\sigma x_j(0)$. Let u be an arbitrary child of $x_j(1)$ and let v be an arbitrary child of $x_j(0)$. By construction of $G(\mathcal{I})$ and $T(\mathcal{I})$, we know that $uv \in E(G(\mathcal{I})) \setminus E(T(\mathcal{I}))$. Then, due to Lemma 3.2 and our assumption, we see that $u \prec_\sigma x_j(0) \prec_\sigma c_i$. Therefore, the children of $x_j(1)$ are to the left of $x_j(0)$ and thus also to the left of the clause hub vertex C, proving the statement of the lemma. ◀

The next lemma motivates how the choice of the variable assignment is used to check whether a given clause is fulfilled.

▶ **Lemma 3.7.** *If $T(\mathcal{I})$ is an $\mathcal{L}$-tree of $G(\mathcal{I})$ for the search ordering σ, then for each clause $C^i = \{X_{j_0}(p_0), X_{j_1}(p_1), X_{j_2}(p_2)\}$ it holds that $c^i \prec_\sigma x^i_{j_q}$ for some $q \in \{0, 1, 2\}$.*

Proof. Assume for contradiction that c^i is to the right of $x^i_{j_q}$ for all $q \in \{0, 1, 2\}$. Using the hook and U-bend rules from Lemmas 3.2 and 3.4, we can show that $T(\mathcal{I})$ is not an $\mathcal{L}$-tree for σ. For all $q \in \{0, 1, 2\}$, the set $\{C, e^i_{j_q}, r\}$ forms a hook with point C and eye $e^i_{j_q}$ and, thus, $C \prec_\sigma e^i_{j_q}$. Now the U-bend induced by the edge $e^i_{j_q} c^i$ implies that $c^i \prec_\sigma e^i_{j_q}$. For any $q \in \{0, 1, 2\}$, the vertex $d^i_{j_q}$ is adjacent to some vertex $b^i_{q'}$ and the corresponding edge induces a U-bend. Since all three vertices $x^i_{j_q}$ are to the left of all three vertices $a^i_{q'}$, these U-bends imply that at least one vertex $d^i_{j_q}$ is to the left of all $a^i_{q'}$. Fix that vertex $d^i_{j_q}$. Now the edge $b^i_q e^i_{j_q}$ implies that $e^i_{j_q} \prec_\sigma a^i_q$. Summarizing, $c^i \prec_\sigma e^i_{j_q} \prec_\sigma a^i_q$. However, this contradicts Lemma 3.2 as the edge $f^i_{j_q} a^i_q$ induces a U-bend where both parents (c^i and $e^i_{j_q}$) are to the left of both children (a^i_q and $f^i_{j_q}$). This concludes the proof. ◀

This lemma shows that for each clause there is one literal for which the corresponding child belonging to that clause has to be chosen after the clause vertex. This will be the literal that satisfies the clause in a fulfilling assignment. On the other hand, if there is a literal whose child is chosen after the clause vertex, then the corresponding clause gadget can be correctly traversed. Combining these results proves the main result of this section.

▶ **Theorem 3.8.** *The rooted $\mathcal{L}$-tree recognition problem of Generic Search is* NP*-complete for rooted spanning trees of height* 5*.*

Proof. Let σ be a GS ordering with $\mathcal{L}$-tree $T(\mathcal{I})$. We define an assignment $\mathcal{A}$ by setting any variable to false if and only if the positive literal appears before its negative literal in σ. We claim that this is a fulfilling assignment. Clearly we only need to show that for each clause at least one literal was chosen to be true. In particular, by Lemma 3.7 we see that for each $C^i = \{X_{j_0}(p_0), X_{j_1}(p_1), X_{j_2}(p_2)\}$ at least one vertex $x^i_{j_q}$ is to the right of c^i and, thus, to the right of the vertex C. Due to Lemma 3.6, the parent of $x^i_{j_q}$ is to the right of the vertex of variable. This implies that at least one literal contained in C^i is fulfilled in $\mathcal{A}$.

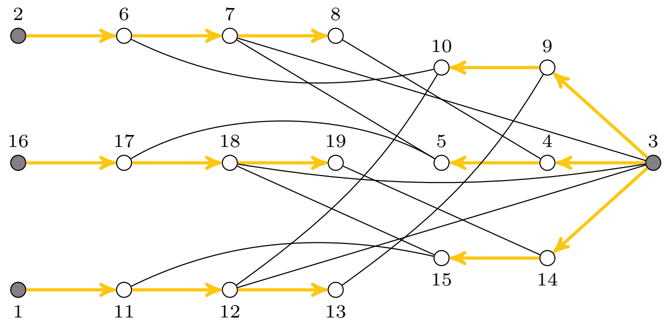


Figure 5 Example of an ordering of the technical vertices for a clause where two literals (1 and 2) were chosen to be false. Compare with Figure 4.

Let $\mathcal{A}$ be a fulfilling assignment of $\mathcal{I}$. We now show that in this case we can construct a GS ordering of $G(\mathcal{I})$ which has the $\mathcal{L}$-tree $T(\mathcal{I})$. The broad idea is to choose the literals that are false for $\mathcal{A}$ first followed by their children. Then we choose the literals that are true, followed by the clause hub vertex C and then the clause vertices. Finally, we need to visit the technical vertices in the correct order, followed by the descendants of the true literals. In the following, we define several suborders that need to be combined into the final linear ordering σ.

As explained above, we begin the search ordering σ by visiting the root r and then all literals that are set to false by $\mathcal{A}$ in arbitrary order. Next we visit all children of these vertices. In the next phase, we visit all remaining literal vertices in an arbitrary order. At this point, we can visit the clause hub vertex C (see Lemma 3.6) followed by the clause vertices in arbitrary order. Let $C^i = \{X_{j_0}(p_0), X_{j_1}(p_1), X_{j_2}(p_2)\}$ be some clause. If all literals of C^i were chosen to be true (i.e., all the vertices $x^i_{j_0}$, $x^i_{j_1}$, and $x^i_{j_2}$ are to the right of c^i), then we can visit the technical vertices of c^i in the order $a^i_0, a^i_1, a^i_2, b^i_0, b^i_1, b^i_2$ (or any other order that conforms with GS) followed by the literal vertices and their technical vertices following the order that is implied by the tree edges. If exactly one of the literals, say w.l.o.g. $X_{j_0}(p_0)$, is chosen to be false by $\mathcal{A}$, then we use the order $x^i_{j_0}, c^i, a^i_0, b^i_0, d^i_{j_0}, e^i_{j_0}, f^i_{j_0}$. Then we visit the remaining technical vertices of c^i followed by $x^i_{j_1}, x^i_{j_2}$ and their technical vertices.

If exactly two of the literals, say w.l.o.g. $X_{j_0}(p_0)$ and $X_{j_1}(p_1)$, are chosen to be false by $\mathcal{A}$, then we use the order $x^i_{j_0}, x^i_{j_1}, c^i, a^i_1, b^i_1, d^i_1, e^i_1, f^i_1, a^i_0, b^i_0, d^i_0, e^i_0, f^i_0$. Then we visit the remaining technical vertices of c^i followed by $x^i_{j_2}$ and its technical vertices (see Figure 5 for an illustration).

Because $\mathcal{A}$ is a fulfilling assignment, we know that each clause has at most two literals that are set to false. Therefore, we can combine all of these orderings to a comprehensive GS ordering σ and confirm that $T(\mathcal{I})$ is in fact an $\mathcal{L}$-tree of σ. ◀

Using [24, Theorems 21 and 23], we can strengthen Theorem 3.8 to the case of split graphs and give a similar result for Breadth First Search.

▶ **Corollary 3.9.** *The rooted $\mathcal{L}$-tree recognition problems of Generic Search and Breadth First Search are* NP*-complete even if the input is restricted to split graphs and to rooted spanning trees of height* 12.

Note that these results are all for the *rooted* $\mathcal{L}$-tree recognition problem. Using a small gadget, we can also extend the hardness of the GS $\mathcal{L}$-tree recognition to the unrooted problem.

▶ **Corollary 3.10.** *The $\mathcal{L}$-tree recognition problem of Generic Search is* NP*-complete.*

Proof. Let G be some graph and T be a spanning tree of G with root r. We add three vertices a, b, c to G in the following way to form a new graph G': Let $V(G') = V(G) \cup \{a, b, c\}$. Furthermore, $E(G') = E(G) \cup \{ab, ac, bc, ar\}$. Finally, we define a spanning tree T' of G' with $V(T') = V(G')$ and $E(T') = E(T) \cup \{ar, ab, ac\}$.

Due to the conflicting hooks among a, b, c, either b or c must be visited before a, if T' is to be an $\mathcal{L}$-tree of G'. This makes r the de facto root of $T' - \{a, b, c\}$, showing that the rooted $\mathcal{L}$-tree recognition problem for G and T is equivalent to the unrooted one for G' and T'. ◀

4 The Intermezzo Problem

Given a rooted spanning tree T, the basic property that has to be fulfilled by a vertex ordering σ for it to have T as an $\mathcal{L}$-tree is the following: If there is a vertex z with parent y and z has a non-tree edge to vertex x, then x is not allowed to be between y and z in the vertex ordering. These constraints are similar to those used in the following problem introduced by Guttmann and Maucher [14].

▶ **Problem 2** (General Intermezzo Problem).
Instance: *Finite set A, set C of triples of distinct elements of A*
Question: *Is there an ordering of A such that for all $(x, y, z) \in C$ it holds that $x \prec_\sigma y \prec_\sigma z$ or $y \prec_\sigma z \prec_\sigma x$?*

We call an ordering that fulfills the constraints of C an *intermezzo ordering*. Note that Guttmann and Maucher do not give a name for the general problem as they introduce it as one case of a large family of constrained ordering problems. We derived the name from the more restricted *Intermezzo problem*. This problem additionally forces the triples in C to be pairwise disjoint.

▶ **Problem 3** (Intermezzo Problem [14]).
Instance: *Finite set A, set B of pairs of A, set C of pairwise disjoint triples of distinct elements of A.*
Question: *Is there an ordering of A such that for all $(x, y) \in B$ it holds that $x \prec_\sigma y$ and for all $(x, y, z) \in C$ it holds that $x \prec_\sigma y \prec_\sigma z$ or $y \prec_\sigma z \prec_\sigma x$?*

Besides the tuples in B, in both problems the second and the third entry of the triples in C imply simple order constraints on the elements of A. Therefore, we can define the relations $\pi(B, C)$ and $\pi(C)$, respectively, as the reflexive and transitive closure of the relation $R \subseteq A \times A$ where $(y, z) \in R$ if and only if $(y, z) \in B$ or there is some tuple $(x, y, z) \in C$. If (A, C) is a positive instance of the General Intermezzo problem, then $\pi(C)$ must form a partial order and every intermezzo ordering of (A, C) forms a linear extension of $\pi(C)$. The same properties hold for positive instances (A, B, C) of the Intermezzo problem and the partial order $\pi(B, C)$. Thus, we also can interpret the (General) Intermezzo problem as a special kind of linear extension problems with additional non-betweenness constraints. This motivates the consideration of restricted problems where the partial order has to fulfill certain properties.

4.1 The Intermezzo Problem for CS-trees

Following the terminology of Trotter [29], we call a partial order a *cs-tree* (short for *computer science tree*) if its Hasse diagram forms a tree rooted in the unique minimal element. Using the terminology in [25], we call a leaf of a rooted tree a *branch leaf* if it is not equal to the root of the tree. Recall that the height of a cs-tree and the height of the tree that is formed by its Hasse diagram differ by one.

▶ **Lemma 4.1.**

1. *The rooted $\mathcal{L}$-tree recognition problem of Generic Search for rooted spanning trees of fixed height $h \geq 2$ is polynomial-time reducible to the General Intermezzo problem for instances (A, C) where $\pi(C)$ is a cs-tree of height $h+1$.*
2. *The General Intermezzo problem for instances (A, C) where $\pi(C)$ is a cs-tree of fixed height $h \geq 2$ is polynomial-time reducible to the rooted $\mathcal{L}$-tree recognition problem of Generic Search for rooted spanning trees of height $2h-1$.*
3. *The rooted $\mathcal{L}$-tree recognition problem of Generic Search for rooted spanning trees having k branch leaves is polynomial-time equivalent to the General Intermezzo problem for instances (A, C) where $\pi(C)$ is a cs-tree of width k.*

Proof. First, we reduce the $\mathcal{L}$-tree recognition problem of GS to the General Intermezzo Problem. Let G be a graph and T be a spanning tree of G rooted in r of height $h \geq 2$. Let $A = V(G) \cup \{s\}$ where $s \notin V(G)$. Let C be the set containing the following triples:

(C1) (r, t, s), for some child t of r in T,

(C2) (s, u, v), for any vertex $v \in V(G) \setminus \{r\}$ and its parent u in T,

(C3) (w, u, v), for any vertex $v \in V(G) \setminus \{r\}$, its parent u in T and any vertex w with $vw \in E(G) \setminus E(T)$.

It is easy to see that $\pi(C)$ is a cs-tree of height $h+1$. We claim that there is an intermezzo ordering of A fulfilling the constraints given by C if and only if T is a rooted GS $\mathcal{L}$-tree of G.

First assume that there is an intermezzo ordering σ fulfilling the constraints of C. We delete s from σ and call the resulting ordering σ'. The following claim is implied by the constraints given in (C2).

▷ Claim 1. If u is the parent of v in T, then $u \prec_{\sigma'} v$.

This claim implies directly that σ' is a GS ordering of G starting in r. Now assume for contradiction that the $\mathcal{L}$-tree T' of σ' is not equal to T. Then there is a vertex v whose parent u' in T' is different from its parent u in T. By Claim 1, it holds that $u \prec_{\sigma'} v$. Therefore, it must hold that $u \prec_{\sigma'} u' \prec_{\sigma'} v$. This implies that u' is not a child of v in T, due to Claim 1. Hence, the edge $u'v$ is not part of T but part of G. Then, the set C contains the triple (u', u, v) (see (C3)). This is a contradiction because σ' and, thus, σ does not fulfill the constraint given by that triple.

Now assume that T is the $\mathcal{L}$-tree of the GS ordering σ starting with r. Then let σ' be the ordering constructed by appending s to the end of σ. The ordering fulfills the constraint (r, t, s) given in (C1). Furthermore, as parents are to the left of their children in σ, the ordering σ' also fulfills the constraints given by (C2). Assume for contradiction that some triple (w, u, v) of (C3) is not fulfilled in σ', i.e., $u \prec_{\sigma'} w \prec_{\sigma'} v$. Then u is not the parent of v in the $\mathcal{L}$-tree of σ since w is a neighbor of v and w lies between u and v in σ'; a contradiction.

This proves the first statement of the lemma. To prove the same direction for the third statement, we slightly change the set C. Instead of the triple (r, t, s) given in (C1), we add the triple (t, s, r). It is easy to see that then the width of $\pi(C)$ is equal to the number of branch leaves of T. The rest of the proof works analogously with the only difference that we append s to the beginning and not to the end of the GS ordering of G.

Now we reduce the General Intermezzo problem where $\pi(C)$ is a cs-tree to the rooted $\mathcal{L}$-tree recognition problem of GS. Let (A, C) be an instance where $\pi(C)$ is a cs-tree of height h and width k. We define the vertex set to be $V(G) := \{v^1, v^2 \mid v \in A\}$. Let H be the Hasse diagram of $\pi(C)$. The set of edges of T is defined as $E(T) := \{v^1 v^2 \mid v \in A\} \cup \{u^2 v^1 \mid uv \in E(H) \wedge (u, v) \in \pi(C)\}$. Let $(x, y, z) \in C$ and let $y = w_0, w_1, \ldots, w_\ell = z$ be the elements of the path between y and z in the Hasse diagram of $\pi(C)$. We add a non-tree edge to G from x^2 to any vertex w_i^1 and w_i^2 with $i \geq 1$. It is obvious that the constructed tree T has height $2h - 1$ and k branch leaves if it is rooted in the minimal element of $\pi(C)$.

First assume that there is an intermezzo ordering σ for (A, C). Then let σ' be the ordering that is constructed by replacing every element $v \in A$ in σ by the ordering (v^1, v^2). Then we claim that σ' is a GS ordering of G having $\mathcal{L}$-tree T. Let T' be the $\mathcal{L}$-tree of σ'. Obviously, it holds for σ' that any vertex is to the right of its parent in T since σ' is constructed from a linear extension of $\pi(C)$. Therefore, σ' is a GS ordering of G. Furthermore, any vertex v^2 has parent v^1 both in T and T' since these two vertices are consecutive in σ'. Assume for contradiction that there is a vertex v^1 whose parent in T is u^2 but the parent of v^1 in T' is t^p with $u^2 \neq t^p$. By construction it holds that $t^p = t^2$ since v^1 has no neighbor with index 1. It holds that $u^2 \prec_{\sigma'} t^2 \prec_{\sigma'} v^1$ and t^2v^1 is an edge in $E(G) \setminus E(T)$. This non-tree edge has been added to $E(G)$ because of some triple $(t, a, b) \in C$ where a is an ancestor of v and b is a descendant of v in the Hasse diagram of $\pi(C)$ (or $b = v$). However, by construction of σ', it holds that $a^2 \prec_{\sigma'} u^2 \prec_{\sigma'} t^2 \prec_{\sigma'} v^1 \prec_{\sigma'} b^2$. Hence $a \prec_\sigma t \prec_\sigma b$; a contradiction to the fact that σ is an intermezzo ordering of C.

Now assume that there is a GS ordering σ of G having T as its $\mathcal{L}$-tree. We construct the ordering σ' of A as follows. Consider the subordering of σ containing only the vertices v^2 for any $v \in A$ and replace v^2 by v. We claim that σ' is an intermezzo ordering of (A, C). Let (x, y, z) be a triple in C. It holds that $y \prec_\sigma z$ since y^2 is an ancestor of z^2 in T. Assume for contradiction that $y \prec_{\sigma'} x \prec_{\sigma'} z$. Consider the $w_0, \ldots, w_\ell$ as defined above. Note that these vertices appear by ascending index in σ'. Let w_i be the leftmost of these vertices in σ' that is to the right of x. Then it holds that $y \preceq_{\sigma'} w_{i-1} \prec_{\sigma'} x \prec_{\sigma'} w_i \preceq_{\sigma'} z$. This implies that $w_{i-1}^2 \prec_\sigma x^2 \prec_\sigma w_i^2$. Now we consider two cases. If $w_i^1 \prec_\sigma x^2$, then it must hold that $w_{i-1}^2 \prec_\sigma w_i^1 \prec_\sigma x^2 \prec_\sigma w_i^2$ as w_{i-1}^2 is the parent of w_i^1 in T. However, by construction x^2 is adjacent to w_i^2 and, hence, w_i^1 cannot be the parent of w_i^2 in the $\mathcal{L}$-tree of σ. If $x_2 \prec_\sigma w_i^1$, then it must hold that $w_{i-1}^2 \prec_\sigma x^2 \prec_\sigma w_i^1 \prec_\sigma w_i^2$ as w_i^1 is the parent of w_i^2 in T. However, by construction x^2 is adjacent to w_i^1 and, hence, w_{i-1}^2 cannot be the parent of w_i^1 in the $\mathcal{L}$-tree of σ. This contradicts the choice of σ. ◀

Combining the first statement of the lemma with Theorem 3.8 yields the following result.

▶ **Theorem 4.2.** *The General Intermezzo problem is* NP*-complete even if the input is restricted to instances (A, C) where $\pi(C)$ is a cs-tree of height* 6.

We can extend this result to the Intermezzo problem as follows.

▶ **Lemma 4.3.** *The General Intermezzo Problem for instances (A, C) where $\pi(C)$ is a cs-tree of width k is polynomial-time reducible to the Intermezzo Problem for instances (A', B', C') where $\pi(B', C')$ is a cs-tree of width k.*

This lemma implies the following complexity result.

▶ **Theorem 4.4.** *The Intermezzo problem is* NP*-complete even if the input is restricted to instances (A, B, C) where $\pi(B, C)$ is a cs-tree.*

As was shown by Wolk [30], cs-trees have dimension 2. Combining this with Theorems 4.2 and 4.4, we get the following result.

▶ **Corollary 4.5.** *The (General) Intermezzo problem is* NP*-complete even if $\pi(B, C)$ or $\pi(C)$, respectively, has dimension at most 2.*

Note that – in difference to Theorem 4.2 – we were not able to bound the height of the partial order in Theorem 4.4 since in the proof of Lemma 4.3 the height of the constructed partial order depends on the number of elements in A. We can adapt the proof of that lemma such that the height of the partial order increases only by a constant factor. However, in this case, we then loose the property that the partial order is a cs-tree.

▶ **Corollary 4.6.** *The Intermezzo problem is* NP*-complete even if the input is restricted to instances (A, B, C) where $\pi(B, C)$ has height* 36*.*

The complexity of the Intermezzo problem for cs-trees of bounded height remains open.

4.2 The Intermezzo Problem for Partial Orders of Bounded Width

As we have seen in the section above, the (General) Intermezzo problem is NP-complete even if the height or the dimension of the partial order is bounded. One may ask whether this also holds for another notable parameter of partial orders, the width. Adapting an idea of Colbourn and Pulleyblank [6] (explained in more detail in [4]), we can show that – unless P = NP – this is not the case as we can give an XP-algorithm for the General Intermezzo problem parameterized by the width of $\pi(C)$.

▶ **Theorem 4.7.** *The General Intermezzo problem can be solved in time $\mathcal{O}(k \cdot n^{k+2})$ on any instance (A, C) where $n = |A|$ and k is the width of $\pi(C)$.*

Proof. We only sketch the idea of the algorithm; for a comprehensive description and analysis of a similar algorithm see [4]. Using Dilworth's Chain Covering Theorem [11], we can partition the set A into k disjoint chains of $\pi(C)$. Now the set of elements of any prefix σ^{pre} of a linear extension of $\pi(C)$ can be represented by a tuple $(a_1, \dots, a_k) \in \{0, \dots, |A|\}^k$ where a_i represents the number of elements of chain i that are part of σ^{pre}. Since all elements of a chain are strictly ordered, the number of used elements of the chain directly implies the elements of the chain that are part of the prefix set.

Now the algorithm uses dynamic programming to compute whether a given prefix set can be reached in such a way that it fulfills all conditions of C. To this end, we have a table M with 0-1-entries for every tuple representing a prefix. We fill the entries of this table inductively, starting with those tuples whose entries sum up to 1. Such a tuple gets a 1-entry in M if and only if the minimal element of the respective chain is a minimal element of the partial order. For tuples $\gamma = (a_1, \dots, a_k)$ with larger entry sums we check for any tuple γ' that is constructed by decrementing exactly one non-zero entry of γ, say a_i, the following:

1. Is the M-entry of tuple γ' equal to 1?
2. Is the a_i-th element x of the i-th chain minimal in $\pi(C)$ restricted to those elements that are not part of the prefix set encoded by γ'?
3. Is there no triple (x, y, z) with y is an element of the prefix set encoded by γ' and z is not such an element?

If the answer to all three question is yes, then we set the M-entry of γ to 1. It is easy to check that M has $\mathcal{O}(n^k)$ entries and for any entry we can answer the three questions above for all triples γ' in time $\mathcal{O}(kn^2)$. This leads to the claimed running time. ◀

Lemma 4.1 implies an XP-algorithm for the rooted $\mathcal{L}$-tree recognition of GS parameterized by the number of branch leaves of the given spanning tree.

▶ **Corollary 4.8.** *The rooted $\mathcal{L}$-tree recognition problem of Generic Search can be solved in time $\mathcal{O}(k \cdot n^{k+2})$ on a graph G and a rooted spanning tree T of G where $n = |V(G)|$ and k is the number of branch leaves of T.*

One may ask whether this rather poor running time bound given in Theorem 4.7 can be improved significantly and whether there is an FPT algorithm for the (General) Intermezzo problem parameterized by the width of the partial order. We will show that – under certain assumptions – this is not the case.

▶ **Theorem 4.9.** *The (General) Intermezzo problem is* W[1]*-hard if it is parameterized by the width k of $\pi(C)$ or $\pi(B, C)$, respectively, even if that partial order is a cs-tree. Furthermore – under the assumption of the Exponential Time Hypothesis – there is no algorithm that solves the problem in time $f(k) \cdot n^{o(k)}$ for any computable function f where $n = |A|$.*

We prove this result by an FPT-reduction from the following problem, applying a technique also used in [4].

▶ **Problem 4** (Multicolored Clique Problem (MCP)).
Instance: *A graph G with a proper coloring of k colors.*
Question: *Is there a clique in G that contains exactly one vertex of each color?*

The MCP was shown to be W[1]-hard by Pietrzak [21] and independently by Fellows et al. [12]. In fact, in [10, 19] the authors show the following result.

▶ **Theorem 4.10** (Cygan et al. [10], Lokshtanov et al. [19]). *Under the assumption of the Exponential Time Hypothesis, there is no $f(k)n^{o(k)}$ time algorithm for the Multicolored Clique Problem for any computable function f where n is the number of vertices of the given graph.*

We give an FPT-reduction from the MCP to the General Intermezzo problem parameterized by the width of $\pi(C)$. Lemma 4.3 implies such a reduction also for the Intermezzo problem.

Let G be an instance of the MCP with k colors. W.l.o.g. we may assume that every color class has exactly q elements, i.e., we assume that $V(G) = \{v_p^i \mid 1 \leq i \leq k, 1 \leq p \leq q\}$. In the following, we construct an equivalent instance (A, C) for the General Intermezzo problem.

First we describe the set A. For every $i \in \{1, \dots, k\}$ and every $p \in \{1, \dots, q\}$, we define the set $U_p^i := \{u_{p,j}^i \mid 0 \leq j \leq k\}$. The set U^i is defined as $U^i := \bigcup_{1 \leq p \leq q} U_p^i$. Now set A is defined as follows.

$$A := \{s^i \mid 1 \leq i \leq k+1\} \cup \{c^{i,j} \mid 1 \leq i \leq j \leq k\} \cup \bigcup_{1 \leq i \leq k} U^i.$$

In the remainder of the section, we construct the set C by adding subsets of triples with specific properties. We start with some simple order constraints. For the sake of convenience, we only give a set B of tuples encoding these constraints. Note that these tuples can also be encoded using triples by introducing a new element that is not allowed to be between any of the elements of those tuples.

$$\begin{aligned}
B := &\{(s^i, u_{p,j}^i) && \mid 1 \leq i \leq k,\ 0 \leq j \leq k,\ 1 \leq p \leq q\} \cup \\
&\{(u_{p,j}^i, u_{r,\ell}^i) && \mid 1 \leq i \leq k,\ p < r \text{ or } p = r \text{ and } j < \ell\} \cup \\
&\{(u_{1,0}^i, s^{i+1}) && \mid 1 \leq i \leq k\} \cup \\
&\{(c^{i,j}, c^{\ell,m}) && \mid i < \ell \text{ or } i = \ell \text{ and } j < m\} \cup \\
&\{(c^{i,k}, c^{i+1,i+1}) && \mid i < k\} \cup \\
&\{(s^{k+1}, c^{1,1})\}
\end{aligned}$$

▶ **Lemma 4.11.** *The reflexive and transitive closure of B forms a cs-tree of width $k+1$.*

In the following, we will present the rest of the triples of the set C. First note that the last two elements of these triples will also be contained as a tuple in the reflexive and transitive closure of B. Thus, they do not contribute any new tuples to $\pi(C)$ and, hence, Lemma 4.11 implies that $\pi(C)$ is a cs-tree of width $k+1$.

We will present the new triples not all at once. Instead we present specific subsets of them. Then we will give properties that are fulfilled by any intermezzo ordering of A that fulfills the constraints of B and all triples presented up to that point. In any of these properties, the ordering σ will be the respective intermezzo ordering of A. We divide this ordering into a *selection phase* where we choose one vertex of every color to be part of the candidate clique. In the *verification phase*, we check whether the chosen vertices indeed form a clique in G. We start with the triples for the selection phase.

$$\begin{aligned}
C^1_{\text{sel}} &:= \{(s^{i+1}, u^i_{p,j}, u^i_{p+1,0}) \mid 1 \le i \le k, 1 \le p < q, 1 \le j \le k\}\\
C^2_{\text{sel}} &:= \{(u^i_{q,j}, s^i, s^{i+1}) \quad\quad \mid 1 \le i \le k, 1 \le j \le k\}\\
C^3_{\text{sel}} &:= \{(u^i_{p,j}, s^{i+1}, c^{1,1}) \quad \mid 1 \le i \le k, 1 \le p \le q, 0 \le j \le k\}
\end{aligned}$$

▶ **Property 1.** *There exist indices $p_1, \dots, p_k \in \{1, \dots, q\}$ such that the prefix σ' of σ ending in $c^{1,1}$ fulfills the following conditions:*

1. *σ' starts with s^1 and does not contain any vertices $c^{i,j}$ with $i \neq 1$ or $j \neq 1$.*
2. *for all $i \in \{1, \dots, k\}$ it holds:*
   a. *vertex s^i and $\bigcup_{r=1}^{p_i-1} U^i_r$ are part of σ',*
   b. *$U^i_{p_i} \cap \sigma' = \{u^i_{p_i,0}\}$,*
   c. *none of the vertices of U^i_r with $r > p_i$ are part of σ'.*

We now present the first triples for the verification phase. They ensure that between certain c-elements only some elements are allowed to be taken. By Property 1 we assume in the following that $p_1, \dots, p_k$ are fixed.

$$\begin{aligned}
C^1_{\text{ver}} &:= \{(x, c^{i,k}, c^{i+1,i+1}) \quad \mid 1 \le i < k,\ x \in A \setminus \{c^{i,k}, c^{i+1,i+1}\}\}\\
C^2_{\text{ver}} &:= \{(u^i_{p,j}, c^{\ell,m}, c^{\ell,m+1}) \mid i \neq \ell \text{ and } i \neq m+1 \text{ or } i = m+1 \text{ and } j \neq \ell \text{ or}\\
&\qquad\qquad\qquad\qquad\qquad\quad i = \ell \text{ and } j \neq m\}
\end{aligned}$$

▶ **Property 2.** *There are at most two elements in σ between $c^{\ell,m}$ and $c^{\ell,m+1}$, namely $u^\ell_{p_\ell,m}$ and $u^{m+1}_{p_{m+1},\ell}$.*

In the next step we want to ensure that $u^\ell_{p_\ell,m}$ and $u^{m+1}_{p_{m+1},\ell}$ have to be taken between $c^{\ell,m}$ and $c^{\ell,m+1}$.

$$\begin{aligned}
C^3_{\text{ver}} := &\{(c^{\ell,m+1}, u^\ell_{p,m-1}, u^\ell_{p,m}) \mid 1 \le \ell \le m < k, 1 \le p \le q\} \cup\\
&\{(c^{\ell,m+1}, u^{m+1}_{p,\ell-1}, u^{m+1}_{p,\ell}) \mid 1 \le \ell \le m < k, 1 \le p \le q\}
\end{aligned}$$

▶ **Property 3.** *The elements $u^\ell_{p_\ell,m}$ and $u^{m+1}_{p_{m+1},\ell}$ have to be between $c^{\ell,m}$ and $c^{\ell,m+1}$ in σ.*

Finally, we have to ensure that $u^\ell_{p_\ell,m}$ and $u^{m+1}_{p_{m+1},\ell}$ can only be taken if $v^\ell_p v^{m+1}_{p_{m+1}} \in E(G)$. This is ensured by the following triples.

$$\begin{aligned}
C^4_{\text{ver}} := &\{(u^{m+1}_{r,\ell}, u^\ell_{p,m}, u^\ell_{p,m+1}) \mid 1 \le \ell \le m < k, 1 \le p \le q, 1 \le r \le q, v^\ell_p v^{m+1}_r \notin E(G)\} \cup\\
&\{(u^\ell_{p,m}, u^{m+1}_{r,\ell}, u^{m+1}_{r,\ell+1}) \ \mid 1 \le \ell \le m < k, 1 \le p \le q, 1 \le r \le q, v^\ell_p v^{m+1}_r \notin E(G)\}
\end{aligned}$$

▶ **Property 4.** *The elements $u^\ell_{p_\ell,m}$ and $u^{m+1}_{p_{m+1},\ell}$ can be between $c^{\ell,m}$ and $c^{\ell,m+1}$ in σ only if $v^\ell_{p_\ell} v^{m+1}_{p_{m+1}} \in E(G)$.*

Using Properties 1–4, we can prove that the described instance of the General Intermezzo problem is a feasible reduction from the MCP.

▶ **Lemma 4.12.** *There is an intermezzo ordering σ for (A, C) if and only if G has a multicolored clique of size k.*

Proof. First assume that there is an intermezzo ordering σ. Let the p_i be chosen as in Property 1. Then, we define the set $K \subseteq V(G)$ as follows: $K := \{v^i_{p_i} \mid 1 \leq i \leq k\}$. Properties 3 and 4 imply that the set K forms a clique in G.

For the other direction, assume that there is a multicolored clique $K = \{v^1_{p_1}, \dots, v^k_{p_k}\}$ in G. We start our intermezzo ordering in s^1. Now we take all the elements of U^1 following their ordering implied by B up to $u^1_{p_1,0}$ and then we take s^2. We repeat this process for all $i \in \{2, \dots, k\}$. Now we take the c-elements following the ordering implied by B. Between $c^{\ell,m}$ and $c^{\ell,m+1}$, we take $u^\ell_{p_\ell,m}$ and $u^{m+1}_{p_{m+1},\ell}$ which is possible due to Property 4 since K forms a clique. Eventually, we take $c^{k,k}$. Now we first take the remaining elements of U^1 in their order in B, followed by the remaining elements of U^2 and so on. It is easy to check that this ordering is an intermezzo ordering. ◀

Theorem 4.9 follows directly from Lemmas 4.3, 4.11, and 4.12, Theorem 4.10 as well as the W[1]-hardness of the Multicolored Clique Problem [12, 21]. Furthermore, Theorem 4.9 and Lemma 4.1 imply the following.

▶ **Theorem 4.13.** *The $\mathcal{L}$-tree recognition problem of Generic Search is* W[1]*-hard if it is parameterized by the number k of leaves of the spanning tree. Furthermore, assuming the Exponential Time Hypothesis, there is no algorithm that solves the problem in time $f(k) \cdot n^{o(k)}$ for any computable function f where n is the number of vertices of the given graph.*

5 Conclusion

We have investigated two problems that extend a partial order to a total order while maintaining certain additional constraints. In the first problem, a spanning tree of a graph G is given, which is supposed to be the $\mathcal{L}$-tree of Generic Search on G. Surprisingly, deciding this problem turned out to be NP-complete, although numerous problems involving Generic Search, such as the associated end vertex problem, are straightforward to solve in polynomial time. This complexity result could be used in the investigation of the second problem. Here we have shown that the General Intermezzo problem cannot be solved in polynomial time even when the Hasse diagram of the given partial order forms a tree. With respect to the width, we were able to specify an XP-algorithm and at the same time show W[1]-hardness.

Several questions remain unanswered. For the GS $\mathcal{L}$-tree problem it is not clear whether the bounds for the tree height in the NP-completeness results are best possible. We conjecture that the problem is easy for height 2 and maybe even 3. Furthermore, we suspect that the NP-completeness also holds for the class of bipartite graphs. While the problem is hard for split graphs, it might be solved efficiently on the subclass of threshold graphs.

Similar questions arise for the (General) Intermezzo problem: We have not shown that the height-bounds for the partial order are best possible. In particular, the bound of 36 in Corollary 4.6 seems very high. Restricting these partial orders in other ways, e.g. lattices or interval orders, could also be used to find tractable instances of the problem. Furthermore, the complexity status of the Intermezzo problem for cs-trees of bounded height remains open.

References

1 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. On the end-vertex problem of graph searches. *Discrete Mathematics & Theoretical Computer Science*, 21(1), 2019. doi:10.23638/DMTCS-21-1-13.

2 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaž Krnc, Nevena Pivač, Robert Scheffler, and Martin Strehler. The recognition problem of graph search trees. *SIAM Journal on Discrete Mathematics*, 35(2):1418–1446, 2021. doi:10.1137/20M1313301.

3 Jesse Beisegel, Ekkehard Köhler, Fabienne Ratajczak, Robert Scheffler, and Martin Strehler. Graph search trees and the intermezzo problem, 2024. arXiv:2404.18645.

4 Jesse Beisegel, Fabienne Ratajczak, and Robert Scheffler. Computing Hamiltonian paths with partial order restrictions, 2024. arXiv:2404.16662.

5 Pierre Charbit, Michel Habib, and Antoine Mamcarz. Influence of the tie-break rule on the end-vertex problem. *Discrete Mathematics & Theoretical Computer Science*, 16(2):57–72, 2014. doi:10.46298/dmtcs.2081.

6 Charles J. Colbourn and William R. Pulleyblank. Minimizing setups in ordered sets of fixed width. *Order*, 1:225–229, 1985. doi:10.1007/BF00383598.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, chapter 22.4: Topological sort, pages 549–552. MIT Press, 2nd edition, 2001.

8 Derek G. Corneil and Richard M. Krueger. A unified view of graph searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008. doi:10.1137/050623498.

9 Derek G. Corneil, Ekkehard Köhler, and Jean-Marc Lanlignel. On end-vertices of Lexicographic Breadth First Searches. *Discrete Applied Mathematics*, 158(5):434–443, 2010. doi:10.1016/j.dam.2009.10.001.

10 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, Cham, 2015. doi:10.1007/978-3-319-21275-3.

11 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. doi:10.2307/1969503.

12 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science*, 410(1):53–61, 2009. doi:10.1016/j.tcs.2008.09.065.

13 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

14 Walter Guttmann and Markus Maucher. Variations on an ordering theme with constraints. In Gonzalo Navarro, Leopoldo Bertossi, and Yoshiharu Kohayakawa, editors, *Fourth IFIP International Conference on Theoretical Computer Science- TCS 2006*, volume 209 of *IFIPAICT*, pages 77–90, Boston, MA, 2006. Springer. doi:10.1007/978-0-387-34735-6_10.

15 Torben Hagerup. Biconnected graph assembly and recognition of DFS trees. Technical Report A 85/03, Universität des Saarlandes, 1985. doi:10.22028/D291-26437.

16 Torben Hagerup and Manfred Nowak. Recognition of spanning trees defined by graph searches. Technical Report A 85/08, Universität des Saarlandes, 1985.

17 John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

18 Ephraim Korach and Zvi Ostfeld. DFS tree construction: Algorithms and characterizations. In Jan van Leeuwen, editor, *Graph-Theoretic Concepts in Computer Science – WG '88*, volume 344 of *LNCS*, pages 87–106, Berlin, Heidelberg, 1989. Springer. doi:10.1007/3-540-50728-0_37.

19 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 105:41–71, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

20 Udi Manber. Recognizing breadth-first search trees in linear time. *Information Processing Letters*, 34(4):167–171, 1990. doi:10.1016/0020-0190(90)90155-Q.

21 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003. doi:10.1016/S0022-0000(03)00078-3.

22 Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.

23 Robert Scheffler. Linearizing partial search orders. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science – WG 2022*, volume 13453 of *LNCS*, pages 425–438, Cham, 2022. Springer. doi:10.1007/978-3-031-15914-5_31.

24 Robert Scheffler. On the recognition of search trees generated by BFS and DFS. *Theoretical Computer Science*, 936:116–128, 2022. doi:10.1016/j.tcs.2022.09.018.

25 Robert Scheffler. Graph search trees and their leaves. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science – WG 2023*, volume 14093 of *LNCS*, pages 462–476, Cham, 2023. Springer. doi:10.1007/978-3-031-43380-1_33.

26 Robert Scheffler. *Ready to Order? On Vertex and Edge Orderings of Graphs.* Doctoral thesis, BTU Cottbus-Senftenberg, 2023. doi:10.26127/BTUOpen-6301.

27 Robert Scheffler. Recognizing LBFS trees of bipartite graphs. *Information Processing Letters*, 186:106483, 2024. doi:10.1016/j.ipl.2024.106483.

28 Edward Szpilrajn. Sur l’extension de l’ordre partiel. *Fundamenta Mathematicae*, 16:386–389, 1930. In French. doi:10.4064/fm-16-1-386-389.

29 William T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory.* Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, Baltimore, London, 1992.

30 Elliot S. Wolk. A note on “The comparability graph of a tree”. *Proceedings of the American Mathematical Society*, 16(1):17–20, 1965. doi:10.1090/S0002-9939-1965-0172274-5.

31 Meibiao Zou, Zhifeng Wang, Jianxin Wang, and Yixin Cao. End vertices of graph searches on bipartite graphs. *Information Processing Letters*, 173:106176, 2022. doi:10.1016/j.ipl.2021.106176.

Minimizing Cost Register Automata over a Field

Yahia Idriss Benalioua ✉
Aix Marseille Univ, CNRS, LIS, Marseille, France

Nathan Lhote ✉
Aix Marseille Univ, CNRS, LIS, Marseille, France

Pierre-Alain Reynier ✉
Aix Marseille Univ, CNRS, LIS, Marseille, France

Abstract

Weighted automata (WA) are an extension of finite automata that define functions from words to values in a given semiring. An alternative deterministic model, called Cost Register Automata (CRA), was introduced by Alur et al. It enriches deterministic finite automata with a finite number of registers, which store values, updated at each transition using the operations of the semiring. It is known that CRA with register updates defined by linear maps have the same expressiveness as WA. Previous works have studied the register minimization problem: given a function computable by a WA and an integer k, is it possible to realize it using a CRA with at most k registers?

In this paper, we solve this problem for CRA over a field with linear register updates, using the notion of linear hull, an algebraic invariant of WA introduced recently by Bell and Smertnig. We then generalise the approach to solve a more challenging problem, that consists in minimizing simultaneously the number of states and that of registers. In addition, we also lift our results to the setting of CRA with affine updates. Last, while the linear hull was recently shown to be computable by Bell and Smertnig, no complexity bounds were given. To fill this gap, we provide two new algorithms to compute invariants of WA. This allows us to show that the register (resp. state-register) minimization problem can be solved in 2-ExpTime (resp. in NExpTime).

2012 ACM Subject Classification Theory of computation → Quantitative automata

Keywords and phrases Weighted automata, Cost Register automata, Zariski topology

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.23

Related Version *Full Version*: `https://arxiv.org/abs/2307.13505` [5]

Funding This work has been partly funded by the QuaSy project (ANR-23-CE48-0008).

1 Introduction

Weighted automata (WA). WA are a quantitative extension of finite state automata and have been studied since the sixties [17]. These automata define functions from words to a given semiring: each transition has a weight in the semiring and the weight of an execution is the product of the weights of the transitions therein; the non-determinism of the model is handled using the sum of the semiring: the weight associated with a word is the sum of the weights of the different executions over this word. Functions realized by weighted automata are called rational series. This fundamental model has been widely studied during the last decades [14]. While some expressiveness results can be obtained in a general framework (such as the equivalence with rational expressions), the decidability status of important problems heavily depends on the considered semiring. Amongst the classical problems of interest, one can mention *equivalence*, *sequentiality* (resp. *unambiguity*), which aims at determining whether there exists an equivalent deterministic (resp. unambiguous) WA, and *minimization*, which aims at minimizing the number of states.

Weighted automata over a field (e.g. the field of rationals $\mathbb{Q}$) enjoy many nice properties: the equivalence of weighted automata is decidable and they can be minimized, and both can be done efficiently (see e.g. [16, Theorem 4.10 and Corollary 4.17 (Chapter III)]). The

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

sequentiality and unambiguity are also decidable, as shown recently in [3, 4], with no complexity bounds however. The most studied semirings which are not fields are the tropical semirings and the semiring of languages, and in both cases equivalence is undecidable (see [9, Section 3] and [6, Theorem 8.4]) and no minimization algorithm is known. Regarding sequentiality, partial decidability results have been obtained for these semirings using the notion of twinning property [7, 15].

Cost register automata (CRA). CRA have been introduced more recently by Alur et al. [1]. A cost register automaton is a deterministic finite state automaton endowed with a finite number of registers storing values from the semiring. The registers are initialized by some values, then at each transition the values are updated using the operations and constants of the semiring. Several fragments of CRA can be considered by restricting the operations allowed. For instance, an easy observation is that WA are exactly CRA with one state (however, one can observe that adding states does not extend expressiveness) and linear updates, *i.e.* updates of the form $X \coloneqq \sum_{i=1}^{k} X_i * c_i$ (intuitively, the new values of the registers only depend linearly on the previous ones). Thus, the model of linear CRA is an alternative to WA which allows to trade non-determinism for registers.

The register minimization problem. As CRA are finite state automata extended with registers storing elements from the semiring, it is natural to aim at minimizing the number of registers used. For a given class $\mathcal{C}$ of CRA, this problem asks, given a WA and a number k, whether there exists an equivalent CRA in $\mathcal{C}$ with at most k registers. From a practical point of view, reducing the number of registers allows to reduce the memory usage, since a register can require unbounded memory. From a theoretical point of view, this problem can be understood as a refinement of the classical problem of minimization of WA. Indeed, a WA can be translated into a linear CRA with a single state, and as many registers as the number of states of the WA. This problem has been studied in [2, 11, 10] for three different models of CRA but in all these works, the additive law of the semiring is not allowed (*i.e.* updates of the form $X \coloneqq Y + Z$ are forbidden). It is worth noticing that [11] encompasses the case of CRA over a field, with only updates of the form $X \coloneqq Y * c$, with c an element of the field.

While the minimal number of registers needed to realise a WA (also known as the *register complexity*) is upper bounded by the number of states of a minimal WA, it may be possible to build an equivalent CRA with fewer registers, but more states. Hence there is a *tradeoff* between the number of states and the number of registers. This leads to the following *state-register minimization problem for* CRA which asks, for a class $\mathcal{C}$ of CRA, given a WA and integers n, k whether an equivalent CRA in $\mathcal{C}$ with n states and k registers can be constructed. In this framework, the classical minimization of WA corresponds to minimizing the number of registers while using only one state, for the class of linear CRA.

The linear hull. As mentioned before, the case of fields is well-behaved to obtain decidability results. In their recent work [3], Bell and Smertnig introduced the notion of *linear hull* of a WA. This notion is inspired by the algebraic theory needed to study polynomial automata but cast into a linear setting. A linear algebraic set (aka linear Zariski closed set) is a finite union of vector subspaces: we later call them *Z-linear sets.* Given a Z-linear set $S = \bigcup_{i=1}^{p} V_i$, the dimension of S is the maximum of the dimensions of the V_is. In this work, the size of the union, p, is called the length of S. Observe that such Z-linear sets were also used in [8] for a category-theoretic approach to minimization of weighted automata over a field. We say such a set is an invariant if it contains the initial vector and is stable under the updates of the automaton. Then the linear hull of a weighted automaton is the strongest Z-linear invariant. In [3], Bell & Smertnig show that computing the linear hull of a minimal automaton allows to decide sequentiality and unambiguity. In addition, in [4], they show that the linear hull can effectively be computed, without providing complexity bounds however.

Contributions. In this work, we deepen the analysis of the linear hull of a WA in order to solve the register and state-register minimization problems for linear CRA. In addition, we also provide new algorithms to compute the linear hull which come with complexity upper bounds, which can be used to derive complexity results for minimization problems as well as for sequentiality and unambiguity of WA. More precisely, our contributions are as follows:

- Firstly, we show that *the register minimization problem* for the class of linear CRA over a field can be solved in 2-ExpTime. To this end, given a rational series f, we show that the minimal number of registers needed to realize f using a linear CRA is exactly the dimension of the linear hull of a minimal WA of f. We then show that the linear hull of a WA can be computed in 2-ExpTime. We show that this complexity drops down to ExpTime for the particular case of commuting transition matrices (which includes the case of a single letter alphabet), with a matching lower bound.
- As a consequence of the computation of the linear hull of a WA and of results proved in [3], we obtain a 2-ExpTime upper bound for the problems of *sequentiality and unambiguity of weighted automata* over a field, closing a question raised in [4].
- Secondly, we prove that the *state-register minimization problem* for linear CRA can be solved in NExpTime. More precisely, given a minimal WA A, we show a correspondence between Z-linear invariants of A and linear CRA equivalent to A. This correspondence maps the length (resp. dimension) of the invariant to the number of states (resp. registers) of the equivalent linear CRA. We then provide a (constructive) NExpTime algorithm that, given a minimal WA and two integers n, k, guesses a well-behaved invariant allowing to exhibit a satisfying equivalent CRA.
- Last, we actually present these results in a more general setting, by considering *affine* CRA, which are a slight extension of linear CRA allowing to use affine maps in the updates of the registers.

Outline of the paper. We present the models of weighted automata and cost register automata in Section 2. We then formally define the two problems we consider, *i.e.* register and state-register minimization problems, and state our main results in Section 3. In Section 4, we introduce the necessary topological notions to define Z-linear/Z-affine set and invariants of weighted automata, and detail our characterizations of the register and state-register complexities of a rational series. Finally in Section 5, we present our algorithms, as well as their consequences in terms of decidability and complexity for the two problems we consider. Omitted proofs and more details for Sections 4 and 5 can be found in the appendix of the full version of this paper [5].

2 Weighted Automata and Cost Register Automata

Basic concepts and notations. An alphabet Σ is a finite set of letters. The set of finite words over Σ will be denoted by Σ^*, the empty word by ϵ and, for two words u and v, uv will denote their concatenation. For two sets X and Y, we denote by $X \times Y$ their cartesian product and by $\pi_X : X \times Y \to X$ and $\pi_Y : X \times Y \to Y$ we denote the canonical projection on X and Y respectively. The set nonnegative integers will be denoted by $\mathbb{N}$. For two integers i, j, we will denote by $[\![i, j]\!]$ the interval of integers between i and j (both included).

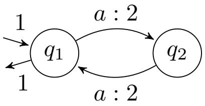


Figure 1 The WA of Example 2.

A *semigroup* $(S, *)$ is a set S together with an associative binary operation $*$. If $(S, *)$ has an identity element e, $(S, *, e)$ is called a *monoid* and if, moreover, every element has an inverse, $(S, *, e)$ is called a *group*. If there is no ambiguity, we will identify algebraic structures with the set that they are defined on. A semigroup (or a monoid/group) is said to be *commutative* if its law is. A sub-semigroup (or submonoid/subgroup) of S is a subset of S that is a semigroup (or a monoid/group). Given $E \subseteq S$, the monoid *generated* by E, denoted $\langle E \rangle$, is the smallest sub-monoid of S containing E.

A *field* $(\mathbb{K}, +, \cdot)$ is a structure where $(\mathbb{K}, +, 0)$ and $(\mathbb{K} \setminus \{0\}, \cdot, 1)$ are commutative groups and multiplication distributes over addition. In this work, we will consider $\mathbb{K}$ as the field of rational numbers $\mathbb{Q}$, or any finite field extension of $\mathbb{Q}$, to perform basic operations in polynomial time. For all $n \in \mathbb{N}$, $\mathbb{K}^n$ is an n-dimensional *vector space* over the field $\mathbb{K}$. We will work with row vectors and apply matrices on the right, and we will identify linear maps (resp. linear forms) with their corresponding matrices (resp. column vectors). The set of n by m matrices over $\mathbb{K}$ will be denoted by $\mathbb{K}^{n\times m}$, and $\mathbb{K}^{1\times n}$ (or simply $\mathbb{K}^n$ when there is no ambiguity) will denote the set of n-dimensional vectors. For any matrix M (resp. vector v), and indices i and j, $M_{i,j}$ (resp. v_i) will denote the value of the entry in the i-th row and the j-th column of M (resp. the i-th entry of v). Matrix transposition will be denoted by M^t. A *vector subspace* of $\mathbb{K}^n$ is a subset of $\mathbb{K}^n$ stable by linear combinations and for all subsets E of $\mathbb{K}^n$, $\mathsf{span}\,(E)$ will denote the smallest vector subspace of $\mathbb{K}^n$ containing E (if E contains a single vector $(x_1, \ldots, x_n)$, $\mathsf{span}\,(E)$ will be denoted by $\mathsf{span}\,(x_1, \ldots, x_n)$).

$\mathbb{K}^n$ can also be seen as an n-dimensional *affine space*. Affine maps $f \colon \mathbb{K}^n \to \mathbb{K}^m$ are maps of the form $f(u) = uf^{(l)} + f^{(a)}$ where $f^{(l)} \in \mathbb{K}^{n\times m}$ and $f^{(a)} \in \mathbb{K}^{1\times m}$. An *affine subspace* A of $\mathbb{K}^n$ is a subset of $\mathbb{K}^n$ of the form $A = p + V$ with $p \in A$ and V a vector subspace of $\mathbb{K}^n$. They are stable by affine combinations (linear combinations with coefficients adding up to 1). For all $E \subseteq \mathbb{K}^n$, $\mathsf{aff}\,(E)$ will denote the smallest affine subspace of $\mathbb{K}^n$ containing E.

Weighted Automata. Let Σ be a finite alphabet and $(\mathbb{K}, +, \cdot)$ be a field.

▶ **Definition 1** (Weighted Automaton). *A* Weighted Automaton *(*WA *for short) of dimension d, on Σ over $\mathbb{K}$, is a triple $\mathcal{R} = (u, \mu, v)$, where $u \in \mathbb{K}^{1\times d}$, $v \in \mathbb{K}^{d\times 1}$ and $\mu \colon \Sigma^* \to \mathbb{K}^{d\times d}$ is a monoid morphism. We will call u and v the* initial *and* terminal *vectors respectively and $\mu(a)$, for $a \in \Sigma$, will be called a* transition matrix. *A* WA *realizes a* formal power series *over Σ^* with coefficients in $\mathbb{K}$ (a function from Σ^* to $\mathbb{K}$) defined, for all $w \in \Sigma^*$, by $[\![\mathcal{R}]\!](w) = u\mu(w)v$. Any series that can be realized by a* WA *will be called* rational.

WA also have a representation in terms of finite-state automata, in which transitions are equipped with weights. We then say that a WA is sequential (resp. unambiguous) when its underlying automaton is. Formally, we say that a WA $\mathcal{R} = (u, \mu, v)$ is *sequential* when u has a single non-zero entry and, for each letter a, and each index i, there is at most one index j such that $\mu(a)_{i,j} \neq 0$.

▶ **Example 2.** We consider the WA, on the alphabet $\{a\}$ and over the field of real numbers, $\mathcal{R} = (u, \mu, v)$ with $u = (1, 0)$, $v = (1, 0)^t$, and $\mu(a) = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$. One can verify that the function realized by this WA maps the word a^n to 2^n if n is even, and to 0 otherwise. It can be represented graphically by the automaton depicted on Figure 1.

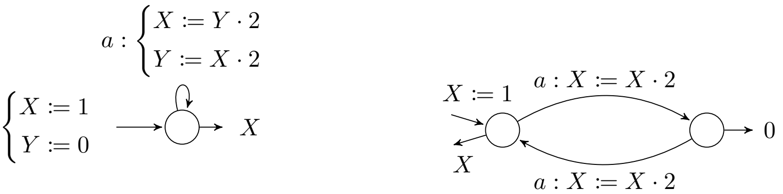

$a : \begin{cases} X := Y \cdot 2 \\ Y := X \cdot 2 \end{cases}$
$\begin{cases} X := 1 \\ Y := 0 \end{cases}$
X
$X := 1$
$a : X := X \cdot 2$
X
$a : X := X \cdot 2$
0

Figure 2 Two CRA detailed in Example 5. Registers are denoted by letters X, Y.

A WA realizing a rational series f is said to be *minimal* if its dimension is minimal among all the WA realizing f. We also have the following characterization of minimal WA (see [16, Proposition 4.8 (Chapter III)]):

▶ **Proposition 3.** *Let $\mathcal{R} = (u, \mu, v)$ be a d-dimensional* WA *and let $\mathsf{LR}(\mathcal{R}) = u\mu(\Sigma^*) = \{u\mu(w) \mid w \in \Sigma^*\}$ be its (left)* reachability set *and $\mathsf{RR}(\mathcal{R}) = \mu(\Sigma^*)v$ be its right reachability set.*

$\mathcal{R}$ is a minimal WA *if and only if $\mathsf{span}(\mathsf{LR}(\mathcal{R})) = \mathbb{K}^{1\times d}$ and $\mathsf{span}(\mathsf{RR}(\mathcal{R})) = \mathbb{K}^{d\times 1}$.*

Expressions, substitutions and Cost Register Automata. For a field $(\mathbb{K}, +, \cdot)$ and a finite set of variables $\mathcal{X}$ disjoint from $\mathbb{K}$, let $\mathsf{Exp}(\mathcal{X})$ denote the set of expressions generated by the grammar $e ::= k \mid X \mid e + e \mid e \cdot e$, where $k \in \mathbb{K}$ and $X \in \mathcal{X}$. A *substitution* over $\mathcal{X}$ is a map $s\colon \mathcal{X} \to \mathsf{Exp}(\mathcal{X})$. It can be extended to a map $\mathsf{Exp}(\mathcal{X}) \to \mathsf{Exp}(\mathcal{X})$ by substituting each variable X in the expression given as an input by $s(X)$. By identifying s with its extension, we can compose substitutions. We call *valuations* the substitutions of the form $v\colon \mathcal{X} \to \mathbb{K}$. The set of substitutions over $\mathcal{X}$ will be denoted by $\mathsf{Sub}(\mathcal{X})$ and the set of valuations $\mathsf{Val}(\mathcal{X})$.

▶ **Definition 4** (Cost Register Automaton). *A* cost register automaton *(*CRA *for short), on the alphabet Σ over the field $\mathbb{K}$, is a tuple $\mathcal{A} = (Q, q_0, \mathcal{X}, v_0, o, \delta)$ where Q is a finite set of* states, *$q_0 \in Q$ is the initial state, $\mathcal{X}$ is a finite set of registers (variables), $v_0 \in \mathsf{Val}(\mathcal{X})$ is the registers' initial valuation, $o\colon Q \to \mathsf{Exp}(\mathcal{X})$ is the output function, and $\delta\colon Q \times \Sigma \to Q \times \mathsf{Sub}(\mathcal{X})$ is the transition function. We will denote by $\delta_Q := \pi_Q \circ \delta$ the transition function of the underlying automaton of the* CRA *and $\delta_{\mathcal{X}} := \pi_{\mathsf{Sub}(\mathcal{X})} \circ \delta$ its register update function.*

$\mathcal{A}$ computes a function $[\![\mathcal{A}]\!]\colon \Sigma^* \to \mathbb{K}$ defined as follows: the configurations of $\mathcal{A}$ are pairs $(q, v) \in Q \times \mathsf{Val}(\mathcal{X})$. The run of $\mathcal{A}$ on a word $w = a_1 \ldots a_n \in \Sigma^*$ is the sequence of configurations $(q_i, v_i)_{i\in[\![0,n]\!]}$ where, q_0 is the initial state, v_0 is the initial valuation and, for all $i \in [\![1, n]\!]$, $q_i = \delta_Q(q_{i-1}, a_i)$ and $v_i = v_{i-1} \circ \delta_{\mathcal{X}}(q_{i-1}, a_i)$. We then define $[\![\mathcal{A}]\!](w) = v_n(o(q_n))$.

δ can be extended to words by setting, for all $q \in Q$, $\delta(q, \epsilon) = (q, id_{\mathcal{X}})$, where $id_{\mathcal{X}}$ is the substitution such that $id_{\mathcal{X}}(X) = X$ for all $X \in \mathcal{X}$, and, for all $a \in \Sigma$ and $w \in \Sigma^*$, $\delta_Q(q, aw) = \delta_Q(\delta_Q(q, a), w)$ and $\delta_{\mathcal{X}}(q, aw) = \delta_{\mathcal{X}}(q, a) \circ \delta_{\mathcal{X}}(\delta_Q(q, a), w)$. We then have

$$[\![\mathcal{A}]\!](w) = v_0 \circ \delta_{\mathcal{X}}(q_0, w)(o(\delta_Q(q_0, w)))$$

▶ **Example 5** (Example 2 continued). Two CRA are depicted on Figure 2. They are both on the alphabet $\{a\}$ and over the field of real numbers, and both realize the same function as the WA considered in Example 2.

An expression is called *linear* if it has the form $\sum_{i=1}^{k} \alpha_i X_i$, for some family of $\alpha_i \in \mathbb{K}$ and $X_i \in \mathcal{X}$, and if it has the form $\sum_{i=1}^{k} \alpha_i X_i + \beta$, for some $\beta \in \mathbb{K}$, it is called *affine.* We will denote by $\mathsf{Exp}_\ell(\mathcal{X})$ (resp. $\mathsf{Exp}_a(\mathcal{X})$) the set of linear (resp. affine) expressions.

▶ **Definition 6** (Linear/Affine CRA). *A* CRA $\mathcal{A} = (Q, q_0, \mathcal{X}, v_0, o, \delta)$ *is called* linear *if,* $\delta_{\mathcal{X}}(q, a)(X) \in \mathsf{Exp}_\ell(\mathcal{X})$ *and* $o(q) \in \mathsf{Exp}_\ell(\mathcal{X})$*, for all* $q \in Q, a \in \Sigma$ *and* $X \in \mathcal{X}$*, and if* $\delta_{\mathcal{X}}(q, a)(X) \in \mathsf{Exp}_a(\mathcal{X})$ *and* $o(q) \in \mathsf{Exp}_a(\mathcal{X})$*, the* CRA *is called* affine.

Linear CRA are a particular case of affine CRA and, given an affine CRA it is always possible to define an equivalent linear CRA using one more register with a constant value of 1 to realize affine register updates in a linear way, thus :

▶ Remark 7. Linear and affine CRA have the same expressiveness.

The added cost of a register will however become relevant when we will consider minimization problems in the next sections.

Observe that we can assume that $\mathcal{X} = \{X_1, \dots, X_k\}$ is ordered, and identify any linear expression $e = \sum_{i=1}^{k} \alpha_i X_i$ (with the α_i not present in the expression assumed to be 0) with the linear form $\underline{e}\colon \mathbb{K}^k \to \mathbb{K}$ defined by the column vector $(\alpha_1, \dots, \alpha_k)^t$. We can then identify any linear substitution $s\colon \mathcal{X} \to \mathsf{Exp}_\ell(\mathcal{X})$ with the linear map $\underline{s}\colon \mathbb{K}^k \to \mathbb{K}^k$ defined by the block matrix $(\underline{s(X_1)}|\cdots|\underline{s(X_k)})$, and we can identify any valuation $v\colon \mathcal{X} \to \mathbb{K}$ with the vector $\underline{v} = (v(X_1), \cdots, v(X_k))$ of the vector space $\mathbb{K}^k$.

In the following, we will drop the underline notation and make the identifications implicitly.

Thanks to these observations, the registers of a linear CRA and their updates can be characterized by the values of the vector associated with v_0, and the linear maps associated with the $\delta_{\mathcal{X}}(q, a)$ and $o(q)$, for all $q \in Q$ and $a \in \Sigma$, and we can check that

$$[\![\mathcal{A}]\!](w) = v_0 \ \delta_{\mathcal{X}}(q_0, w) \ o(\delta_Q(q_0, w))$$

We can also identify affine expressions with affine forms and affine substitutions with affine maps to simplify dealing with affine CRA. We define and use these identifications in Appendix A.2 of the full version of this paper [5].

▶ **Proposition 8** ([1]). *There is a bijection between* WA *and linear* CRA *with a single state.*

Given a WA, one can build an equivalent CRA with as many registers as states of the WA: for each letter a, the transition matrix $\mu(a)$ can be interpreted as a (linear) substitution, associated with the self-loop of label a. The converse easily follows from the previous observations when the CRA has a single state.

▶ **Example 9** (Example 2 continued). The CRA depicted on the left of Figure 2 is obtained by the translation of the WA of Figure 1 into CRA with a single state.

▶ Remark 10. Sequential WA are exactly linear CRA with a single register.

Indeed, both sequential WA and linear CRA with only one register are deterministic finite automata that can also store a single value updated at each transition using only products. They can then be identified.

3 Problems and Main Results

▶ **Definition 11** (Register minimization problem). *Given a class* $\mathcal{C}$ *of* CRA*, we ask:*

- ***Input:*** *a rational series* f *realized by a given* WA*, and an integer* $k \in \mathbb{N}$
- ***Question:*** *Does there exist a* CRA *realizing* f *in the class* $\mathcal{C}$ *with at most* k *registers?*

We will show this problem is decidable for the classes of linear and affine CRA:

▶ **Theorem 12.** *The register minimization problem is decidable for the classes of linear and affine* CRA *in 2-*ExpTime*. Furthermore, the algorithm exhibits a solution when it exists.*

For a rational series f, the minimal number of registers needed to realize f using CRA in some class $\mathcal{C}$ is called its *register complexity with respect to class $\mathcal{C}$*. Dually, if one wants to minimize the number of states, then we know we can always build a linear (hence affine) CRA with a single state (Proposition 8). A more ambitious goal is to try to reduce simultaneously the number of states and of registers, in some given class $\mathcal{C}$ of CRA. Observe that, in general, there is no CRA with minimal numbers of both states and registers (see Example 5). Given a rational series f, we say that a pair (n, k) is *optimal* if f can be realized by a CRA in class $\mathcal{C}$ with n states and k registers and no CRA of $\mathcal{C}$ realizing f with at most n states can have strictly less than k registers and vice-versa.

Formally, we call the *state-register complexity with respect to class $\mathcal{C}$* of a rational series f, the set of optimal pairs of integers (n, k).

This leads to the definition of a second minimization problem:

▶ **Definition 13** (State-Register minimization problem)**.** *Given a class $\mathcal{C}$ of* CRA*, we ask:*

- ***Input:*** *a rational series f realized by a given* WA*, and two integers $n, k \in \mathbb{N}$*
- ***Question:*** *Does there exist a* CRA *realizing f in the class $\mathcal{C}$ with at most n states and at most k registers?*

In the sequel, we solve this problem for linear and affine CRA:

▶ **Theorem 14.** *The state-register minimization problem is decidable for the classes of linear and affine* CRA *in NExpTime. Furthermore, the algorithm exhibits a solution when it exists.*

▶ Remark 15. The complexities we give are valid for fields where it is possible to perform elementary operations efficiently (e.g. $\mathbb{Q}$). See Remark 40 for a more detailed discussion on the matter.

4 Characterizing the state-register complexity using invariants of WA

4.1 Zariski topologies and invariants of WA

Let $\mathbb{K}$ be a field and $n \in \mathbb{N}$. The *Zariski topology* on $\mathbb{K}^n$ is defined as the topology whose closed sets are the sets of common roots of a finite collection of polynomials of $\mathbb{K}[X_1, \ldots, X_n]$. A linear version of this topology, called the *linear Zariski topology*, was introduced by Bell and Smertnig in [3]. Its closed sets, which we will call *Z-linear sets*, are finite unions of vector subspaces of $\mathbb{K}^n$.

A set $S \subseteq \mathbb{K}^n$ is called *irreducible* if, for all closed sets C_1 and C_2, such that $S \subseteq C_1 \cup C_2$, we have either $S \subseteq C_1$ or $S \subseteq C_2$. The Zariski topologies defined above are Noetherian topologies in which every closed set can be written as a finite union of irreducible components. We then define the *dimension* of a Z-linear set as the maximum dimension of its irreducible components and their number will be called its *length.*

For a set $S \subseteq \mathbb{K}^n$, $\overline{S}^{\ell}$ will denote its closure in the linear Zariski topology. In this topology, closed irreducible sets are vector subspaces of $\mathbb{K}^n$ and linear maps are continuous and closed maps (mapping closed sets to closed sets). In particular, for all $S \subseteq \mathbb{K}^n$ and linear map $f : \mathbb{K}^n \to \mathbb{K}^n$,$\overline{f(S)}^{\ell} = f(\overline{S}^{\ell})$. Moreover, if $S \subseteq \mathbb{K}^n$ is irreducible and $f : \mathbb{K}^n \to \mathbb{K}^n$ is continuous, then $f(S)$ is irreducible. These properties will be used implicitly in the following (see [3, Lemma 3.5] for more details and references).

We will also define an affine version of this topology that enjoy the same properties in Subsection 4.4.

▶ **Definition 16.** *Let Σ be a finite alphabet and let $\mathcal{R} = (u, \mu, v)$ be a d-dimensional* WA *on Σ over $\mathbb{K}$. A subset $I \subseteq \mathbb{K}^d$ is called an* invariant *of $\mathcal{R}$ if $u \in I$ and, for all $w \in I$ and $a \in \Sigma$, $w\mu(a) \in I$. For two invariants I_1 and I_2, we say that I_1 is* stronger *than I_2 if $I_1 \subseteq I_2$. In particular, the strongest invariant of $\mathcal{R}$ is its* reachability set $\mathsf{LR}(\mathcal{R}) = u\mu(\Sigma^*)$.

An invariant that is also a Z-linear set will be called a Z-linear *invariant. The strongest Z-linear invariant of $\mathcal{R}$ is the closure of* $\mathsf{LR}(\mathcal{R})$ *in the linear Zariski topology (which is well-defined since the topology is Noetherian).*

▶ **Example 17** (Example 2 continued)**.** The reachability set of the WA considered in Example 2 is $\mathsf{LR}(\mathcal{R}) = \{(2^{2n}, 0) \mid n \in \mathbb{N}\} \cup \{(0, 2^{2n+1}) \mid n \in \mathbb{N}\}$. Its strongest Z-linear invariant is then the union of the two coordinate axes of the plane $\overline{\mathsf{LR}(\mathcal{R})}^{\ell} = \mathsf{span}(1,0) \cup \mathsf{span}(0,1)$.

Indeed, the inclusion $\subseteq$ comes from the fact that $u = (1,0) \in \mathsf{span}(1,0)$ and $\mathsf{span}(1,0) \cup \mathsf{span}(0,1)$ is stable by multiplication by $\mu(a)$ and the inclusion $\supseteq$ comes from the fact that, for the linear Zariski topology, $\{(1,0)\}$ is dense in $\mathsf{span}(1,0)$ and $\{(0,2)\}$ is dense in $\mathsf{span}(0,1)$.

▶ Remark 18. In the previous example, the strongest Z-linear invariant is actually the strongest algebraic invariant (*i.e.* closed in the Zariski topology). Of course, this is not always the case.

The Z-linear invariants of two WA realizing the same function do not necessarily coincide but, since $\mathbb{K}$ is a field, it is well-known that for every rational series f, there exists a (computable) minimal WA realizing f that is unique up to similarity in the following sense (see [16, Proposition 4.10 (Chapter III)]):

▶ **Definition 19.** *Let $\mathcal{R} = (u, \mu, v)$ and $\mathcal{R}' = (u', \mu', v')$ be two d-dimensional* WA *over $\mathbb{K}$.*

$\mathcal{R}$ and $\mathcal{R}'$ are said to be similar *if there exists an invertible (change of basis) matrix $P \in \mathbb{K}^{d \times d}$ such that $u' = uP$, $\mu'(a) = P^{-1}\mu(a)P$ for all $a \in \Sigma$ and $v' = P^{-1}v$.*

▶ Remark 20. The Z-linear invariants of two similar WA $\mathcal{R}$ and $\mathcal{R}'$ only differ by a change of basis. i.e. there is a bijection between the Z-linear invariants of $\mathcal{R}$ and those of $\mathcal{R}'$ that, in particular, preserves the length and dimension.

4.2 Strongest invariants and characterization

The notion of strongest Z-linear invariant was introduced by Bell and Smertnig in [3], under the name "linear hull". They showed, in [4], that it is computable and can be used to decide whether a WA is equivalent to a deterministic (or an unambiguous) one.

▶ **Theorem 21** ([3, Theorem 1.3])**.** *A rational series f can be realized by a sequential* WA *iff the strongest Z-linear invariant of a minimal* WA *realizing f has dimension at most 1.*

The following result generalizes this theorem by linking linear CRA to Z-linear invariants. It constitutes the key characterization that will allow us to solve the minimization problems.

▶ **Theorem 22** (Characterization)**.** *Let f be a rational series. Then f can be realized by a linear* CRA *with n states and k registers iff there exists a minimal* WA *realizing f that has a Z-linear invariant of length at most n and dimension at most k.*

As we will see in Subsection 4.4, this theorem can also be extended to affine CRA.

Observe that, thanks to Remark 20, the property of the above characterization is actually valid for *every* minimal WA realizing f. Moreover, since the dimension of the strongest Z-linear invariant is minimal, finding this dimension allows to solve the register minimization problem for linear CRA. This is formalized in the following result, which generalizes Theorem 21 thanks to Remark 10.

▶ **Corollary 23.** *The register complexity of a rational series f w.r.t. the class of linear* CRA *is the dimension of the strongest Z-linear invariant of any minimal* WA *realizing f.*

An immediate consequence of this result is that computing the strongest invariant allows to decide the register minimization problem.

▶ **Example 24** (Example 2 continued)**.** As we have seen in Example 17, $\overline{\mathsf{LR}(\mathcal{R})}^{\ell}$ is 1-dimensional and has two irreducible components, thus $[\![\mathcal{R}]\!]$ can be realized by a CRA with two states and one register (depicted on the right of Figure 2).

4.3 Invariants of minimal WA and correspondence with CRA

▶ **Proposition 25.** *Let $\mathcal{R}$ be a* WA *realizing a rational series f. If $\mathcal{R}$ has a Z-linear invariant of length n and dimension k, then every minimal* WA *realizing f has a Z-linear invariant of length $\leq n$ and dimension $\leq k$.*

Thanks to Remark 20, it suffices to show the existence of one minimal WA verifying the proposition, since they are all similar. It is known (see Proposition 3) that a minimal WA can be obtained from a WA by alternating between two constructions which reduce the dimension to make it match the one of the span of the left (resp. right) reachability set. The result then follows from the next lemma, which states that both constructions decrease the length and dimension of the invariants. We prove it by considering an adequate change of basis, and verifying that it preserves invariants.

▶ **Lemma 26.** *Let $\mathcal{R}$ be a* WA *realizing a rational series f, let $S_{\mathcal{R}}$ be a Z-linear invariant of $\mathcal{R}$ of length n and dimension k and let $r = \dim(\mathsf{span}\,(\mathsf{LR}\,(\mathcal{R})))$. We can construct an r-dimensional* WA *$\mathcal{R}'$ realizing f, with a Z-linear invariant $S_{\mathcal{R}'}$ of length $\leq n$ and dimension $\leq k$. The same holds with $r = \dim(\mathsf{span}\,(\mathsf{RR}\,(\mathcal{R})))$.*

The next proposition allows to go from Z-linear invariants of WA to CRA. This construction builds on the one of [3, Lemma 3.13], in which they build an equivalent WA from the strongest Z-linear invariant of a WA. We show that an analogous construction is valid for any Z-linear invariant, and that we can use states of CRA to represent the different irreducible components of the invariant, thus reducing the number of registers used to the dimension of the invariant.

▶ **Proposition 27.** *Let $\mathcal{R}$ be a* WA*. If $\mathcal{R}$ has a Z-linear invariant of length n and dimension k, then there exists a linear* CRA *$\mathcal{A}$, with n states and k registers, such that $[\![\mathcal{A}]\!] = [\![\mathcal{R}]\!]$.*

The next proposition shows the converse direction, from CRA to invariants of WA. The construction is the classical one from CRA to WA. The existence of the adequate invariant follows from the determinism of the CRA which ensures that in any reachable configuration, only coordinates associated with the reachable state of the CRA can be non-zero.

▶ **Proposition 28.** *Let $\mathcal{A}$ be a linear* CRA*. If $\mathcal{A}$ has n states and k registers, then there exists a* WA *$\mathcal{R}$, with a Z-linear invariant of length n and dimension k, such that $[\![\mathcal{A}]\!] = [\![\mathcal{R}]\!]$.*

Using the three previous propositions, we can finally prove the main characterization:

Proof of Theorem 22. Given a linear CRA with n states and k registers, we can construct, thanks to Proposition 28, an equivalent WA with a Z-linear invariant of length n and dimension k. Then the desired minimal WA exists thanks to Proposition 25.

Reciprocally, applying the construction of Proposition 27 to any minimal WA gives the desired linear CRA. ◀

As we will discuss in the next subsection below, the three propositions we used for this proof can also be adapted to yield the same result for affine CRA.

4.4 Z-affine invariants and affine CRA

All the results of Section 4 can actually be extended to affine CRA using the *affine Zariski topology* instead of the linear one. It is a slight generalization of the linear Zariski topology where closed sets, called *Z-affine* sets, are finite unions of affine spaces instead of vector spaces, with lengths and dimensions defined like in the linear case. It is still a Noetherian topology coarser than the Zariski topology, affine maps are continuous and closed maps in this topology and, more broadly, it enjoys the same properties as the linear Zariski topology we considered throughout this section. For a set $S \subseteq \mathbb{K}^n$, we will denote by $\overline{S}^a$ it closure in the affine Zariski topology and, similarly to the linear case, for a WA $\mathcal{R} = (u, \mu, v)$, we will call any invariant of $\mathcal{R}$ that is a Z-affine set a *Z-affine invariant* of $\mathcal{R}$. Of course, the strongest Z-affine invariant of $\mathcal{R}$ is still the closure of its reachability set i.e. its "affine hull" $\overline{\mathsf{LR}(\mathcal{R})}^a$ and Remark 20 is still true for Z-affine invariants.

We obtain the same characterization of Theorem 22 in the affine setting :

▶ **Theorem 29** (Characterization). *Let f be a rational series. Then f can be realized by an affine* CRA *with n states and k registers iff there exists a minimal* WA *realizing f that has a Z-affine invariant of length at most n and dimension at most k.*

We can show that Propositions 25, 27 and 28 are also true if we replace Z-linear invariants by Z-affine ones and linear CRA by affine ones. So, the proof of Theorem 29 remains the same as Theorem 22. All the details can be found in Appendix A.2 of the full version of this paper [5].

Of course, this theorem has the same consequences of its linear counterpart and we obtain an affine version of Corollary 23

▶ **Corollary 30.** *The register complexity of a rational series f w.r.t. the class of affine* CRA *is the dimension of the strongest Z-affine invariant of any minimal* WA *realizing f.*

Working in the affine Zariski topology instead of the linear one can decrease the dimension of the strongest invariant by one, as shown in the following example.

▶ **Example 31.** On the alphabet $\Sigma = \{a\}$, let $\mathcal{R} = (u, \mu, v)$, where $u = (1, 2)$, $\mu(a) = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$ and $v = (1, 0)^t$, be a WA (over $\mathbb{R}$) realizing the rational series f defined by $f(a^n) = \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$.

The reachability set of $\mathcal{R}$ is $\mathsf{LR}(\mathcal{R}) = \left\{ \left(\sum_{i=0}^{n} 2^i, 2^{n+1}\right) \,\middle|\, n \in \mathbb{N} \right\}$.

For the linear Zariski topology, $\mathsf{LR}(\mathcal{R})$ is dense in $\mathbb{R}^2$. So the strongest Z-linear invariant $\overline{\mathsf{LR}(\mathcal{R})}^\ell = \mathbb{R}^2$ is two-dimensional. However, note that, for all $(x, y) \in \mathsf{LR}(\mathcal{R})$, $y = x + 1$. So, by an argument of density in the affine Zariski topology, the strongest Z-affine invariant $\overline{\mathsf{LR}(\mathcal{R})}^a$ is the affine line $y = x + 1$, which is one-dimensional.

Thus, in the case where the dimensions of the affine and linear hulls doesn't match, using affine CRA instead of linear CRA can allow to save one register :

▶ **Example 32** (Example 31 continued). The two CRA depicted on Figure 3 both realize the function of Example 31. On the left we have a linear CRA with two registers and, on the right, an affine CRA with only one register. The characterization theorems show that both have the minimal number of registers for their respective classes of CRA.

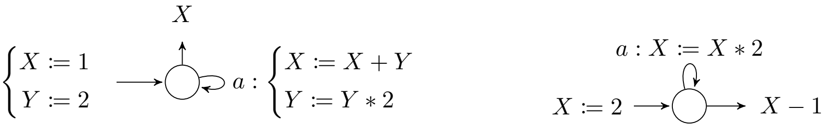

$\begin{cases} X := 1 \\ Y := 2 \end{cases}$
X
$a : \begin{cases} X := X + Y \\ Y := Y * 2 \end{cases}$
$a : X := X * 2$
$X := 2$
$X - 1$

Figure 3 Two CRA detailed in Example 32.

5 Algorithms and complexity for the minimization problems

We present two original algorithms to solve the minimization problems we consider. It is worth observing the difference between the two characterizations we have obtained: while the register complexity can be computed from a canonical object (the strongest Z-linear invariant of the WA), the state-register complexity is based on the existence of a particular Z-linear invariant. This explains why we derive a non-deterministic procedure for the latter, and a deterministic for the former.

5.1 Algorithm for the state-register minimization problem

We provide here a NExpTime algorithm for the state-register minimization problem, hence proving Theorem 14. The algorithm runs in NPTime in n, k, and the size of the automaton. The fact that n is given in binary explains the exponential discrepancy.

Small representations of Z-affine sets. Let $\mathcal{R} = (u, \mu, v)$ be a WA of dimension d over an alphabet Σ. Let $L = A_1 \cup \cdots \cup A_n$ be a Z-affine set of length n of $\mathbb{K}^d$.

An $\mathcal{R}$-representation R of L is a set of n finite sets of words $S_1, \ldots, S_n$ such that $\mathsf{aff}\left(\{u\mu(w) | \; w \in S_i\}\right) = A_i$ for all $i \in \{1, \cdots, n\}$. The *size* of R is the sum of the lengths of all words appearing in R. The following key lemma shows that all Z-affine invariants of $\mathcal{R}$ have small $\mathcal{R}$-representations, up to considering stronger invariants.

▶ **Lemma 33.** *Let $\mathcal{R}$ be a* WA*. Let I be a Z-affine invariant of $\mathcal{R}$ of length n and dimension k. There exists an $\mathcal{R}$-representation R of size $\leq n^2k^2$ of a Z-affine invariant $J \subseteq I$, of dimension $\leq k$ and length $\leq n$.*

This property allows to derive the non-deterministic algorithm. First, minimization of a WA over a field can be performed in polynomial time (see e.g. [16, Corollary 4.17]). Then, let $\mathcal{R}$ be a minimal WA and let k, n be positive integers. From Lemma 33, we know that a Z-affine invariant of dimension k and length n can be represented in size $O(k^2n^2)$ (up to finding a stronger invariant with smaller dimension and length). The algorithm works thusly: first step is to guess an $\mathcal{R}$-representation R of a Z-affine set. The second step is to check that R represents an invariant, which can be done easily using basic linear algebra. From this one can compute an affine CRA with k registers and n states. Moreover, if we require that R is Z-linear, we obtain a linear CRA. If R is not an invariant, the computation rejects. Note that different accepting computations may give rise to different invariants and thus different CRAs.

5.2 Algorithm for the computation of Z-affine invariants

We describe a deterministic procedure which, given a WA $\mathcal{R}$ and an integer c, returns a Z-affine invariant J which is stronger that any Z-affine invariant I of $\mathcal{R}$ of length at most c. When c is chosen large enough, this procedure returns the strongest Z-affine invariant of $\mathcal{R}$. A similar procedure works as well for the computation of Z-linear invariants.

Algorithm 1 Computing a Z-affine invariant.

```
Require: A WA $\mathcal{R} = (u, \mu, v)$ of dimension $d$, an integer $c$
Ensure: A Z-affine invariant $J$ of $\mathcal{R}$ stronger than $I_c(\mathcal{R})$
1: $J \coloneqq \{u\}$
2: while $J$ is not an invariant of $\mathcal{R}$ do
3:     Pick some component $A$ of $J$, and some matrix $M$ of $\mathcal{R}$ s.t. $A \cdot M \not\subseteq J$
4:     $J \coloneqq J \cup A \cdot M$
5:     if length$(J) > c^d$ then
6:         $J \coloneqq$ REDUCE$(J)$
7:     end if
8: end while
9: return $J$
```

Intuitively, this procedure will build a Z-affine set J as follows: it starts with a set containing only the initial vector of $\mathcal{R}$, and incrementally extends it until it forms an invariant. During this process, it should ensure that J is included in every Z-affine invariant I of $\mathcal{R}$ of length at most c. This relies on the following easy observation: if such an invariant I contains at least $c + 1$ points on the same affine line (*i.e.* a 1-dimensional affine space, denoted D), then I must have a component that contains D. Indeed, as I has length at most c, one of its components contains two such points. As this component is irreducible, it is an affine subspace, hence contains D. This reasoning can be lifted to higher dimensions as follows.

Given a WA $\mathcal{R}$, and $c \in \mathbb{N}$, we denote by $I_c(\mathcal{R}) = \bigcap_{\mathrm{length}(I) \leq c} I$ the intersection of all Z-affine invariants of $\mathcal{R}$ with at most c components.

▶ **Lemma 34.** *Let $\mathcal{R}$ be a* WA *and let $c, k \in \mathbb{N}$. Let $A_1, \ldots, A_{c^k+1} \subseteq I_c(\mathcal{R})$ be affine spaces such that: for any $P \subseteq [\![1, c^k + 1]\!]$ with $|P| \geq c^{k-1} + 1$,* $\mathsf{aff}\left(\cup_{i \in P} A_i\right)$ *has dimension k. Then* $\mathsf{aff}\left(\cup_{i \in [\![1,c^k+1]\!]} A_i\right) \subseteq I_c(\mathcal{R})$*.*

Using this lemma, we derive an effective procedure to simplify a Z-affine set $J = A_1 \cup \cdots \cup A_{c^d+1}$ by "merging" two components. We denote by REDUCE(J) the resulting set.

▷ Claim 35. Let $\mathcal{R} = (u, \mu, v)$ be a WA of dimension d, let $c \in \mathbb{N}$. Let $A_1, \ldots, A_{c^d+1} \subseteq I_c(\mathcal{R})$ be affine spaces. One can find $1 \leq i < j \leq c^d + 1$ such that $\mathsf{aff}\left(A_i \cup A_j\right) \subseteq I_c(\mathcal{R})$, in time $O(c^{p(d)})$, for some fixed polynomial p.

▶ **Theorem 36.** *Algorithm 1 is correct and terminates in time $O(c^{p(d)})$.*

Proof. Let us first discuss termination. Because of line 5-7, the length of J is at most $c^d + 1$. Moreover J is an increasing Z-affine set, thus its value can be modified at most $(d+1) \cdot (c^d + 1)$ times, thus from Claim 35 the algorithm terminates in time $O(c^{p(d)})$.

We now discuss correctness. We need to show that J is stronger than $I_c(\mathcal{R})$. Initially, this holds. Moreover, if $A \subseteq I_c(\mathcal{R})$ is an affine set, then for any $M \in \mu(\Sigma), A \cdot M \subseteq I_c(\mathcal{R})$, since $I_c(\mathcal{R})$ is invariant. Thus, line 4 preserves the property that J is stronger than $I_c(\mathcal{R})$. Using Claim 35, the REDUCE subroutine also preserves this property, since it only merges components whose affine span is contained in $I_c(\mathcal{R})$. ◀

5.3 Complexity of the register minimization problem

In order to compute the strongest Z-linear and Z-affine invariants of a WA using Algorithms 1, it is sufficient to be able to bound their lengths. The following result gives such bounds.

▶ **Theorem 37.** *Let $\mathcal{R} = (u, \mu, v)$ be a d-dimensional* WA *on a finite alphabet Σ. We have the following upper bounds :*

- *The lengths of $\overline{\mathsf{LR}(\mathcal{R})}^{\ell}$ and $\overline{\mathsf{LR}(\mathcal{R})}^{a}$ are at most doubly-exponential in d.*
- *If $\langle\mu(\Sigma)\rangle$ is commutative (*e.g.* Σ is unary), then the length of $\overline{\mathsf{LR}(\mathcal{R})}^{\ell}$ is at most exponential in d.*

We also have the following lower bound (which also hold for WA *over a unary alphabet):*

- *For all $d > 0$, there exist a d-dimensional* WA *having strongest Z-linear and Z-affine invariants with lengths exponential in d.*

Proof sketch. The first item is shown in [4], where the authors sketch a proof of a double-exponential upper bound on the length of the strongest Z-linear invariant of a WA, using tools from algebraic geometry, which holds for $\mathbb{Q}$ in particular and for any field $\mathbb{K}$ where there is a double-exponential bound on the maximal order of finite groups of invertible matrices (see [4, Proposition 48 and Remark 41]). Their proof can be adapted to $\overline{\mathsf{LR}(\mathcal{R})}^{a}$. The proof of the second item relies on basic linear algebra and on results and ideas from [4] for invertible matrices (see [4, Lemma 13 and Theorem 10]). Last, the lower bound is shown using a family of WA $(\mathcal{R}_i)_{i\in\mathbb{N}}$ whose dimension is polynomial in i and strongest Z-linear invariant has a length that is exponential in i. It is defined, using permutation matrices of dimension p, for some prime number p, which generate cyclic groups. The family is obtained by using block matrices composed of such permutation matrices. All the details are given in Appendix B.4 of the full version of this paper [5]. ◀

Thanks to this theorem, using Algorithm 1 with a large enough c (at most doubly-exponential in the dimension of the given WA), and thanks to Theorem 36, we can prove the following result:

▶ **Theorem 38.** *The strongest Z-linear/affine invariant of a* WA *is computable in 2-*EXPTIME*.*

This allows us to prove Theorem 12. Indeed, given a WA $\mathcal{R}$, we first compute an equivalent minimal WA, which can be done in polynomial time (see e.g. [16, Corollary 4.17]). Then, using Algorithm 1, we compute the strongest Z-linear (resp. Z-affine) invariant of $\mathcal{R}$. Corollary 23 (resp. Corollary 30) ensures that its dimension is the register complexity of f w.r.t. the class of linear (resp. affine) CRA, and the effectiveness follows from Proposition 27 (resp. its affine version).

Moreover, thanks to Theorem 38 and the results of [3], we also have:

▶ **Theorem 39.** *The sequentiality and unambiguity of a rational series are in 2-*EXPTIME*.*

Note that the complexities of the last two theorems drop down to EXPTIME when we have a simply exponential bound on the length of the strongest invariant. This is the case when one considers unary alphabets or WA with commuting transition matrices in the linear setting, as stated in Theorem 37. In these cases, the bound is sharp. It is still not clear however whether it is possible to close the gap between the bounds in the general case.

▶ Remark 40. It is also worth noting that, while the characterizations that we obtained are valid for any field, the complexities of the algorithms are given in terms of number of elementary operations over the considered field. Which means that they hold for fields where we can perform basic operations in polynomial time (such as $\mathbb{Q}$ or its finite extensions). Moreover, the general upper bounds on the lengths given by Theorem 37 were proven only for fields verifying a specific property (which is verified by $\mathbb{Q}$). See the proof for more details.

5.4 State/register tradeoff

Reducing the number of registers may increase the number of states and vice-versa. The following theorem summarizes what we know on this tradeoff.

▶ **Theorem 41.** *Let f be a rational series realized by some d-dimensional* WA $\mathcal{R}$*. Consider some pair of integers (n, k) optimal for f w.r.t. the class of linear* CRA*. The inequalities* $1 \leq n \leq \mathrm{length}(\overline{\mathsf{LR}\,(\mathcal{R})}^{\ell}) = O(2^{2^d})$ *and* $\dim(\overline{\mathsf{LR}\,(\mathcal{R})}^{\ell}) \leq k \leq d$ *hold true.*

(They are valid in the affine setting as well)

▶ Remark 42. Building the CRA from the strongest invariant is not always optimal. There are some cases where it is possible to reduce the number of states of a CRA exponentially, while keeping the minimal number of registers, by choosing an invariant that is weaker than the strongest Z-linear/Z-affine invariant but shorter.

6 Conclusion

We have shown how to decide variants of CRA minimization problems, and have given complexity for the respective algorithms. There are several ways in which these algorithms could be improved. First, it would be worth reducing the gap between the lower and the upper bounds on the length of the strongest Z-linear invariant. Second, identifying a canonical invariant associated with the state-register minimization problem would allow to derive a deterministic algorithm for this problem. Third, one could hope for better complexity if one only considers the existence of equivalent CRA. For instance, in [13] the authors give a PSpace algorithm for the determinization problem (i.e. 1-register minimization problem) in the case of a polynomially ambiguous automaton, via a quite different approach.

Another line of research consists in trying to use the techniques we developed to solve the register minimization problem for other classes of CRA, for instance copyless CRA (which correspond to multi-sequential WA). Another ambitious goal is to consider register minimization in the context of different semirings, but there all the linear algebra tools which are crucial to solving these problems completely break down. Similarly, it seems that register minimization for polynomial automata would be very difficult: it was shown recently in [12] that the strongest algebraic invariant of a polynomial automaton is not computable. One possibility may be to bound the “degree” of the invariants, where Z-affine sets would correspond to algebraic sets of degree one.

References

1 Rajeev Alur, Loris D’Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 13–22. IEEE Computer Society, 2013.

2 Rajeev Alur and Mukund Raghothaman. Decision problems for additive regular functions. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2013.

3 Jason Bell and Daniel Smertnig. Noncommutative rational pólya series. *Selecta Mathematica*, 27, July 2021. doi:10.1007/s00029-021-00629-2.

4 Jason P. Bell and Daniel Smertnig. Computing the linear hull: Deciding deterministic? and unambiguous? for weighted automata over fields. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023, Boston, MA, USA, June 26-29, 2023*, pages 1–13. IEEE, 2023. doi:10.1109/LICS56636.2023.10175691.

5 Yahia Idriss Benalioua, Nathan Lhote, and Pierre-Alain Reynier. Minimizing cost register automata over a field, 2024. arXiv:2307.13505.

6 Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979. URL: https://www.worldcat.org/oclc/06364613.

7 Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977. doi:10.1016/0304-3975(77)90049-4.

8 Thomas Colcombet and Daniela Petrisan. Automata in the category of glued vector spaces. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, volume 83 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.52.

9 Laure Daviaud. Containment and equivalence of weighted automata: Probabilistic and max-plus cases. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications - 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings*, volume 12038 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2020. doi:10.1007/978-3-030-40608-0_2.

10 Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois. Degree of sequentiality of weighted automata. In *FOSSACS 2017*, volume 10203 of *Lecture Notes in Computer Science*, pages 215–230, 2017.

11 Laure Daviaud, Pierre-Alain Reynier, and Jean-Marc Talbot. A generalised twinning property for minimisation of cost register automata. In *LICS '16*, pages 857–866. ACM, 2016.

12 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. On strongest algebraic program invariants. *J. ACM*, 70(5):29:1–29:22, 2023. doi:10.1145/3614319.

13 Ismaël Jecker, Filip Mazowiecki, and David Purser. Determinisation and unambiguisation of polynomially-ambiguous rational weighted automata. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 46:1–46:13. ACM, 2024. doi:10.1145/3661814.3662073.

14 Heiko Vogler Manfred Droste, Werner Kuich. *Handbook of Weighted Automata*. Springer Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-01492-5.

15 Mehryar Mohri. Finite-state transducers in language and speech processing. *Comput. Linguistics*, 23(2):269–311, 1997.

16 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009. doi:10.1017/CBO9781139195218.

17 Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

Breaking a Graph into Connected Components with Small Dominating Sets

Matthias Bentert
University of Bergen, Norway

Michael R. Fellows
University of Bergen, Norway
Lebanese American University, Beirut, Lebanon

Petr A. Golovach
University of Bergen, Norway

Frances A. Rosamond
University of Bergen, Norway
Lebanese American University, Beirut, Lebanon

Saket Saurabh
The Institute of Mathematical Sciences, Chennai, India

Abstract

We study Dominated Cluster Deletion. Therein, we are given an undirected graph $G = (V, E)$ and integers k and d and the task is to find a set of at most k vertices such that removing these vertices results in a graph in which each connected component has a dominating set of size at most d. We also consider the special case where d is a constant. We show an almost complete tetrachotomy in terms of para-NP-hardness, containment in XP, containment in FPT, and admitting a polynomial kernel with respect to parameterizations that are a combination of k, d, c, and Δ, where c and Δ are the degeneracy and the maximum degree of the input graph, respectively. As a main contribution, we show that the problem can be solved in $f(k, d) \cdot n^{O(d)}$ time, that is, the problem is FPT when parameterized by k when d is a constant. This answers an open problem asked in a recent Dagstuhl seminar (23331). For the special case $d = 1$, we provide an algorithm with running time $2^{\mathcal{O}(k \log k)}nm$. Furthermore, we show that even for $d = 1$, the problem does not admit a polynomial kernel with respect to $k + c$.

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

Keywords and phrases Parameterized Algorithms, Recursive Understanding, Polynomial Kernels, Degeneracy

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.24

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
Michael R. Fellows: Supported by the Alexander von Humboldt Foundation under the Humboldt Renewed Research Stay in Germany and the Return to Germany Program for Humboldt Research Prizewinner Fellows.
Petr A. Golovach: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
Frances A. Rosamond: Supported by the Western Sydney University and the Alexander von Humboldt Foundation under the Humboldt Renewed Research Stay in Germany for Humboldt Research Prizewinner Fellows.
Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; 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

Cluster Vertex Deletion is an important problem in theoretical computer science [3, 22]. Therein, one is given a graph and the task is to delete as few vertices as possible such that the resulting graph is a *cluster graph*, that is, a graph in which each connected component is a clique. However in many real-world applications, it is often too restrictive to require that each connected component is a clique. For this reason, many generalizations (but also special cases) of Cluster Vertex Deletion have been studied. Examples include variants where each connected component in the resulting graph is an s-club (a graph of diameter at most s) [12, 19] or an s-plex (a graph in which each vertex has degree at least $n - s$) [15, 23]. The special case, where each clique in the solution graph has to be of size one is Vertex Cover, one of the most studied problems in the entire field of theoretical computer science [1, 16].

Different problems of this clustering type were explored in the recent Dagstuhl seminar on "Recent Trends in Graph Decomposition" [2]. Starting from the observation that each connected component of the resulting graph in Cluster Vertex Deletion contains a dominating set of size 1, the participants introduced a problem that explores a generalization in which the resulting connected components (clusters) satisfy other "simple integrity requirements" (see Ajwani et al. [2]). The notion they came up with was the following.

Dominated Cluster Deletion

Input: A graph G and positive integers k and d.

Question: Is there a set S of at most k vertices such that the domination number of each connected component of $G - S$ is at most d?

In this work, we consider both Dominated Cluster Deletion as well as the special case where d is a constant. We refer to the latter problem as d-Dominated Cluster Deletion. A natural question to ask is whether Dominated Cluster Deletion is fixed parameter tracatble (FPT) when parameterized by $k+d$? That is, can the problem be solved in $f(k,d) \cdot n^{\mathcal{O}(1)}$, where f is a function that depends only on k and d. Unfortunately, even for $k = 0$, the problem corresponds to the Dominating Set, and hence it is unlikely to be FPT, as Dominating Set is W[2]-complete parameterized by solution size [10]. This led the participants of the Dagstuhl seminar to ask for a parameterized tractability result, where d is allowed to play an XP-role in the exponent of the polynomial. In particular, they ask the following question, which we answer in the affirmative.

Can Dominated Cluster Deletion be solved in $f(k,d) \cdot n^{\mathcal{O}(d)}$ time?

Our Results. We study Dominated Cluster Deletion and d-Dominated Cluster Deletion and show an almost complete tetrachotomy in terms of para-NP-hardness, containment in XP, containment in FPT, and admitting a polynomial kernel with respect to parameterizations that are a combination of k, d, c, and Δ, where c and Δ are the degeneracy and the maximum degree of the input graph, respectively. As a main contribution and using the framework of *recursive understanding*, we show that the problem can be solved in $f(k,d) \cdot n^{O(d)}$ time, that is, the problem is FPT when parameterized by k when d is a constant. For the special case $d = 1$, we provide an algorithm with running time $2^{\mathcal{O}(k \log k)}nm$. Furthermore, we show that even for $d = 1$, the problem does not admit a polynomial kernel with respect to $k + c$ using OR-cross-compositions. Our results are summarized in Figure 1.

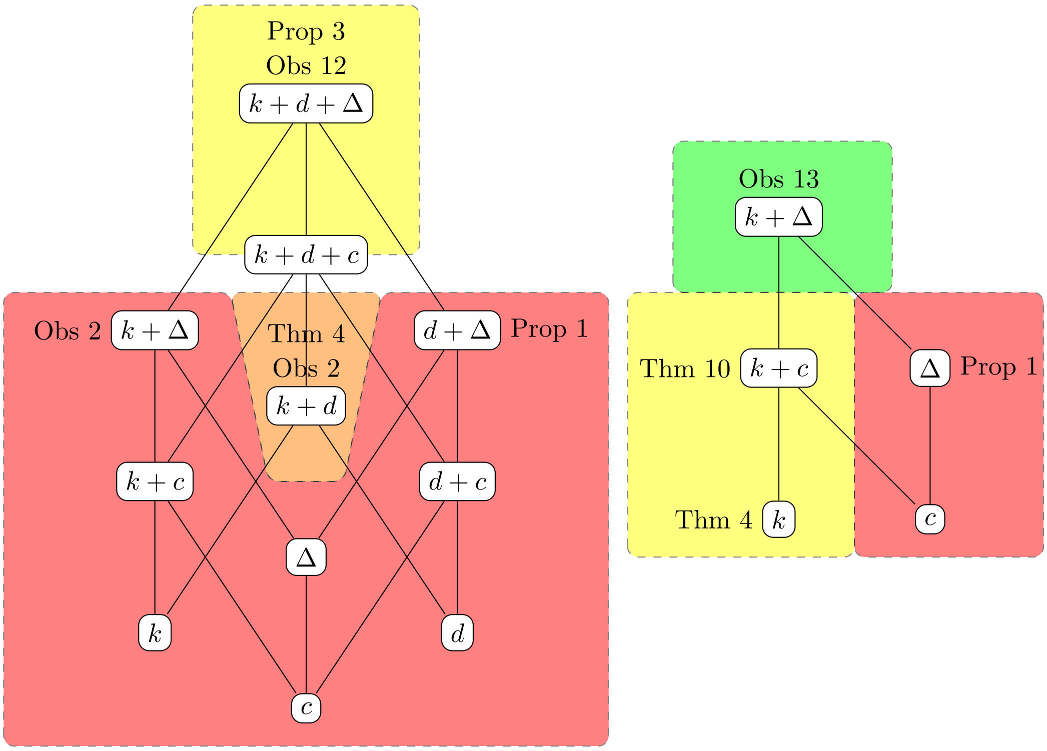


Figure 1 Overview over our results. Results for Dominated Cluster Deletion are shown on the left and results for d-Dominated Cluster Deletion are shown on the right. Therein, k and d are the input values and c and Δ are the degeneracy and the maximum degree of the input graph, respectively. An edge between two parameters α and β where β is above α indicates that the value of β in any instance will always be larger than the value of α in that instance. Any hardness result for β immediately implies the same hardness result for α and any positive result for α immediately implies the same positive result for β. The red boxes indicate parameters that lead to para-NP-hardness (NP-hardness for constant parameter values), an orange box indicates that the problem is in XP and W[1]-hard, a yellow box indicates that the parameterized problem is fixed-parameter tractable, but does not admit a polynomial kernel, and a green box indicated that the problem admits a polynomial kernel. We remark that all negative results apply even if the input graph is connected with the exception of $k+d+\Delta$, which does not admit a polynomial kernel in general but does so when the input graph is connected. We do not know whether Dominated Cluster Deletion parameterized by $k+d+c$ is fixed-parameter tractable or W[1]-hard.

2 Preliminaries and Basic Observations

We consider only undirected simple graphs and use the standard graph-theoretic terminology (see, e.g., [9]). Throughout the paper, we use n and m to denote the number of vertices and edges of G, respectively. For a set of vertices $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G induced by the vertices in X. We define $G - X = G[V(G) \setminus X]$ and we write $G - v$ instead of $G - \{v\}$ for a single vertex. For a vertex v, $N_G(v)$ denotes the *open neighborhood* of v, that is, the set of vertices adjacent to v, and $N_G[v] = \{v\} \cup N_G(v)$ is the *closed neighborhood*. For $X \subseteq V(G)$, we define $N_G(X) = \left(\bigcup_{v\in X} N_G(v)\right) \setminus X$ and $N_G[X] = \bigcup_{v\in X} N_G[v]$. For a vertex v, $d_G(v) = |N_G(v)|$ denotes the *degree* of v. We write $P = v_0v_1 \ldots v_\ell$ to denote the path P with vertices $v_0, v_1, \ldots, v_\ell$ and edges $\{v_{i-1}, v_i\}$ for $i \in \{1, \ldots, \ell\}$. The vertices v_0 and v_ℓ are the *end-vertices* of P. The *length* of P is the number of edges of P. For a path P

with end-vertices u and v, we say that P is a (u, v)*-path.* A (u, v)-path of minimum length is a *shortest* (u, v)-path. We also say that P is a shortest path if P is a shortest (u, v)-path for the end-vertices u and v of P. A graph G is *connected* if for every two vertices u and v, G contains a path whose end-vertices are u and v. A *connected component* (or simply a *component*) is an inclusion maximal induced connected subgraph of G. The *diameter* of a connected graph G is the maximum length of a shortest (u, v)-path where the maximum is taken over all pairs of vertices u and v. A vertex u *dominates* $v \in V(G)$ if $v \in N_G[u]$. A set of vertices $D \subseteq V(G)$ is a *dominating set* if every vertex $v \in V(G)$ is dominated by some vertex of D, that is, $N_G[D] = V(G)$. The minimum size of a dominating set is called the *domination number* of G and is denoted by $\gamma(G)$. For two distinct vertices u and v of a graph G, a set $S \subseteq V(G)$ is a (u, v)*-separator* if $G - S$ has no (u, v)-path. A pair (A, B), where $A, B \subseteq V(G)$ and $A \cup B = V(G)$, is called a *separation* of G if there is no edge $\{u, v\}$ with $u \in A \setminus B$ and $v \in B \setminus A$. In other words, $A \cap B$ is a (u, v)-separator for every $u \in A \setminus B$ and $v \in B \setminus A$. We consider only separations with $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. The *order* of (A, B) is $|A \cap B|$.

We refer to the books of Downey and Fellows [11] and Cygan et al. [7] for a detailed introduction to parameterized complexity theory. Formally, a *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is a set of strings over a finite alphabet Σ. This means that an input of a parameterized problem is a pair (x, k), where x is a string over Σ and $k \in \mathbb{N}$ is a *parameter.* A parameterized problem is *fixed-parameter tractable* (or FPT) if it can be solved in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time for some computable function f. We also say that a parameterized problem belongs to the class XP if it can be solved in $|x|^{f(k)}$ time for some computable function f. The complexity class FPT contains all fixed-parameter tractable parameterized problems.

A *kernelization algorithm* (or *kernel*) for a parameterized problem is a polynomial-time algorithm that maps each instance (x, k) of a parameterized problem L to an instance (x', k') of the same problem such that (i) $(x, k) \in L$ if and only if $(x', k') \in L$, and (ii) $|x'| + k' \leq f(k)$ for some computable function f. A kernel is *polynomial* if f is polynomial. While it can be shown that every decidable parameterized problem is FPT if and only if it admits a kernel, it is unlikely that every problem in FPT has a polynomial kernel. In particular, the standard *cross-composition* technique [4] can be used to show that a parameterized problem has no polynomial kernel unless NP $\subseteq$ coNP /poly.

In the conclusion of the section, we show a couple of simple observations. We start by showing that Dominated Cluster Deletion is NP-complete. Notice that the property that each connected component of a graph has a dominating set of size at most d is not hereditary and we cannot use the general result of Lewis and Yannakakis [18] about deletion problems.

▶ **Proposition 1.** *For any constant $d \geq 1$, d-Dominated Cluster Deletion is* NP*-complete in graphs of maximum degree* 4.

Proof. We reduce from Vertex Cover in graphs of maximum degree 3, which is known to be NP-hard [14]. We remind that a set S of vertices in a graph G is a *vertex cover* if for each edge $\{u, v\}$ in G, $u \in S$ or $v \in S$. Then, the task of Vertex Cover is the following. Given a graph G and a positive integer k, decide whether G has a vertex cover of size at most k.

Let (G, k) be an instance of Vertex Cover where the maximum degree in G is 3. We construct the graph G' as follows:

- construct a copy of G,
- for each vertex $v \in V(G)$, construct a path P_v on $3d$ vertices and identify one end-vertex of P_v with v.

Note that the maximum degree in G' is four. We claim that G has a vertex cover of size at most k if and only if (G', k) is a yes-instance of d-Dominated Cluster Deletion.

If G has a vertex cover S of size at most k, then each connected component of $G - S$ is either P_v or $P_v - v$ for $v \in V(G)$. Observe that $\gamma(P_v) = \gamma(P_v - v) = d$. Thus, S is a solution to the instance (G', k) of d-Dominated Cluster Deletion, that is, (G', k) is a yes-instance.

For the opposite direction, assume that (G', k) is a yes-instance of Dominated Cluster Deletion, that is, there is a set $S' \subseteq V(G')$ of at most k vertices such that for each connected component C of $G' - S'$, $\gamma(C) \leq d$. We construct the set $S \subseteq V(G)$ from S' by replacing each vertex $u \in S' \setminus V(G)$ by the nearest vertex in G. This means that if u is a vertex of $P_v - v$ for some $v \in V(G)$ then u is replaced by v. Clearly, $|S| \leq |S'| \leq k$. We claim that S is a vertex cover of G. The proof is by contradiction. Assume that there is an edge $\{u, v\}$ of G such that $u \notin S$ and $v \notin S$. By the construction of S, we have that $V(P_u) \cap S' = \emptyset$ and $V(P_v) \cap S = \emptyset$. This means that the vertices of P_u and P_v are in the same component C of $G' - S'$. However, any set D_u dominating the vertices of $P_u - u$ has size at least d. Symmetrically, we need at least d vertices to dominate the vertices of $P_v - v$. This implies that $\gamma(C) \geq 2d > d$; a contradiction. We obtain that S is a vertex cover of G of size at most k. This concludes the proof. ◀

We next observe that Dominated Cluster Deletion with $k = 0$ is equivalent to Dominating Set on connected input graphs. In Dominating Set, we ask whether a graph G has a dominating set of size at most d. It is well-known that Dominating Set is W[2]-complete when parameterized by the solution size and hence, an algorithm with running time $f(d) \cdot n^{\mathcal{O}(1)}$ would contradict the basic assumptions of parameterized complexity that FPT $\neq$ W[2] [11]. Furthermore, the existence of an algorithm running in $f(k) \cdot n^{o(d)}$ time would contradict the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [17] (we refer to the textbook by Cygan et al. [7] for an introduction to computational lower bounds based on the ETH). Finally, Dominating Set is NP-hard on graphs of maximum degree 3 [5].

▶ **Observation 2.** *Dominated Cluster Deletion with $k = 0$ is NP-hard on subcubic graphs,* W[2]*-hard on general graphs, and any $f(k) \cdot n^{o(d)}$-time algorithm for it would contradict the ETH.*

Finally, we show that Dominated Cluster Deletion is fixed-parameter tractable when parameterized by $k + d + \Delta$.

▶ **Proposition 3.** *Dominated Cluster Deletion can be solved in $\mathcal{O}((d(\Delta + 1) + 1)^k \cdot 3^{d(\Delta+1)} \cdot (n + m))$ time.*

Proof. We construct a simple branching algorithm as follows. First, we find any connected subgraph with $d(\Delta + 1) + 1$ vertices. Note that any connected component in the graph obtained after removing a solution S can contain at most $d(\Delta + 1)$ vertices as it has a dominating set of size d and the maximum degree is Δ. Hence, each of the d vertices in the dominating set can dominate at most $(\Delta + 1)$ vertices each. Thus, at least one vertex of the chosen connected subgraph is contained in the solution S and we branch on that vertex and reduce k by one. After reaching depth k in the search tree, we discard the current branch if we can still find a connected subgraph with $d(\Delta + 1) + 1$ vertices. Each remaining branch represents a graph in which each connected component contains at most $d(\Delta + 1)$ vertices. For each such component, we can compute an optimal solution in $\mathcal{O}(3^{d(\Delta+1)}(n + m))$ time by trying all possible choices of assigning vertices in the subgraph to the solution S, the dominating set, or the rest of the graph. We can then check in linear time whether the current assignment is a valid solution and remember the solution that assigns a minimum number of vertices to S. Finally, we sum up all the optimal values for k in each of the connected components to check whether the current branch in the search tree leads to a solution. If any branch leads to a solution, then we return yes and otherwise we return no. The correctness follows from the fact that we exhaustively search for all possible solutions and the running time is in $\mathcal{O}((d(\Delta+1)+1)^k \cdot 3^{d(\Delta+1)} \cdot (n+m))$. This concludes the proof. ◀

3 An FPT algorithm for d-Dominated Cluster Deletion

In this section, we show d-Dominated Cluster Deletion parameterized by k is fixed-parameter tractable. This also shows that Dominated Cluster Deletion is contained in XP.

▶ **Theorem 4.** *For every positive integer d, d-Dominated Cluster Deletion can be solved in $f(k,d) \cdot n^{\mathcal{O}(d)}$ time.*

We remark that the exponential dependency on d of the polynomial part of the running time seems unavoidable by Observation 2. We prove Theorem 4 by applying the *recursive understanding* technique introduced by Chitnis et al. [6] and using the meta theorem of Lokshtanov et al. [20]. The meta theorem considerably simplifies the arguments. However, the proof becomes nonconstructive and we can only claim a nonuniform FPT algorithm. Still, a uniform FPT algorithm may be given using the techniques of Chitnis et al. [6] or the dynamic programming approach of Cygan et al. [8].

The meta theorem of Lokshtanov et al. [20] allows to reduce solving a graph problem on general graphs to highly connected graphs. Let p, q be positive integers. A graph G is said to be (p,q)-*unbreakable* if for every separation (A,B) of G of order at most q, either $|A \setminus B| \leq p$ or $|B \setminus A| \leq p$, that is, G has no separator of size at most q that partitions the graph into two parts of size at least $p+1$ each.

We remind that in *monadic second-order logic* (MSO) on graphs, we have logical connectives $\vee$, $\wedge$, $\neg$, and variables for vertices, edges, and sets of vertices and edges. We can apply the quantifiers $\forall$ and $\exists$ to these variables and use the predicates (i) $x \in X$ where x is a vertex (respectively, edge) variable and X is a vertex (respectively, edge) set variable, (ii) $inc(u,e)$ where u is a vertex variable and e is an edge variable that denotes that u is incident to e, (iii) $adj(u,v)$ where u and v are vertex variables denoting the adjacency of u and v, and (iv) the equality predicate for variables of the same type. The *counting monadic second-order logic* (CMSO) extends MSO by allowing testing whether the cardinality of a set is equal to p modulo q for integers $0 \leq p < q$ with $q \geq 2$. Given a CMSO sentence φ, the task of the model-checking problem for φ (denoted by CMSO[φ]) is to decide whether $G \models \varphi$ for a graph G. Lokshtanov et al. [20] proved the following result.

▶ **Proposition 5** ([20]). *Let φ be a CMSO sentence. For all positive integers q, there exists a positive integer p such that if there exists an algorithm that solves CMSO[φ] on (p,q)-unbreakable graphs in $\mathcal{O}(n^c)$ time for some $c > 4$ then there exists an algorithm that solves CMSO[φ] on general graphs in $\mathcal{O}(n^c)$ time.*

To apply Proposition 5, we observe that Dominated Cluster Deletion can be expressed in MSO.

▶ **Observation 6.** *For given $d \geq 1$ and $k \geq 0$, there is a MSO sentence φ (depending on d and k) such that $G \models \varphi$ if and only if (G,k) is a yes-instance of Dominated Cluster Deletion.*

Proof. For $k = 0$, (G, k) is a yes-instance of Dominated Cluster Deletion if and only if every connected component C of G has a dominating set of size at most d, that is, it holds that $\exists v_1 \in V(C)\colon \dots \exists v_d \in V(C)\colon \forall w \in V(C)\colon (w = v_1) \vee \dots \vee (w = v_d) \vee adj(w, v_1) \vee \dots \vee adj(w, v_d)$. Further, we have that $C = G[X]$ for $X \subseteq V(G)$ is a connected component of G if and only if $G[X]$ is connected and for any proper superset Y of X, $G[Y]$ is not connected. It is standard to express the connectivity of $G[X]$ as follows: $\forall Z \subset X\colon\ \exists u \in Z\colon\ \exists v \in X \setminus Z\colon adj(u, v)$. This proves that the property that $(G, 0)$ is a yes-instance of Dominated Cluster Deletion can be expressed in MSO.

For $k \geq 1$, (G, k) is a yes-instance of Dominated Cluster Deletion if and only if $(G, 0)$ is a yes-instance or $\exists v_1 \in V(G) \dots \exists v_k \in V(G)$ such that every connected component C of $G - \{v_1, \dots, v_k\}$ has a dominating set of size at most d. It is straightforward to modify the formula for $k = 0$ to get an MSO sentence for the case $k \geq 1$. ◀

We next show that d-Dominated Cluster Deletion is fixed-parameter tractable for unbreakable graphs. More precisely, we prove the following lemma.

▶ **Lemma 7.** *Let p and k be positive integers. Then, d-Dominated Cluster Deletion can be solved in $(p + k)^{\mathcal{O}(k+d)} \cdot n^{\mathcal{O}(d)}$ time on (p, k)-unbreakable graphs.*

Proof. Let (G, k) be an instance of d-Dominated Cluster Deletion where G is (p, k)-unbreakable.

Suppose that (G, k) is a yes-instance, that is, there is a set $S \subseteq V(G)$ of at most k vertices such that for each connected component C of $G - S$, $\gamma(C) \leq d$. Because G is a (p, k)-unbreakable graph, $G - S$ has a connected component C with at least $n - |S| - p \geq n - k - p$ vertices and the total number of vertices in the other connected components of $G - S$ is at most p. We call the component C *big*. Denote by $C_1, \dots, C_\ell$ the connected components of $G - S$ that are distinct from C and let D be a dominating set of C of size at most d. Observe that $V(C) \subseteq N_G[D] \subseteq V(C) \cup S$ and $V(C_i) \subseteq V(G) \setminus N_G[D]$ for all $i \in \{1, \dots, \ell\}$. Furthermore, if $S' = S \setminus N_G[D]$ then $S = S' \cup N_G(\bigcup_{i=1}^{\ell} V(G_i))$. These observations lead to the following algorithm.

We guess a dominating set D of the (potential) big connected component by trying all possible choices of at most d vertices of G. Let $R = V(G) \setminus N_G[D]$. We immediately discard the choice of D if $|R| > p + k$ because such sets D cannot be dominating sets of the big component. From now on, we assume that D is fixed and $|R| \leq p + k$.

Next, we guess the set of at most k vertices $S' = R \cap S$ for a potential solution S to (G, k) by trying all possible choices of such a set. For each choice of S', we consider $H = G[R \setminus S']$ and find the connected components $C_1, \dots, C_\ell$ of H. For each $i \in \{1, \dots, \ell\}$, we verify whether C_i has a dominating set of size at most d and immediately discard the current choice of S' if this does not hold. Further, we construct $S = S' \cup N_G(\bigcup_{i=1}^{\ell} V(C_i))$ and check whether $|S| \leq k$. If this holds then we conclude that S is a solution, return it, and stop. Otherwise, we discard the current choice of S'.

If we do not find a solution for all choices of S' for the given D, we discard the choice of D. If all sets D are discarded, then we conclude that (G, k) is a no-instance.

Our structural observations about yes-instances immediately imply the correctness of the algorithm. To evaluate the running time, observe that we have at most n^d choices of D that can be generated in $n^{\mathcal{O}(d)}$ time. For each choice of D with $|R| \leq p + k$, we have at most $(p + k)^k$ possibilities to choose S'. We can find connected components of H in $\mathcal{O}(p + k)$ time, and we can check in time $(p + k)^{\mathcal{O}(d)}$ whether each connected component can be dominated by at most d vertices. Thus, the total running time is $(p + k)^{\mathcal{O}(k+d)} \cdot n^{\mathcal{O}(d)}$. This concludes the proof. ◀

Combining Observation 6 and Lemma 7, we obtain Theorem 4 as a direct corollary of Proposition 5.

4 An FPT algorithm for 1-Dominated Cluster Deletion

In this section, we demonstrate a uniform algorithm for Dominated Cluster Deletion for the special case $d = 1$, that is, for 1-Dominated Cluster Deletion. The crucial property that makes the problem significantly easier for $d = 1$ is that each cluster is of diameter at most two. We use a simple branching algorithm to reduce the input graph to a connected graph of diameter at most two. Then we can exploit the following structural property of a solution.

▶ **Lemma 8.** *Let G be a graph of diameter at most two and let (A, B) be a separation of G with $X = A \cap B$. Suppose that for $S \subseteq V(G)$, each connected component of $G - S$ is dominated by a single vertex and it holds that (i) $(A \setminus B) \setminus S \neq \emptyset$, (ii) $(B \setminus A) \setminus S \neq \emptyset$, and (iii) $X \cap S = \emptyset$. Then for every connected component C of $G - S$, any vertex $v \in V(C)$ dominating C is in X. Furthermore, the vertices of each connected component of $G[X]$ are in the same connected component of $G - S$ and the vertices of distinct connected component of $G[X]$ are in separate components of $G - S$.*

Proof. Let C be a connected component of $G - S$. Assume for the sake of a contradiction that there is a vertex $v \in V(C)$ that dominates C such that $v \in A \setminus B$. By (ii), there is a vertex $u \in (B \setminus A) \setminus S$. Since the diameter of G is at most two, there is a (u, v)-path P of length at most two. Since u and v are separated by X, $P = uxv$ for some $x \in X$. Note that (iii) implies that $x \notin S$. Then, $u \in V(C)$. However, u is not dominated by v; a contradiction. This proves the first claim.

For the second claim, it is straightforward to see that the vertices of each connected component of $G[X]$ are in the same connected component of $G - S$ because of (iii). Let C_1 and C_2 be distinct connected components of $G[X]$. Notice that any vertex of G adjacent to some vertices in both connected components is either in $A \setminus B$ or $B \setminus A$. According to the first claim of the lemma, such vertices cannot be dominating vertices for the cluster. This proves that the vertices of distinct connected components of $G[X]$ are in separate components of $G - S$ and this concludes the proof. ◀

We are now in a position to prove the main theorem of this section.

▶ **Theorem 9.** *1-Dominated Cluster Deletion can be solved in $\mathcal{O}((k + 3)^{k+5/2} \cdot nm)$ time.*

Proof. We use the method of bounded search trees to solve the problem. Our recursive branching algorithm, called Cluster (Algorithm 1), takes as input a graph G and an integer k, and returns either Yes or No depending on whether (G, k) is a yes- or a no-instance. For simplicity, we solve the decision problem, but the algorithm can easily be modified to output a solution if it exists.

We analyze the algorithm to show its correctness. Trivially, if $k < 0$ then (G, k) is a no-instance. Hence, we correctly return No and quit in Line 2. Also, if the set of vertices becomes empty then we should return Yes as is done in Line 3.

If G has a connected component C that can be dominated by one vertex, then this component is already a required cluster and we can exclude it from the consideration. This is done in Lines 4–5.

In Lines 7–14, we deal with disconnected graphs. In this case, we solve the problem for each connected component separately, and for each component C_i, we find the minimum k_i such that (C_i, k_i) is a yes-instance. Notice that because of the previous step, each connected component is not dominated by a single vertex. Thus, we consider only k_i satisfying $1 \leq k_i \leq k - 1$. Then, we verify whether the solutions for components can be combined. From now on, we assume that G is connected.

Algorithm 1 Cluster(G, k).

```
Input: A graph $G$ and an integer $k$.
Result: Yes or No.
begin
    if $k < 0$ then return No and quit;
    if $V(G) = \emptyset$ then return No and quit;
    if $G$ has a connected component $C$ with $\gamma(G) = 1$ then
        call Cluster$(G - V(C), k)$, return the result, and quit
    end
    if $G$ is disconnected then
        find connected components $C_1, \dots, C_\ell$ of $G$;
        foreach $i \in \{1, \dots, \ell\}$ do
            find minimum $1 \leq k_i \leq k - 1$ s.t. Cluster$(C_i, k_i)$ returns Yes and
            set $k_i := k$ if Cluster$(C_i, k - 1)$ returns No
        end
        return Yes if $k_1 + \dots + k_\ell \leq k$ and return No, otherwise, and quit
    end
    if there is a shortest path $v_1v_2v_3v_4$ then
        foreach $i \in \{1, 2, 3, 4\}$ do
            call Cluster$(G - v_i, k - 1)$;
            return Yes and quit if the algorithm returns Yes
        end
        return No and quit
    end
    if there is $v \in V(G)$ s.t. $|V(G) \setminus N_G[v]| \leq k$ then
        return Yes and quit
    end
    find a separation $(A, B)$ of $G$ with the separator $X = A \cap B$ of size at most $k$
    and return No and quit if such a separation does not exist;
    foreach $v \in X$ do
        call Cluster$(G - v, k - 1)$;
        return Yes and quit if the algorithm returns Yes
    end
    call Cluster$(G - (A \setminus B)|, k - |A \setminus B|)$ and
    return Yes and quit if the algorithm returns Yes;
    call Cluster$(G - (B \setminus A)|, k - |B \setminus A|)$ and
    return Yes and quit if the algorithm returns Yes;
    if there is $v \in V(G) \setminus X$ s.t. $v$ has neighbors in distinct components of $G[X]$ then
        call Cluster$(G - v, k - 1)$;
        return Yes and quit if the algorithm returns Yes
    end
    return No
end
```

In Lines 15–21, we analyze the case when G has a shortest path on four vertices. If $v_1v_2v_3v_4$ is a shortest (v_1, v_4)-path then the vertices v_1, v_2, v_3, v_4 cannot belong to the same cluster because they cannot be dominated by the same vertex. This implies that at least one vertex of such a path should be included in any solution. We branch on four possible options and return Yes if we have a yes-instance for one of the branches. Otherwise, we return No and stop. From this moment, we assume that G has no shortest paths of length three, that is, the diameter of G is at most two.

Further, in Lines 22–24, we check whether the instance admits a trivial solution with one cluster. This holds if there is $v \in V(G)$ such that $|V(G) \setminus N_G[v]| \leq k$ – we simply delete all the vertices that are not dominated by v. Now, we can assume that for each solution S, $G - S$ has at least two components.

In Lines 25–26, we find a separation (A, B) of G with the separator $X = A \cap B$ of size at most k (we remind that we only consider separations (A, B) where both $A \setminus B$ and $B \setminus A$ are nonempty). If such a separation does not exist, then for each $S \subseteq V(G)$ of size at most k, $G - X$ is connected. Hence, (G, k) may have only a solution with one cluster. However, this case was already excluded. Thus, we correctly return No and stop in Line 26. From now on, we assume that a separation (A, B) of order at most k exists.

In Lines 27–30, we branch on the possibility that X contains a vertex of a solution. We return Yes and stop if we find such a solution. Next, in Lines 31–34, we check whether there is a solution that includes either every vertex of $A \setminus B$ or every vertex of $B \setminus A$.

Finally, we conclude that any solution to (G, k) excludes the vertices of X and, furthermore, each solution contains at least one vertex of $A \setminus B$ and at least one vertex of $B \setminus A$. Then by Lemma 8, for any solution S to (G, S), it holds that for each connected component C of $G - S$, any vertex $v \in V(C)$ dominating C is in X and, furthermore, the vertices of each connected component of $G[X]$ are in the same connected component of $G - S$ and the vertices of distinct connected component of $G[X]$ are in separate components of $G - S$. Recall that we excluded the case when (G, k) has a solution with one cluster. Hence, if (G, k) is a yes-instance then $G[X]$ should have at least two connected components. Let C_1 and C_2 be two distinct connected components of $G[X]$. Because the diameter of G is at most two, we have that for every $u \in V(C_1)$ and every $v \in V(C_2)$, there is a vertex $x \in V(G) \setminus X$ such that uxv is a path in G. Because u and v are in distinct connected components of $G - S$, it holds that $x \in S$. We conclude that if (G, k) is a yes-instance then there is at least one vertex $v \in V(G) \setminus X$ which has neighbors in two distinct connected components of $G[X]$ and all such vertices are included in any solution S. We use this in Lines 35-39. We either find v and obtain that that $(G - v, k - 1)$ is a yes-instance or conclude that (G, k) is a no-instance. This concludes the correctness proof.

To evaluate the running time, notice that in Lines 15–21, we have four recursive calls. If we go to Lines 27-38, then we have at most $|X| + 3 \leq k + 3$ branches. For each branch, we reduce the parameter by at least one. Thus, the total number of leaves of the search tree is at most $(k+3)^k$. We can find connected components in $\mathcal{O}(n+m)$ time using standard techniques and we can check in $\mathcal{O}(n)$ time whether a connected graph has a dominating vertex. A shortest path on four vertices can be found in $\mathcal{O}(nm)$ time by performing a breadth-first search from each vertex. The existence of a vertex v such that $|V(G) \setminus N_G[v]| \leq k$ can be checked in $\mathcal{O}(m)$ time. To verify whether G has a separation of order k, we can use the algorithm of Gabow [13] with running time $\mathcal{O}((n + \min(k^{5/2}, kn^{3/4}))m)$. In fact, for undirected graphs, m can be replaced by kn, but we use the bound in the weaker form $\mathcal{O}(k^{5/2}nm)$. Finally, note that in Lines 9–11, for each $i \in \{1, \ldots, \ell\}$, we compute the minimum k_i such that (C_i, k_i) is a yes-instance. For this, we can use binary search, which adds an additional $\log k$-factor to

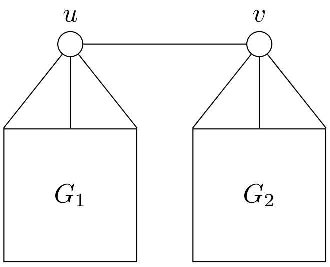


Figure 2 An illustration of the construction in the proof of Theorem 10.

the running time. However, $k_i \leq k-1$ and this means that the steps in Lines 9–11 can be executed in time $\mathcal{O}((k+3)^{k+5/2} \cdot nm)$. Summarizing, we obtain that the total running time is in $\mathcal{O}((k+3)^{k+5/2} \cdot nm)$. This concludes the proof. ◀

5 (No) Polynomial Kernels

In this section, we show some results related to the existence of polynomial kernels. Our main result here is that Dominated Cluster Deletion does not admit a polynomial kernel with respect to $k+c$ even if $d=1$. We mention that the construction is identical to a reduction by Figiel et al. [12] that shows that 2-Club Cluster Vertex Deletion does not have a polynomial kernel with respect to the solution size k. The two problems are loosely related as a graph with a universal vertex (a dominating set of size one) is a 2-Club. However, the other direction is not true, that is, not every 2-club has a dominating set of size one. The two results therefore do not directly follow from one another.

▶ **Theorem 10.** *1-Dominated Cluster Deletion does not admit a polynomial kernel with respect to $k+c$ unless $\mathrm{NP} \subseteq \mathrm{coNP}/\mathrm{poly}$ even if the input graph is connected.*

Proof. We show this result by providing a OR-cross-composition (see [4]) for 1-Dominated Cluster Deletion from itself. Recall that 1-Dominated Cluster Deletion is NP-complete by Proposition 1. Without loss of generality, we start with $T = 2^t$ instances of 1-Dominated Cluster Deletion, where each instance has the same number n of vertices, the same value of $k < n-1$, and where each of the input graphs is connected.

The main ingredient for our proof is a construction to merge two instances $I_1 = (G_1, k)$ and $I_2 = (G_2, k)$ into one instance $I^* = (G^*, k^*)$. We first prove that I^* is a yes-instance if and only if at least one of I_1 and I_2 is a yes-instance. Afterwards, we will show how to use this construction to get a cross-composition for all t instances.

The construction is depicted in Figure 2 and works as follows. We take the disjoint union of the two graphs G_1 and G_2 and add two vertices u and v. We make u adjacent to all vertices in G_1 and v adjacent to all vertices in G_2. Moreover, we add the edge $\{u, v\}$. Finally, we set the parameter $k^* = k+1$.

It remains to show that I^* is a yes-instance if and only if at least one of I_1 and I_2 is a yes-instance. First assume, that one of I_1 and I_2 is a yes-instance. Since both cases are completely analogous, we assume without loss of generality that I_1 is a yes instance. A solution for I^* consists of an optimal solution for I_1 and the vertex u. Note that adding u to the solution disconnects the graphs G_1 and G_2. Moreover, the connected component including G_2 also contains v and therefore has a domination number of one.

For the other direction, assume that the constructed instance contains a solution S of size at most $k+1$. Note that since we assume $k < n-1$, we need to include either u or v in S as otherwise we would need to include all $n > k^*$ vertices in either G_1 or G_2 into S as one vertex of each instance together with u and v induces a P_4 and no vertex is adjacent to both a vertex in G_1 and a vertex in G_2. We assume without loss of generality that u is included in S. Note that this leaves the graph G_1 as one connected component and since we already know that $u \in S$, it holds that S contains at most k vertices from G_1 which results in each connected component of G_1 being dominated by a single vertex. Thus I_1 is a yes-instance.

To complete the reduction, we iteratively half the number of instances by partitioning all instances into arbitrary pairs and merge the two instances in a pair into one instance. Note that in each iteration, all of our assumption still hold as all instances have $d = 1$, the number of vertices in I^* is $2n + 2 > k^* + 1$, and G^* is connected. After $\log(T)$ iterations, we are left with a single instance which is a yes-instance if and only if at least one of the T original instances is a yes-instance. The time required to compute the cross composition is in $O(\sum_{i\in[T]} |G_i| \cdot \log(T))$ which is polynomial in $\sum_{i\in[T]} |G_i|$ as each graph G_i contains at least one vertex. Moreover, the parameter k in the constructed instance is $k + \log(T)$ and the degeneracy of the instance is at most $n + \log(T)$ as we can put all the vertices in the T original instances first into the degeneracy ordering and afterwards add the newly added vertices in increasing order of their degree. Note that each vertex in an original instance has at most $n + \log(T)$ neighbors and each of the newly added vertices has at most $\log(T)$ neighbors "above themself". Thus, all requirements of a cross-composition are met and the proof is completed. ◀

We mention in passing that it is simple to generalize Theorem 10 to any constant $d > 0$. In each step, we simply add the following graph to each of the two newly added vertices u and v. We show the construction only for v. We start with a path $(w_1, w_2, \ldots, w_{3d-2})$ with $3d - 2$ vertices and make w_1 adjacent to v. Next, we iteratively replace each vertex w_i by a clique of size $k + \log(T) + 1$ vertices, where each vertex has the same neighborhood as w_i had. Note that we cannot fully remove any of the cliques and hence whenever we remove neither u nor v, then we are left with a connected graph which requires at least $2d > d$ vertices to dominate. Whenever we remove one of the two vertices, say u, then we can dominate the connected component of v by selecting v and one copy of each vertex in the clique corresponding to the vertices $w_3, w_6, \ldots, w_{3(d-1)}$. The rest of the proof is completely analogous to the proof of Theorem 10.

We next show that d-Dominated Cluster Deletion parameterized by $k + \Delta$ admits a polynomial kernel.

▶ **Proposition 11.** *d-Dominated Cluster Deletion parameterized by $k + \Delta$ admits a polynomial kernel of size $\mathcal{O}(k^2\Delta^3)$.*

Proof. First, we check for each connected component of the input graph whether it contains a dominating set of size d in $\mathcal{O}(n^{d+2})$ time. Note that since we assume d to be a constant, this is a polynomial running time. Moreover, each such connected component can be removed from the input graph to create a smaller equivalent instance. If afterwards, the number of connected components in the graph is larger than k, then we return a trivial no-instance as we need to include at least one vertex from each connected component into the solution S, meaning that any solution has size at least $k+1$. Finally, we bound the size of each connected component as follows. If the number of vertices in one connected component is larger than $k + (k(\Delta - 1) + 1)d(\Delta + 1) \in \mathcal{O}(k\Delta^2)$, then we again return a trivial no-instance as removing any solution S of size at most k leaves at most $k(\Delta - 1) + 1$ connected components and hence by the pigeonhole principle, at least one connected component has size larger than $d(\Delta + 1)$. Note that this connected component cannot contain a dominating set of size at most d and hence the entire instance is a no-instance.

The number remaining vertices is now at most $k\cdot(k+(k(\Delta-1)+1)d(\Delta+1)) \in \mathcal{O}(k^2\Delta^2)$ and since the maximum degree is Δ, the number of vertices and edges is in $\mathcal{O}(k^2\Delta^3)$. This concludes the proof. ◀

Next, we show that the previous result cannot be generalized to Dominated Cluster Deletion even if we include the parameter d to the kernel and restrict k to be 0.

▶ **Observation 12.** *Dominated Cluster Deletion does not admit a polynomial kernel with respect to $d+\Delta$ even if $k=0$ unless $\mathrm{NP} \subseteq \mathrm{coNP}/\mathrm{poly}$.*

Proof. We show a simple AND-composition from Dominating Set on connected input graphs. Given t instances of Dominating Set where all instances are connected and all instances ask for a dominating set of the same size d, we simply take the disjoint union of all of these graphs and set $k=0$. If all t instances are yes-instances, then each connected component of the constructed graph has a dominating set of size at most d, that is, the constructed instance is a yes-instance. If the constructed instance is a yes-instance, then also each connected component of the constructed graph has a dominating set of size at most d as $k=0$. Thus, all t original instances are yes-instances. Note that $k=0$, the value of d is the same as in each input instance, and the maximum degree is at most the number of vertices in one of the t input instances. Thus, we have provided an AND-cross-composition and this concludes the proof. ◀

Note that the previous reduction produces a disconnected graph. We show that this is necessary as Dominated Cluster Deletion admits a polynomial kernel when parameterized by $k+d+\Delta$ if the input graph is connected.

▶ **Observation 13.** *Dominated Cluster Deletion admits a kernel of size $\mathcal{O}(kd\Delta^3)$ when parameterized by $k+d+\Delta$ if the input graph is connected.*

Proof. If the number of vertices in the input graph is at most $kd\Delta^2+k+d(\Delta+1)-kd$, then we return the input graph as a trivial kernel of size $\mathcal{O}(kd\Delta^3)$. Otherwise we return a trivial no-instance as a kernel. We next show that this is justified. Assume towards a contradiction that the input instance has more vertices but is a yes-instance. Then, it contains a set S of at most k vertices such that removing S from the graph results in connected components which can each be dominated by at most d vertices. Note that removing a set of k vertices of degree at most Δ from a connected graph can result in a graph with at most $k(\Delta-1)+1$ connected components. Moreover, the dominating set of size d in each of these can dominate at most $d(\Delta)+1$ vertices. Thus, the entire graph contains at most $k+(k(\Delta-1)+1)d(\Delta+1) = kd\Delta^2+k+d(\Delta+1)-kd$ vertices, a contradiction. ◀

6 Conclusion

In this work, we initiated the study of Dominated Cluster Deletion and showed an almost complete tetrachotomy regarding its parameterized complexity when parameterized by combinations of k, d, c, and Δ. We remark that we leave two questions unanswered. First, is Dominated Cluster Deletion fixed-parameter tractable or W[1]-hard when parameterized by $k+d+c$? We conjecture that it is fixed-parameter tractable and the algorithm may be obtained by combining the recursive understanding technique [6, 20] and the results of Telle and Villanger [21]. Second, does d-Dominated Cluster Deletion parameterized by k admit a polynomial kernel when d and c are constants?

References

1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 62–69. SIAM, 2004.

2 Deepak Ajwani, Rob H. Bisseling, Katrin Casel, Ümit V. Çatalyürek, Cédric Chevalier, Florian Chudigiewitsch, Marcelo Fonseca Faraj, Michael R. Fellows, Lars Gottesbüren, Tobias Heuer, George Karypis, Kamer Kaya, Jakub Lacki, Johannes Langguth, Xiaoye Sherry Li, Ruben Mayer, Johannes Meintrup, Yosuke Mizutani, François Pellegrini, Fabrizio Petrini, Frances A. Rosamond, Ilya Safro, Sebastian Schlag, Christian Schulz, Roohani Sharma, Darren Strash, Blair D. Sullivan, Bora Uçar, and Albert-Jan Yzelman. Open problems in (hyper)graph decomposition. *CoRR*, abs/2310.11812, 2023. `arXiv:2310.11812`.

3 Stéphane Bessy, Marin Bougeret, Dimitrios M. Thilikos, and Sebastian Wiederrecht. Kernelization for graph packing problems via rainbow matching. In *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3654–3663. SIAM, 2023.

4 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

5 Valentin Bouquet, François Delbot, Christophe Picouleau, and Stéphane Rovedakis. On minimum dominating sets in cubic and (claw, H)-free graphs. *CoRR*, abs/2002.12232, 2020. `arXiv:2002.12232`.

6 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016.

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

8 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019.

9 Reinhard Diestel. *Graph Theory*. Springer, 2012.

10 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

12 Aleksander Figiel, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. On 2-clubs in graph-based data clustering: Theory and algorithm engineering. *Journal of Graph Algorithms and Applications*, 25(1):521–547, 2021.

13 Harold N. Gabow. Using expander graphs to find vertex connectivity. *Journal of the ACM*, 53(5):800–844, 2006.

14 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

15 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.

16 David G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In *Proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

18 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

19 Hong Liu, Peng Zhang, and Daming Zhu. On editing graphs into 2-club clusters. In *Proceedings of the 2nd Joint International Conference on Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM)*, pages 235–246. Springer, 2012.

20 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

21 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theoretical Computer Science*, 770:62–68, 2019.

22 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory of Computing Systems*, 65(2):323–343, 2021.

23 René van Bevern, Hannes Moser, and Rolf Niedermeier. Approximation and tidying – A problem kernel for s-plex cluster vertex deletion. *Algorithmica*, 62(3-4):930–950, 2012.

Multiway Cuts with a Choice of Representatives

Kristóf Bérczi
MTA-ELTE Matroid Optimization Research Group, Budapest, Hungary
HUN-REN–ELTE Egerváry Research Group on Combinatorial Optimization,
Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Tamás Király
HUN-REN–ELTE Egerváry Research Group on Combinatorial Optimization,
Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Daniel P. Szabo
Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Abstract

In the Multiway Cut problem, we are given an undirected graph with nonnegative edge weights and a subset of k terminals, and the goal is to determine a set of edges of minimum total weight whose removal disconnects each terminal from the rest. The problem is APX-hard for $k \geq 3$, and an extensive line of research has concentrated on closing the gap between the best upper and lower bounds for approximability and inapproximability, respectively.

In this paper, we study several generalizations of Multiway Cut where the terminals can be chosen as *representatives* from sets of *candidates* $T_1, \ldots, T_q$. In this setting, one is allowed to choose these representatives so that the minimum-weight cut separating these sets *via their representatives* is as small as possible. We distinguish different cases depending on (A) whether the representative of a candidate set has to be separated from the other candidate sets completely or only from the representatives, and (B) whether there is a single representative for each candidate set or the choice of representative is independent for each pair of candidate sets.

For fixed q, we give approximation algorithms for each of these problems that match the best known approximation guarantee for Multiway Cut. Our technical contribution is a new extension of the CKR relaxation that preserves approximation guarantees. For general q, we show $o(\log q)$-inapproximability for all cases where the choice of representatives may depend on the pair of candidate sets, as well as for the case where the goal is to separate a fixed node from a single representative from each candidate set. As a positive result, we give a 2-approximation algorithm for the case where we need to choose a single representative from each candidate set. This is a generalization of the $(2 - 2/k)$-approximation for k-Cut, and we can solve it by relating the tree case to optimization over a gammoid.

2012 ACM Subject Classification Theory of computation → Rounding techniques; Theory of computation → Facility location and clustering; Theory of computation → Network optimization; Theory of computation → Linear programming; Theory of computation → Graph algorithms analysis

Keywords and phrases Approximation algorithms, Multiway cut, CKR relaxation, Steiner multicut

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.25

Funding The research was supported by the Lendület Programme of the Hungarian Academy of Sciences – grant number LP2021-1/2021, by the Ministry of Innovation and Technology of Hungary – grant number ELTE TKP 2021-NKTA-62, and by Dynasnet European Research Council Synergy project – grant number ERC-2018-SYG 810115.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 25; pp. 25:1–25:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

For an undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_+$, the Multiway Cut problem asks for a minimum-weight cut $C \subseteq E$ separating any pair of terminals in a given terminal set $S = \{s_1, \ldots s_k\}$. As cuts can be identified with partitions of the nodes, this is equivalent to finding a node coloring of G with k colors such that terminal s_i is colored with color i for $i \in [k]$, and we seek to minimize the total weight of dichromatic edges.

1.1 Previous work

Dahlhaus et al. [5] showed that Multiway Cut is NP-hard even for $k = 3$, and provided a very simple combinatorial $(2 - 2/k)$-approximation that works as follows. For each s_i, determine a minimum-weight cut $C_i \subseteq E$ that separates s_i from all other s_j for $j \neq i$ – such a cut is called an *isolating* cut of s_i – and then take the union of the $k - 1$ smallest ones among the k cuts thus obtained. In an optimal multiway cut, the boundary of the component containing s_i is a cut isolating s_i, hence its weight is at least as large as that of C_i. Summing up these inequalities for all but the largest isolating cuts, since this counts each edge at most twice except for the boundary of the largest one, leads to a $(2 - 2/k)$-approximation.

Since the pioneering work of Dahlhaus et al., Multiway Cut has been a central problem in combinatorial optimization. The best known approximability as well as inapproximability bounds are based on a geometric relaxation called the *CKR relaxation*, introduced by Călinescu, Karloff and Rabani [4]. The current best approximation algorithm is due to Sharma and Vondrák [18] with an approximation factor of 1.2965, while the best known lower bound (assuming the Unique Games Conjecture) is slightly above 1.2 [2].

Various generalizations of Multiway Cut have been introduced. In the Multicut problem, we are given an undirected graph with non-negative edge weights, together with a demand graph consisting of edges $(s_1, t_1), \ldots, (s_k, t_k)$, and the goal is to determine a minimum-weight cut whose removal disconnects each s_i from its pair t_i. Multicut is NP-hard to approximate within any constant factor assuming the Unique Games Conjecture [3], and there is a polynomial-time $O(\log k)$-approximation algorithm [7]. The Uniform Metric Labeling problem takes as input a list of possible colors for each node in an edge-weighted graph, and asks for a coloring that respects these lists with the minimum total weight of dichromatic edges; Multiway Cut arises as a special case when the terminals have distinct lists of length 1 and all other nodes can be colored arbitrarily. Kleinberg and Tardos [12] gave a 2-approximation to Uniform Metric Labeling with a tight integrality gap using a geometric relaxation, similar to that of CKR. In the k-Cut problem, we are given only an edge-weighted graph G and a positive integer k, and the goal is to find a minimum-weight cut whose deletion breaks the graph into k components. One can think of this problem as a version of Multiway Cut where the terminals can be chosen freely. The k-Cut problem admits a 2-approximation [16] that is tight [13]. The Steiner Multicut [11] problem takes as input an undirected graph G and subsets $X_1, X_2, \ldots, X_q$ of nodes, and asks for a minimum cut such that each X_i is separated into at least 2 components. A generalization of Steiner Multicut is the Requirement Cut problem [9], where requirements r_i are given for each set X_i, and the goal is to find the minimum cut that cuts each X_i into at least r_i components. The current best algorithms for Requirement Cut are those given in [9, 17], of which we will use the $O(\log k \log q)$ approximation, where $k = |\bigcup_{i=1}^{q} X_i| \leq n$.

1.2 Our results

We introduce generalizations of Multiway Cut, where we are allowed to choose *representatives* from some terminal candidate sets $T_1, \dots, T_q \subseteq V$, and the goal is to find the minimum-weight cut separating these sets *via their representatives.* The variants are distinguished by (A) whether the representative has to be separated from all candidates of the other candidate sets or only from their representatives, and (B) whether there is a single representative for each candidate set or whether the choice of representative is independent for each pair of candidate sets. In order to make it easier to distinguish these problems, we use the following naming rules.

- When the goal is to separate *all* candidates, we use All; for example, the All-to-All problem requires all nodes of T_i to be separated from all nodes of T_j, for each $i \neq j$.
- When the goal is to choose a *single* representative for each candidate set, we use Single, and we denote the chosen representative of T_i by t_i. For example, the Single-to-Single problem requires choosing a representative $t_i \in T_i$ for every $i \in [q]$, and finding a cut that separates t_i from t_j for all $i \neq j$. On the other hand, Single-to-All requires the chosen representative $t_i \in T_i$ to be separated from every node of T_j, for all $i \neq j$.
- When only *some* representative of T_i ought to be separated from some part of T_j for each i, j pair, we use Some, and denote the representative chosen from T_i to be separated from T_j by t_i^j. For example, the Some-to-Some problem asks for a minimum-weight subset of edges such that after deleting these edges, for any pair $i \neq j$, there are nodes $t_i^j \in T_i$ and $t_j^i \in T_j$ that are in different components.
- When there is a *fixed* node that needs to be separated from the candidate sets, we use Fixed, and denote the fixed node by s. In the Fixed-to-Single problem, we are given a fixed node s, and we want a minimum-weight subset of edges such that after deleting these edges, s is separated from at least one element $t_j \in T_j$ for every $j \in [q]$.

These problems are natural generalizations of Multiway Cut that provide various ways to interpolate between problems with fixed terminals like Multiway Cut and problems with freely chosen terminals like k-Cut. Although, as we will discuss later, some of our problems are equivalent or closely related to problems that have already been considered in the literature, a systematic study of this type of generalization has not yet been done, and some of our results (Theorem 3, Theorem 9) require new observations and techniques.

In each problem, we want to minimize over all possible choices of representatives, as well as over all possible subsets of edges. The problem where we need to separate each candidate set from every other, All-to-All, is equivalent to Multiway Cut by contracting each candidate set to a single node. The other problems are not directly reducible to Multiway Cut. We denote by $\alpha \approx 1.2965$ the current best approximation factor for Multiway Cut [18]. The different problems, as well as our results, are summarized in Table 1. The main results that require new techniques are indicated in bold in the table, and are discussed in the next subsection.

1.3 Techniques

Approximation when q is part of the input. We give 2-approximations for Single-to-All and Single-to-Single. For the latter, we first give an exact algorithm on trees, by showing that the feasible solutions have a gammoid structure. This then leads to a 2-approximation for general graphs using the Gomory-Hu tree, which is best possible, since Single-to-Single generalizes the k-Cut problem. Also, we show that the Some-to-Some problem is equivalent to Steiner Multicut, leading to an $O(\log q \cdot \log n)$ approximation in this case.

Table 1 A summary of our results, where $\alpha \approx 1.2965$ [18] is the current best approximation factor for Multiway Cut. The tightness of 2-approximation assumes SSEH, while the other inapproximability results hold assuming P $\neq$ NP. The main results are highlighted in bold.

Problem	Demands	Fixed q	Unbounded q
All-to-All	$T_i - T_j$	α-approx	α-approx
Single-to-All	$t_i - T_j$	**α-approx**	2-approx
Single-to-Single	$t_i - t_j$	α-approx	**Tight 2-approx**
Fixed-to-Single	$s - t_j$	In P	No $o(\log q)$ approx
Some-to-Single	$t_i^j - t_j$	α-approx	No $o(\log q)$ approx
Some-to-Some	$t_i^j - t_j^i$	α-approx	$O(\log q \cdot \log n)$ approx [9]
Some-to-All	$t_i^j - T_j$	**α-approx**	No $o(\log q)$ approx

Approximation for fixed q. Some of the problems with fixed q are directly reducible to solving a polynomial number of Multiway Cut instances. However, this is not the case for Single-to-All and Some-to-All. Our α-approximation algorithms for these are obtained by extending the CKR relaxation to a more general problem that we call Lifted Cut (see Section 3) in such a way that the rounding methods used in [18] still give an α-approximation. Lifted Cut may have independent interest as a class of metric labeling problems that is broader than Multiway Cut but can still be approximated to the same ratio. We then show that for fixed q, problems Single-to-All and Some-to-All are reducible to solving polynomially many instances of Lifted Cut.

Hardness of approximation. We prove hardness of Fixed-to-Single by reducing from Hitting Set. We then reduce Some-to-All, Some-to-Single, and Some-to-Some from Fixed-to-Single to give hardness results for those problems as well.

1.4 Structure of the paper

In Section 2, we present the main tools used in our algorithms and proofs. Section 3 introduces the Lifted Cut problem and describes how to extend the α-approximation of [18] to Lifted Cut. The remaining sections present the results for the problems listed in Table 1.

2 Background

Throughout the paper, we denote the set of non-negative reals by $\mathbb{R}_+$, and use $[k] = \{1, \ldots, k\}$. We use e^i to denote ith elementary vector, and Δ_k denotes the convex hull of $\{e^1, \ldots, e^k\}$, that is, $\Delta_k = \{x \in \mathbb{R}^k : x \geq 0,\ \sum_{i=1}^k x_i = 1\}$.

Given an undirected graph $G = (V, E)$, the edge going between nodes $u, v \in V$ is denoted by (u, v). For a weight function $w : E \to \mathbb{R}_+$ and $C \subseteq E$, we use $w(C) = \sum_{e \in C} w(e)$. The graph obtained by deleting the edges in C is denoted by $G - C$. We denote the set of components of G by $\mathcal{K}(G)$. The boundary of a given subset of nodes $S \subseteq V$ is $\delta(S) = \{(u, v) \in E : u \in S, v \in V \setminus S\}$.

We briefly summarize the background results that we build upon in our proofs.

2.1 The CKR Relaxation and Rounding Methods

For a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_+$ and terminals $S = \{s_1, \ldots, s_k\}$, the CKR relaxation [4] is the following linear program (CKR-LP) which assigns to each node $u \in V$ a geometric location x^u in the k-dimensional simplex.

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v)\in E} w_{u,v} \|x^u - x^v\|_1 \\ \text{subject to} \quad & x^u \in \Delta_k \quad u \in V, \\ & x^{s_i} = e^i \qquad i \in [k]. \end{aligned} \tag{CKR-LP}$$

The original paper of Călinescu, Karloff and Rabani [4] gives a $(3/2 - 1/k)$-approximation algorithm that works as follows. First take a threshold $\rho_i \in (0, 1)$ uniformly at random for each dimension $i \in [k]$. Then take one of the two permutations $\sigma = (1, \ldots, k - 1, k)$ and $(k - 1, k - 2, \ldots, 1, k)$ of the terminals at random (that is, with probability 1/2), assign nodes within a distance $\rho_{\sigma(i)}$ of $x^{s_{\sigma(i)}}$ to the component of $s_{\sigma(i)}$ for $i \in [k - 1]$, and assign the remaining nodes to s_k. We call an algorithm that chooses a permutation of the terminals and then assigns the nodes within some threshold to the terminals in that order a *threshold algorithm*. The analyses of the above linear programming formulation revealed several useful properties of the CKR relaxation. One of these observations is that the edges of the graph may be assumed to be *axis-aligned*. An edge u, v is said to be (i, j)-axis-aligned if x^u and x^v differ only in coordinates i and j. Roughly speaking, any edge that is not axis-aligned can be subdivided into several edges that are axis-aligned, forming a piecewise linear path between x^u and x^v. This observation significantly simplifies the analysis of threshold algorithms, as there are at most two thresholds that can cut any axis-aligned edge. Another useful property is *symmetry*. For any threshold algorithm, there is one that achieves the same guarantees by choosing a uniformly random permutation. See [10, Section 2] for a more detailed discussion of these properties.

Another way of rounding the CKR relaxation is provided by the *exponential clocks* algorithm of Buchbinder, Naor and Schwartz [1]. Their approach can be thought of as choosing a uniformly random point in the simplex, and splitting the simplex by axis parallel hyperplanes that meet at this given point. The algorithm gives the same guarantees as the algorithm of Kleinberg and Tardos [12] for Uniform Metric Labeling. This latter problem takes as input a list of possible colors $\ell(v)$ for each node v in a given graph, and asks for a coloring that respects these lists with the minimum total weight of dichromatic edges. Their relaxation (UML-LP) is similar to the CKR relaxation when there are a total of q colors, but it does not require there to be nodes at every vertex of the simplex.

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v)\in E} w_{u,v} \|x^u - x^v\|_1 \\ \text{subject to} \quad & x^u \in \Delta_q \quad u \in V, \\ & x^v_i = 0 \qquad i \notin \ell(v). \end{aligned} \tag{UML-LP}$$

It is shown in [1, Section 6] that Algorithm 1 gives the same guarantees as the exponential clocks algorithm.

The approximation algorithm of Sharma and Vondrák [18] for Multiway Cut randomly chooses between four different algorithms of the above two types with some careful analysis to achieve an α-approximation, where $\alpha \approx 1.2965$.

Algorithm 1 The Kleinberg-Tardos Algorithm for Uniform Metric Labeling.

```
Input: A graph $G = (V, E)$, weights $w : E \to \mathbb{R}_+$, labels $\ell : V \to \mathcal{P}([q])$, and an LP solution $x^u$ for each $u \in V$.
Output: A solution to Uniform Metric Labeling.
while $\exists u \in V$ s.t. $u$ is unassigned do
    Pick a label $i \in [q]$ uniformly at random, and a threshold $\rho \sim unif[0, 1]$.
    Assign label $i$ to any unassigned $u \in V$ with $x^u_i \geq \rho$.
end while
```

2.2 Other Relevant Tools

Our hardness of approximation results are based on two different complexity assumptions. The $o(\log q)$ inapproximability results hold assuming $\mathrm{P} \neq \mathrm{NP}$, based on the hardness of approximating Hitting Set proved by Dinur and Steurer [6]. The other complexity assumption that we use is the *Small Set Expansion Hypothesis* (SSEH), a core hypothesis for proving hardness of approximation for problems that do not have straightforward proofs assuming the Unique Games Conjecture (UGC). It implies the UGC, and we will use it as evidence against a $(2 - \varepsilon)$-approximation, for any $\varepsilon > 0$, for k-Cut [13]. For completeness, we include the relevant theorems here.

▶ **Theorem 1** ([6, 14]). *For any fixed $0 < \alpha < 1$, Hitting Set cannot be approximated in polynomial time within a factor of $(1 - \alpha) \ln N$ on inputs of size N, unless $\mathrm{P} = \mathrm{NP}$.*

▶ **Theorem 2** ([13]). *Assuming the Small Set Expansion Hypothesis, it is NP-hard to approximate k-Cut to within $(2 - \varepsilon)$ factor of the optimum, for any constant $\varepsilon > 0$.*

From matroid theory, we use the notion of gammoids. A *gammoid* $M = (D, S, T)$ is a matroid defined by a digraph $D = (V, E)$, a set of source nodes $S \subseteq V$, and a set of target nodes $T \subseteq V \setminus S$. A set $X \subseteq T$ is independent in M if there exist $|X|$ node-disjoint paths from elements of S into X. Optimizing over a gammoid, as with any other matroid, can be done efficiently using the greedy algorithm.

Finally, Gomory-Hu (GH) tree [8] is a standard tool in graph cut algorithms. The *GH tree* of a graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_+$ is a tree $T = (V, F)$ together with weight function $w_T : F \to \mathbb{R}_+$ that encodes the minimum-weight $s - t$ cuts for each pair s, t of nodes in the following sense: the minimum w_T-weight of an edge on the $s - t$ path in T is equal to the minimum w-weight of a cut in G separating s and t. Furthermore, the two components of the tree obtained by removing the edge of minimum w_T-weight on the path give the two sides of a minimum w-weight $s - t$ cut in G.

3 Lifted Cuts

The goal of this section is to show that the following restriction of the Uniform Metric Labeling relaxation to a one-dimensional lifting of the CKR relaxation admits an α-approximation to its integer optimum. We define the lifted cut problem Lifted Cut, which takes as input a graph $G = (V, E)$ with edge-weights $w : E \to \mathbb{R}_+$, *fixed* terminals $S = \{s_1, s_2, \ldots, s_q\} \subseteq V$, and a list of possible colors for each node $\ell : V \to \mathcal{P}[q + 1]$, the power set of $[q + 1]$, satisfying the following two conditions:

(A) $\ell(s_i) = \{i\}$ for $i = 1 \ldots q$,

(B) $q + 1 \in \ell(v)$ for all $v \in V \setminus S$.

The goal is then to assign a color to each node from its list such that the total weight of dichromatic edges is minimized. We call the following linear programming relaxation of the Lifted Cut problem LIFT-LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{(u,v)\in E} w_{u,v}\|x^u - x^v\|_1 \\ \text{subject to} \quad & x^u \in \Delta_{q+1}, \quad u \in V \\ & x_i^u = 0 \qquad\quad i \notin \ell(u). \end{aligned}$$

Condition A ensures that the set S indeed defines terminals that vertices of the simplex are assigned to, as in Multiway Cut, but Condition B offers a relaxation, allowing a vertex of the simplex to not be assigned to any terminal. This condition gives an additional dimension to the simplex (see Figure 1), while still preserving the approximation guarantees given by the rounding algorithms for CKR.

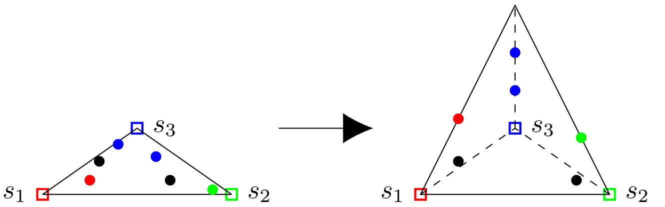


Figure 1 An example of the original CKR relaxation in relation to our extended LIFT-LP on the case for $t_i - T_j$. The point colors represent the different candidate sets.

▶ **Theorem 3.** *The rounding scheme of [18], when applied to LIFT-LP using Algorithm 1 in place of the exponential clocks algorithm, and with the modification that only the first q coordinates are permuted in the threshold algorithms while coordinate $q+1$ is always left last, gives an α-approximation to Lifted Cut.*

Proof. First, we have to argue that the threshold algorithms give feasible solutions to Lifted Cut (for Algorithm 1, this follows since Lifted Cut is a metric labeling problem). In all algorithms, s_i is assigned to the ith component, since x^{s_i} is the ith vertex of the simplex. For other nodes $v \in V \setminus S$, $x_i^v = 0$ guarantees that v is not assigned to the ith component if $i \notin \ell(v)$. Here, we use the fact that the $(q+1)$st component is the only one for which there is no threshold. Although it is possible that $x_{q+1}^v = 0$ and v is still assigned to the $(q+1)$st component, this is not a problem, because $q+1 \in \ell(v)$ by definition.

To prove that we have an α-approximation, we need to show that the relevant bounds that are used in the analysis of the four algorithms mentioned in Sharma-Vondrák [18] carry through to this modified LIFT-LP. We give a sketch here, but the details are written out more carefully in Appendix B. We consider the two types of algorithms (i.e. threshold and exponential clocks) separately.

It was observed in [1] that the exponential clocks algorithm can be replaced by the 2-approximation for the Uniform Metric Labeling problem of Kleinberg-Tardos [12]. Since LIFT-LP corresponds to a Uniform Metric Labeling problem, the bound in [1, Lemma 3] remains valid in our case. Since this is the relevant bound for the exponential clocks algorithm used in the analysis, we can conclude that Algorithm 1 for LIFT-LP gives the same guarantees.

The other algorithms we need to consider are the threshold algorithms. These assume that there is a node at every vertex of the simplex, which is not necessarily true for the LIFT-LP as no variable needs to be at e^{q+1}. We can however use that there is only *one* such vertex, and change the order of the terminals so that this vertex is cut last. We can then use the analysis in [18] and [1] of the CKR relaxation for $k = q + 1$, as sketched below.

The threshold algorithms first choose a random permutation of the nodes to achieve some symmetry, which is necessary for *only* the first $k - 1$ terminals. The last terminal, which is just assigned the remaining nodes, *does not have its own threshold.* In each of the Single Threshold, Descending Threshold and Independent Threshold algorithms of [18], thry prove results for the first two indices, and then argue that these hold for any pair of indices by symmetry. This is only directly clear for pairs in the first $k - 1$. However, when we consider an (i, k)-axis-aligned edge for some $i \in [k - 1]$, the probability of cutting this edge can only be smaller as there is one less threshold to cut it; see [1, Remark 2] for a discussion. This reasoning holds even when there is no terminal at the kth vertex of the simplex.

Thus, the rounding scheme of [18], with the modifications of using Algorithm 1 rather than exponential clocks and only permuting the first q coordinates, gives an α-approximation for Lifted Cut. For completeness, we include the relevant algorithms and lemmas from [18], with the appropriate modifications, in Appendix B. ◀

4 Single-to-All Problem

In this problem, we are looking for a *single* representative from each candidate set that will be separated from *every* candidate in other candidate sets. This includes the other representatives, making the problem very similar to Multiway Cut once the representatives are chosen. A key difference is that the optimal partition may have $q + 1$ components.

We first look at the case where q is constant.

▶ **Theorem 4.** *There is an α-approximation algorithm for Single-to-All when q is fixed.*

Proof. First, guess the representative t_i for each $i \in [q]$. As there are only $\prod_{i=1}^{q} |T_i| \leq n^q$ possible choices, this is polynomial in n for fixed q. If a representative t_i is in T_j for some $j \neq i$, then there is obviously no solution. Otherwise, for a fixed choice of representatives, Single-to-All is an instance of Lifted Cut. To see this, observe that the problem is equivalent to the Uniform Metric Labeling problem obtained by fixing the labels $\ell(v)$ for $v \in V$ as follows:

1. If $v = t_i$ for some i, then set $\ell(v) := \{i\}$.
2. Otherwise, if $v \in T_i \setminus \{t_i\}$ for a unique i, then set $\ell(v) := \{i, q + 1\}$.
3. If $v \in T_i \cap T_j \setminus \{t_i, t_j\}$ for $i \neq j$, then set $\ell(v) := \{q + 1\}$.
4. Finally, if $v \in V \setminus \bigcup_{i\in[q]} T_i$, then set $\ell(v) := [q + 1]$.

This is an instance of Lifted Cut: Condition A is a clear consequence of the first rule, and since any node that is not a representative has $q + 1$ as one of its labels, Condition B follows as well. Therefore, Theorem 3 leads to an α-approximation. ◀

Following the idea of the classical 2-approximation for Multiway Cut discussed in the introduction, there is a simple 2-approximation when q is arbitrary.

▶ **Theorem 5.** *There is a 2-approximation algorithm for Single-to-All.*

Proof. For each candidate set T_i, let $t_i \in T_i$ be a node for which the minimum-weight cut separating t_i from $\cup_{j\neq i} T_j$ is as small as possible, and let C be the union of these isolating cuts. To see that the solution is within a factor 2 of the optimum, consider an optimal solution to

Algorithm 2 Greedy algorithm for Single-to-Single on trees.

```
Input: A tree $G = (V, E)$, weights $w : E \to \mathbb{R}_+$, candidates $T_1, \dots, T_q \subseteq V$.
Output: A minimum-weight good cut $C \subseteq E$.
Set $C \leftarrow \emptyset$.
while $|C| < q - 1$ do
    $e \leftarrow \arg\min\{w(e) : e \notin C, C + e \text{ is } good\}$
    $C = C + e$
end while
```

Single-to-All and let $V_1, V_2, \dots, V_q, V_{q+1}$ denote the components after its deletion, where V_{q+1} may be empty and the components are ordered by the index of the representative they contain. The boundary of each V_i is an isolating cut of some candidate in T_i, which the algorithm minimized. Summing up the weights of the boundaries, we count each edge twice, and the theorem follows. ◀

5 Single-to-Single Problem

In this problem, we are looking for a *single* representative from each candidate set together with a minimum multiway cut separating them. Note that when $T_1 = T_2 = \ldots = T_q = V$, Single-to-Single generalizes k-Cut where we seek the minimum-weight cut that partitions the graph into k parts. It is known that k-Cut is hard to approximate within a factor of $2 - \varepsilon$ for any $\varepsilon > 0$, assuming SSEH [13].

▶ **Theorem 6.** *There is an α-approximation for Single-to-Single when q is fixed.*

Proof. When q is fixed, one can iterate through all the $O(n^q)$ possible choices of representatives, approximate the corresponding Multiway Cut instance, and choose the best one. ◀

For general q, it is helpful to first look at the case where G is a tree. We show that in this special case, the problem reduces to finding the minimum cost basis of a gammoid. We call a cut $C \subseteq E$ *good* if $G - C$ has a valid set of representatives, that is, if we can choose $|C| + 1$ representatives that form a partial transversal of the candidate sets, and each component of $G - C$ contains a single representative from this partial transversal. The algorithm is presented as Algorithm 2.

▶ **Theorem 7.** *Algorithm 2 computes an optimal solution to Single-to-Single on trees.*

Proof. We prove the statement by showing that the problem is equivalent to optimizing over a gammoid. We construct a directed graph as follows. Let $r \in V$ be an arbitrary root node, and orient the edges of the tree towards r. For a non-root node v, we denote the unique arc leaving v by $e(v)$ and define the cost of v to be $w(e(v))$. Furthermore, for each set T_i, we add a node s_i together with arcs from s_i to the candidates in T_i.

Let D denote the digraph thus obtained, $S \coloneqq \{s_1, \dots, s_q\}$, and $T \coloneqq V$, and consider the gammoid $M = (D, S, T)$. The key observation is the following.

▷ Claim 8. For a set $Z \subseteq V \setminus \{r\}$, $C = \{e(v) : v \in Z\}$ is a good cut if and only if $Z \cup \{r\}$ is independent in M.

Proof. For the forward direction, assume that $C = \{e(v) : v \in Z\}$ forms a good cut. Let $Z = \{v_1, \dots, v_p\}$. Without loss of generality, we may assume that the candidate sets having a valid set of representatives in $G - C$ are $T_1, \dots, T_p, T_{p+1}$, where v_i is in the component of the representative t_i of T_i and r is in the same component as the representative t_{p+1} of T_{p+1}.

Algorithm 3 Approximation algorithm for Single-to-Single on graphs.

```
Input: A graph $G = (V, E)$, weights $w : E \to \mathbb{R}_+$, candidates $T_1, \dots, T_q \subseteq V$.
Output: A feasible cut $C \subseteq E$.
Compute the Gomory-Hu tree $H$ of $G$.
Run Algorithm 2 on $H$.
Return the union of the cuts corresponding to edges found in Step 2.
```

For $i \in [p]$, the edge (s_i, t_i) and the path t_i-v_i in the tree form an s_i-v_i path; similarly, the edge (s_{p+1}, t_{p+1}) and the path t_{p+1}-r in the tree form an s_{p+1}-r path. Furthermore, these paths are pairwise node-disjoint, since they use different connected components of $G - C$.

For the other direction, assume that $Z \cup \{r\}$ is independent in M, and let $Z = \{v_1, \dots, v_p\}$. Without loss of generality, we may assume that there are pairwise node-disjoint paths from s_i to v_i for $i \in [p]$ together with a path from s_{p+1} to r. Let $t_i \in T_i$ be the first node on the path starting from s_i for $i \in [p + 1]$. Then $\{t_1, \dots, t_{p+1}\}$ form a valid system of distinct representatives for the cut C as each of these nodes are in a separate component of $G - C$. ◁

By Claim 8, a minimum-weight good cut can be determined using the greedy algorithm for matroids, which is exactly what Algorithm 2 is doing. ◀

Algorithm 2 solves the special case when G is a tree. The classical $(2-2/k)$ approximation for Multiway Cut [5] uses 2-way cuts coming from the Gomory-Hu tree, and so does the $(2 - 2/k)$ approximation for k-Cut [16]. We follow a similar approach in Algorithm 3. The algorithm can be interpreted as taking the minimum edges in the GH tree as long as they allow a valid system of representatives. The algorithm is presented as Algorithm 3.

▶ **Theorem 9.** *Algorithm 3 computes a $(2 - 2/q)$ approximation to Single-to-Single on arbitrary graphs.*

Proof. Let OPT be the optimal solution with representatives $t_1^*, \dots, t_q^*$, and components $V_1^*, \dots, V_q^*$, where V_q^* has the maximum weight boundary $\delta(V_q^*)$. Let also H be the GH tree of G.

We transform OPT into a solution OPT_{GH} on H, losing at most a factor of $(2 - 2/q)$. We do this by repeatedly removing the minimum weight edge in $E(H)$ that separates a pair among the representatives $t_1^*, \dots, t_q^*$ that are in the same component of H. More precisely, we start with $H_0 = H$, and take the minimum-weight edge $e_1 \in E(H_0)$ separating some pair of representatives t_i^*, t_j^* in OPT that are in the same component of H_0. Define the edge $f_1 = (t_i^*, t_j^*)$. Then we construct $H_1 = H_0 - e_1$, and repeat this process to get a sequence of edges $e_1, e_2, \dots, e_{q-1}$ and a tree of representative pairs $F = (\{t_1^*, \dots, t_q^*\}, \{f_1, \dots, f_{q-1}\})$.

Direct the edges of F away from t_q^*, and reorder the edges such that f^1 is the edge going into t_1^*, f^2 into t_2^*, and so on. Let e^i be the edge of the GH tree corresponding to f^i, i.e., the minimum weight edge of the path between the two endpoints of f^i, and let $U(e^i)$ be the cut corresponding to e^i for each i. Then the boundary of each component satisfies $w(\delta(V_i^*)) \geq w(U(e^i))$, as $\delta(V_i^*)$ separates the two representatives in f^i as well, and $U(e^i)$ is the minimum-weight cut between these.

Let the solution OPT_{GH} be $\bigcup_{i \in [q-1]} U(e_i)$, ALG the cut found by the algorithm, ALG_{GH} the corresponding edges in the GH tree H, and w_H the weight function on H. Then

$$w(ALG) \leq w_H(ALG_{GH}) \leq w_H(OPT_{GH}) = \sum_{i=1}^{q-1} w(U(e^i)) \leq \sum_{i=1}^{q-1} w(\delta(V_i^*))$$

$$\leq (1 - 1/q) \sum_{i=1}^{q} w(\delta(V_i^*)) \leq (2 - 2/q) w(OPT).$$ ◀

6 Fixed-to-Single, Some-to-Single, Some-to-Some, and Some-to-All Problems

In this section, we combine the study of four problems, as the techniques are similar.

6.1 Hardness of approximation

All four have similar proofs of hardness of approximation, which we state here but leave the proofs to appendix A for brevity.

▶ **Theorem 10.** *For general q, Fixed-to-Single, Some-to-Single and Some-to-All are at least as hard to approximate as Hitting Set.*

We omit the Some-to-Some problem from Theorem 10 because it follows as a corollary to Theorem 11, which states that it is equivalent to the known Steiner Multicut problem. The conditional $o(\log n)$ inapproximability was already proved for Steiner Multicut in [15], using similar instances as those in our proof of Theorem 10. The Some-to-Some problem asks to find a cut such that each pair of candidate sets have at least one element in separate components, where this choice can depend on the pair.

▶ **Theorem 11.** *The Some-to-Some problem is equivalent to Steiner Multicut.*

Proof. To reduce from Steiner Multicut, we are given q subsets $X_0, X_1, \ldots, X_{q-1}$ of nodes of a graph G, each of which needs to be cut into at least two components. We construct a Some-to-Some instance on the same graph with $2q$ candidate sets $T_0, T_1, \ldots, T_{2q-1}$, where $T_i = X_{\lfloor i/2 \rfloor}$ for $0 \le i \le 2q-1$. Then, for each $j = 0 \ldots q-1$, the condition that T_{2j} must be separated from T_{2j+1} ensures that there are two nodes $t_{2j}^{2j+1}, t_{2j+1}^{2j} \in X_j$ that are in different components. In other words, the solution is a minimal cut that, once removed, divides each set into at least two components. If the conditions of Some-to-Some hold for T_{2j} and T_{2j+1} for any j, then they hold automatically for any other pair of candidate sets too, because once a set has elements in two components, at least one of them will be in a different component than some element of any given candidate set.

For the other direction, we are given q subsets $T_1, \ldots, T_q$ of nodes of a graph G as a Some-to-Some instance. We then make a Steiner Multicut instance with $\binom{q}{2}$ vertex sets indexed by pairs $i, j \in [q]^2$ with $i \neq j$. The set $X_{i,j}$ will then be $T_i \cup T_j$, which means any valid Steiner Multicut solution C will split each of these sets into at least two components. We claim C is a valid Some-to-Some solution as well. Let $v_{i,j}, u_{i,j} \in X_{i,j}$ be in different components of $G \setminus C$. Then one of the following cases must hold:

1. $v_{i,j} \in T_i$ and $u_{i,j} \in T_j$. In this case, let $t_j^i := u_{i,j}$ and $t_i^j := v_{i,j}$.
2. $u_{i,j} \in T_i$ and $v_{i,j} \in T_j$. In this case, let $t_j^i := v_{i,j}$ and $t_i^j := u_{i,j}$.
3. $u_{i,j}, v_{i,j} \in T_i$. Then either
   (i) all of T_j is in the same component of $G \setminus C$ as $u_{i,j}$, in which case let $t_i^j := v_{i,j}$, and set t_j^i to an arbitrary element of T_j, or
   (ii) some vertex $w \in T_j$ is in a different component of $G \setminus C$ than $u_{i,j}$, in which case let $t_j^i := w$, and $t_i^j := u_{i,j}$.
4. Similarly, if $u_{i,j}, v_{i,j} \in T_j$, then either
   (i) all of T_i is in the same component of $G \setminus C$ as $u_{i,j}$, in which case let $t_j^i := v_{i,j}$, and set t_i^j to an arbitrary element of T_i, or
   (ii) some vertex $w \in T_i$ is in a different component of $G \setminus C$ than $u_{i,j}$, in which case let $t_i^j := w$, and $t_j^i := u_{i,j}$.

In all cases above, t_j^i is in a different component than t_i^j on $G \setminus C$, so C is a valid Some-to-Some solution. Any Some-to-Some solution is clearly also a solution for this Steiner Multicut instance, so the optimal cut is the same for both. ◀

6.2 Fixed q

The techniques when q is fixed differ, suggesting that the problems themselves are quite different, despite the apparent similarities.

Fixed Terminal

The Fixed-to-Single problem is slightly different from the others, as the goal here is to choose representatives that need to be separated only from a *fixed* node s. In this case, the problem can be solved efficiently.

▶ **Proposition 12.** *For fixed q, Fixed-to-Single can be solved in polynomial time.*

Proof. In this case, one can iterate through all possible choices of representatives, of which we have at most n^q, calculate a minimum two-way $s - \{t_i : i \in [q]\}$ cut for each, and then take the best of all solutions. ◀

Some to single/some

The Some-to-Single and Some-to-Some problems both become Multicut instances with a constant number of terminals in this case, which gives the following theorem:

▶ **Theorem 13.** *For fixed q, there is an α-approximation to Some-to-Single and Some-to-Some.*

Proof. We will use the α-approximation to Multiway Cut on a polynomial number of instances with fixed terminals. We begin with the Some-to-Single problem. In this problem, the goal is to choose a single representative t_j for each $j \in [q]$ together with some candidate $t_i^j \in T_i$ for each pair $i \neq j$ that are then separated by the cut.

When q is fixed, one can guess the representatives t_i^j and t_j to get a set of terminals S together with some separation demands on them. The number of such terminals can be bounded as $|S| \leq q^2$. A slightly more careful analysis shows that the number of different t_i^j nodes for a candidate set T_i can be bounded by two. Thus, we only have to guess three representatives from each T_i, implying $|S| \leq 3q$. Either way, the number of guesses for S is polynomial in n. Each guess of S defines a minimum multicut problem since we know which pairs of representatives have to be separated. We can compute an α-approximation to each of these Multicut problems by enumerating all possible partitions of S (of which there are exponentially many in q) that satisfy the multicut demands, collapsing the partitions into fixed terminals, and calculating an α-approximating multiway cut for each.

For the Some-to-Some problem, again guess the representatives t_i^j for each $i, j \in [q], i \neq j$ to get a set of terminals S together with some separation demands on them. Since any candidate set with terminals in different components already has at least one element in a separate component for any other candidate set, the number of such terminals can be bounded by $|S| \leq 2q$. For each fixed S, we can find an α-approximation the same way as above. ◀

Combining this approximation for Some-to-Some with Theorem 11 gives the current best approximation for Steiner Multicut in the regime where the number of candidates depends on n, and the number of sets is constant.

Some to all

Finally, we consider the Some-to-All problem, which asks to find representatives $t_i^j \in T_i$ for each pair $i, j \in [q]$ and a minimum-weight cut $C \subseteq E$ such that t_i^j is separated from *all* of T_j in $G - C$. The case for constant q uses the tool from Section 3.

▶ **Theorem 14.** *There is an α-approximation for Some-to-All when q is fixed.*

Proof. We guess all representatives t_i^j; there are at most n^{q^2} possible choices, which is polynomial if q is fixed. Note that we may assume that $t_i^j \neq t_j^\ell$ if $i \neq j$, otherwise there is obviously no solution. We also guess a valid partition $V_1, \ldots, V_{q_1}$ of these representatives into q_1 components, where $2 \leq q_1 \leq q^2$ (validity means that t_i^j and t_j^ℓ are in different classes of the partition if $i \neq j$). The number of such partitions is exponential in q, but we can still enumerate them when q is fixed (note that this is not a partition of V, but a partition of the set of all chosen representatives, which is a vertex set of size at most q^2). For such a partition, the problem becomes an instance of $(q_1 + 1)$-dimensional Lifted Cut with the following labels.

a) If $v \in V_k$ for some $k \in [q_1]$, then set $\ell(v) := \{k\}$.

b) Otherwise, if $v \in T_j$, then we must ensure that the label cannot be any partition containing some t_i^j. In other words, set $\ell(v) := \{1, 2, \ldots, q_1 + 1\} \setminus \{k : v \in T_j \text{ and } t_i^j \in V_k \text{ for some } i, j\}$.

c) Finally, if $v \in V \setminus \bigcup_{i \in [q]} T_i$, then set $\ell(v) := [q_1 + 1]$.

Conditions A and B of Lifted Cut are not difficult to verify. The solution to this problem is a solution to Some-to-All. Indeed, consider the partition given by a solution to Lifted Cut, which is an extension of the partition $V_1, \ldots, V_{q_1}$ by condition a, with an additional class for label $q_1 + 1$. Condition b then ensures, for a given t_i^j, that the component of t_i^j cannot contain any element of T_j. Thus, Theorem 3 gives an α-approximation for Some-to-All as well. ◀

References

1 Niv Buchbinder, Joseph (Seffi) Naor, and Roy Schwartz. Simplex partitioning via exponential clocks and the multiway cut problem. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 535–544, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488675.

2 Kristóf Bérczi, Karthekeyan Chandrasekaran, Tamás Király, and Vivek Madan. Improving the integrality gap for multiway cut. *Mathematical Programming*, 183(1-2):171–193, 2020. doi:10.1007/s10107-020-01485-2.

3 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D Sivakumar. On the hardness of approximating multicut and sparsest-cut. *computational complexity*, 15:94–114, 2006.

4 Gruia Călinescu, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences*, 60(3):564–574, 2000. doi:10.1006/jcss.1999.1687.

5 E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994. doi:10.1137/S0097539792225297.

6 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 624–633, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591884.

7 Naveen Garg, Vijay V Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi) cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

8 Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

9 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. An improved approximation algorithm for requirement cut. *Operations Research Letters*, 38(4):322–325, 2010. doi:10.1016/j.orl.2010.02.009.

10 David R. Karger, Philip Klein, Cliff Stein, Mikkel Thorup, and Neal E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. *Mathematics of Operations Research*, 29(3):436–461, 2004. doi:10.1287/moor.1030.0086.

11 Philip N Klein, Serge A Plotkin, Satish Rao, and Eva Tardos. Approximation algorithms for steiner and directed multicuts. *Journal of Algorithms*, 22(2):241–269, 1997.

12 J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and Markov random fields. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 14–23, 1999. doi:10.1109/SFFCS.1999.814572.

13 Pasin Manurangsi. Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis. *Algorithms*, 11(1):10, 2018.

14 Dana Moshkovitz. The projection games conjecture and the np-hardness of ln n-approximating set-cover. In *APPROX-RANDOM*, pages 276–287. Springer, 2012.

15 Viswanath Nagarajan and R Ravi. Approximation algorithms for requirement cut on graphs. *Algorithmica*, 56:198–213, 2010.

16 Huzur Saran and Vijay V. Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995. doi:10.1137/S0097539792251730.

17 Roy Schwartz and Yotam Sharoni. Approximating Requirement Cut via a Configuration LP. In Jarosław Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.53.

18 Ankit Sharma and Jan Vondrák. Multiway cut, pairwise realizable distributions, and descending thresholds. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 724–733, 2014.

A Hardness Proofs for Fixed-to-Single, Some-to-Single, and Some-to-All

Here we prove Theorem 10. In the Hitting Set problem, the input is a family $S_1, \dots, S_m$ of sets, and the goal is to find a smallest set of elements that intersects each of them.

▶ **Proposition 15.** *For general q, Fixed-to-Single is at least as hard to approximate as Hitting Set.*

Proof. To reduce Hitting Set to Fixed-to-Single, we create a graph G by adding a node s together with edges of weight 1 from s to the elements of the ground set. Thus we get a star with center s, and choose the candidate sets to be the sets $S_1, \dots, S_m$. Then, minimizing the number of edges needed to separate at least one node of each S_i from s is equivalent to finding a minimum hitting set; see Figure 2 for an illustration. Note that the reduction is approximation factor preserving. Since there is no $o(\log m)$ approximation for Hitting Set assuming $\mathrm{P} \neq \mathrm{NP}$ [6,14], the hardness of Fixed-to-Single follows. ◀

▶ **Proposition 16.** *For general q, Some-to-Single is at least as hard to approximate as Fixed-to-Single.*

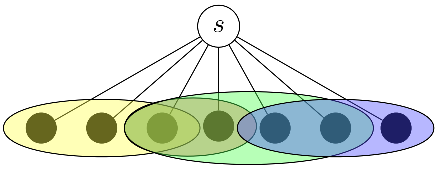


Figure 2 A picture of the reduction for Fixed-to-Single from Hitting Set.

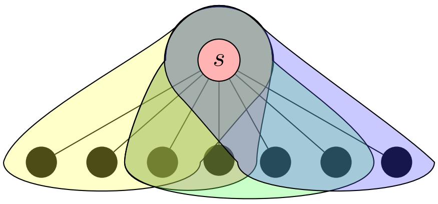


Figure 3 A picture of the reduction from Fixed-to-Single to Some-to-Single.

Proof. Given an instance $s, T_1, \ldots, T_q$ of Fixed-to-Single, we create an instance of Some-to-Single with $T'_i \coloneqq T_i \cup \{s\}$ for $i \in [q]$, and $T'_{q+1} \coloneqq \{s\}$; see Figure 3 for an example.

Given a solution to the Fixed-to-Single instance, we can obtain a solution of the same weight to the Some-to-Single instance by keeping the representatives t_i for $i \in [q]$, setting $t_{q+1} = t^i_{q+1} \coloneqq s$ for $i \in [q]$, $t^{q+1}_i \coloneqq t_i$ for $i \in [q]$, and $t^j_i \coloneqq s$ for $i, j \in [q], i \neq j$.

For the other direction, we observe that each t_j $(j \in [q])$ must be separated from s in a solution of the Some-to-Single instance. Thus, we obtain a solution with the same weight for the Fixed-to-Single if we keep the same representatives t_j $(j \in [q])$. ◀

▶ **Proposition 17.** *For general q, Some-to-All is at least as hard to approximate as Fixed-to-Single.*

Proof. Given an instance of Fixed-to-Single with sets $T_1, \ldots, T_q$ on a graph $G = (V, E)$ where $q \geq 2$, we construct a Some-to-All instance as follows. We add additional nodes $V_0 = \{s_1, s_2, \ldots, s_q\}$, $G' = (V \cup V_0, E)$, $T'_i = T_i \cup \{s_i\}$ for $i = 1 \ldots q$, and $T'_{q+1} = \{s, s_1, s_2, \ldots, s_q\}$; see Figure 4 for an example.

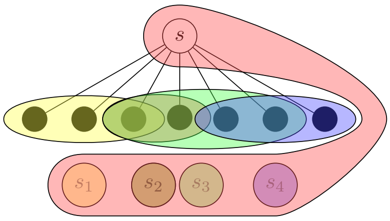


Figure 4 A picture of the reduction from Fixed-to-Single. The candidate set T_1 is in yellow, T_2 in brown, T_3 in green, T_4 in blue, and T_5 in red.

Given a Fixed-to-Single solution with representatives $t^*_1, \ldots, t^*_q$, we get a solution to this instance as follows: $t^j_{q+1} = s_{(j+1) \bmod q}$ for $j \in [q+1]$; if $i \in [q]$, then $t^{q+1}_i = t^*_i$, and $t^j_i = s_i$ for $j \in [q]$. Then the same cut will separate each t^j_i from all of T'_j, and have the same weight.

Given an optimal solution to the Some-to-All instance, we can assume without loss of generality that $t^j_i = s_{(j+1) \bmod q}$ when $i = q+1$, and $t^j_i = s_i$ when $i \neq q+1, j \neq q+1$, as these are separated from the corresponding T'_j in G'. Then we can get a Fixed-to-Single solution by setting $t_j = t^{q+1}_j$ for all $j \in [q]$ and removing the same edges. This reduction preserves approximation, as the solutions have the same weight. ◀

B Details of the Threshold Algorithms

For completeness, we include a detailed description of the α-approximation algorithm for lifted cut. This is just a collection of the results of Sharma and Vondrák [18], but understanding this is necessary for the proof of Theorem 3. The content of this section can be found in more detail in [18], the only modifications we make are to perform the rounding in $k+1$ dimensions and make more clear the role of the $(k+1)$st vertex.

First we describe the three threshold rounding schemes: Single Threshold, Descending Thresholds, and Independent Thresholds. These are described in Algorithms 4, 5, and 6, respectively. Each scheme is given a solution to LIFT-LP, and rounds it to an integer solution by assigning vertices to terminals. The Single Threshold Scheme takes as input some distribution with probability density ϕ, Descending Thresholds some distribution with density ψ, and Independent Thresholds with density ξ. Finally, these schemes are combined with appropriate parameters along with Algorithm 1 according to Algorithm 7, which takes additionally parameters $b, p_1, p_2, p_3, p_4 \in [0,1]$, along with some probability density ϕ.

Algorithm 4 The Single Threshold Rounding Scheme.

```
Choose threshold $\theta \in [0,1)$ with probability density $\phi(\theta)$.
Choose a random permutation $\sigma$ of $[k]$.
for all $i \in [k]$ do
    For any unassigned $u \in V$ with $x^u_{\sigma(i)} \geq \theta$, assign $u$ to terminal $\sigma(i)$.
end for
Assign all remaining unassigned vertices to terminal $k+1$
```

Algorithm 5 Descending Thresholds Rounding Scheme.

```
For each $i \in [k]$, choose threshold $\theta_i \in [0,1)$ with probability density $\psi(\theta)$.
Choose a random permutation $\sigma$ of $[k]$ such that $\theta_{\sigma(1)} \geq \theta_{\sigma(2)} \geq \ldots \geq \theta_{\sigma(k)}$.
for all $i \in [k]$ do
    For any unassigned $u \in V$ with $x^u_{\sigma(i)} \geq \theta_{\sigma_i}$, assign $u$ to terminal $\sigma(i)$.
end for
Assign all remaining unassigned vertices to terminal $k+1$
```

Algorithm 6 Independent Threshold Rounding Scheme.

```
For each $i \in [k]$, choose independently threshold $\theta_i \in [0,1)$ with probability density $\xi(\theta)$.
Choose a uniformly random permutation $\sigma$ of $[k]$.
for all $i \in [k]$ do
    For any unassigned $u \in V$ with $x^u_{\sigma(i)} \geq \theta_{\sigma(i)}$, assign $u$ to terminal $\sigma(i)$.
end for
Assign all remaining unassigned vertices to terminal $k+1$
```

The following three Lemmas are key to the analysis of Algorithm 7. The cut density for an edge of type (i,j) located at $(u_1, u_2, \ldots, u_{k+1}) \in \Delta_{k+1}$ is the limit of the probability that the given threshold scheme assigns $(u_1, u_2, \ldots, u_{k+1})$ and $(u_1+\varepsilon, u_2-\varepsilon, \ldots, u_{k+1})$ to different terminals, normalized by ε as $\epsilon \to 0$.

Algorithm 7 The Sharma-Vondrák Rounding Scheme.

```
With probability $p_1$, choose the Kleinberg-Tardos Rounding Scheme (Algorithm 1).
With probability $p_2$, choose the Single Threshold Rounding Scheme (Algorithm 4) with probability density $\phi$.
With probability $p_3$, choose the Descending Threshold Rounding Scheme (Algorithm 5), where the thresholds are chosen uniformly in $[0, b]$.
With probability $p_4$, choose the Independent Threshold Rounding Scheme (Algorithm 6), where the thresholds are chosen uniformly in $[0, b]$.
```

▶ **Lemma 18** (Lemma 5.1 in [18]). *Given a point $(u_1, u_2, \dots, u_{k+1}) \in \Delta_{k+1}$ and the parameter b of Algorithm 7, let $a = \frac{1-u_i-u_j}{b}$. If $a > 0$, the cut density for an edge of type (i, j), where $i \neq j$ are indices in $[k+1]$ located at $(u_1, u_2, \dots, u_{k+1})$ under the Independent Thresholds Rounding Scheme with parameter b is at most*

- $\frac{2(1-e^{-a})}{ab} - \frac{(u_i+u_j)(1-(1+a)e^{-1})}{a^2b^2}$, *if all the coordinates $u_1, u_2, \dots, u_{k+1}$ are in $[0, b]$.*
- $\frac{(a+e^{-a}-1)}{a^2b}$, *if $u_i \in [0, b], u_j \in (b, 1]$ and $u_\ell \in [0, b]$ for all other $\ell \in [k] \setminus \{i, j\}$.*
- $\frac{1}{b} - \frac{(u_i+u_j)}{6b^2}$, *if $u_i, u_j \in [0, b]$ and $u_\ell \in (b, 1]$ for some other $\ell \in [k] \setminus \{i, j\}$.*
- $\frac{1}{3b}$, *if $u_i \in [0, b], u_j \in (b, 1]$ and $u_\ell \in [0, b]$ for some other $\ell \in [k] \setminus \{i, j\}$.*
- 0, *if $u_i, u_j \in (b, 1]$.*

For $a = 0$, the cut density is given by the limit of the expressions above as $a \to 0$.

▶ **Lemma 19** (Lemma 5.2 in [18]). *For an edge of type (i, j) located at $(u_1, u_2, \dots, u_{k+1})$, where $i \neq j$ are indices in $[k+1]$, the cut density under the Single Threshold Rounding Scheme is at most*

- $\frac{1}{2}\phi(u_i) + \phi(u_j)$, *if $u_\ell \leq u_i \leq u_j$ for all other $\ell \in [k] \setminus \{i, j\}$.*
- $\frac{1}{3}\phi(u_i) + \phi(u_j)$, *if $u_i < u_\ell \leq u_j$ for some other $\ell \in [k] \setminus \{i, j\}$.*
- $\frac{1}{2}\phi(u_i) + \phi(u_j)$, *if $u_i \leq u_j < u_\ell$ for some other $\ell \in [k] \setminus \{i, j\}$.*

▶ **Lemma 20** (Lemma 5.3 in [18]). *For an edge of type (i, j) located at $(u_1, u_2, \dots, u_{k+1})$, where $i \neq j$ are indices in $[k+1]$, the cut density under the Descending Thresholds Rounding Scheme is at most*

- $(1 - \int_{u_i}^{u_j} \psi(u)du)\psi(u_i) + \psi(u_j)$, *if $u_\ell \leq u_i \leq u_j$ for all other $\ell \in [k] \setminus \{i, j\}$.*
- $(1 - \int_{u_i}^{u_j} \psi(u)du)((1 - \int_{u_i}^{u_\ell} \psi(u)du))\psi(u_i) + \psi(u_j)$, *if $u_i < u_\ell \leq u_j$ for some other $\ell \in [k] \setminus \{i, j\}$.*
- $(1 - \int_{u_i}^{u_j} \psi(u)du)(1 - \int_{u_i}^{u_\ell} \psi(u)du)\psi(u_i) + (1 - \int_{u_j}^{u_\ell} \psi(u)du)\psi(u_j)$, *if $u_i \leq u_j < u_\ell$ for some other $\ell \in [k] \setminus \{i, j\}$.*

The proof for each of these Lemmas is exactly as in [18], save for one additional trivial observation: the cut density of an edge of type $(i, k+1)$ is at most that of an edge of type (i, j) for any $j \neq i, j \neq k+1$. This is because the $(k+1)$st terminal is considered last, and has no threshold of its own, and therefore cannot increase the separation probability. With these Lemmas in hand, Theorem 5.6 of [18] shows, with a specific choice of parameters, that Algorithm 7 is a 1.2965-approximation to Lifted Cut as well.

Capturing the Shape of a Point Set with a Line Segment

Nathan van Beusekom
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Marc van Kreveld
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Max van Mulken
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Marcel Roeloffzen
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Bettina Speckmann
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Jules Wulms
Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands

Abstract

Detecting location-correlated groups in point sets is an important task in a wide variety of applications areas. In addition to merely detecting such groups, the group's shape carries meaning as well. In this paper, we represent a group's shape using a simple geometric object, a line segment. Specifically, given a radius r, we say a line segment is *representative* of a point set P of n points if it is within distance r of each point $p \in P$. We aim to find the shortest such line segment. This problem is equivalent to stabbing a set of circles of radius r using the shortest line segment. We describe an algorithm to find the shortest representative segment in $O(n \log h + h \log^3 h)$ time, where h is the size of the convex hull of P. Additionally, we show how to maintain a stable approximation of the shortest representative segment when the points in P move.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Shape descriptor, Stabbing, Rotating calipers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.26

Related Version *Full Version*: https://arxiv.org/abs/2402.12285

1 Introduction

Studying location-correlated groups or clusters in point sets is of interest in a wide range of research areas. There are many algorithms and approaches to find such groups; examples include the well-known *k-means clustering* [23] or *DBSCAN* [18]. In addition to the mere existence of such groups, the group's characteristics can carry important information as well. In wildlife ecology, for example, the perceived shape of herds of prey animals contains information about the behavioral state of animals within the herd [30]. Since shape is an abstract concept that can get arbitrarily complex, it is often useful to have a simplified representation of group shape that can efficiently be computed. The simplest shape (besides a point) that may represent a group is a line segment, suggesting that the group is stretched in a single direction.

When the points move in the plane, as is the case for animals, the representing line segment may change orientation and length. Also, it may disappear if the shape of the points is no longer captured well by a line segment. Conversely, it can also appear when the points form a segment-like shape again.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 26; pp. 26:1–26:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let us concentrate on the static version of the problem first. There are a few simple ways to define a line segment for a set of points. We can use the width of the point set to define a narrowest strip, put tight semi-circular caps on it, and use the centers of these semi-circles as the endpoints of the line segment. We can also use the focal points of the smallest enclosing ellipse, and use them as the endpoints. We can also use a maximum allowed distance from the points to the line segment, and use the shortest line segment possible. The first and third option are based on a hippodrome shape (the Minkowski sum of a line segment and a disk). We note that the second and third option still need a threshold distance to rule out that points are arbitrarily far from the defining line segment. In particular, the case that a line segment is *not* a suitable representation should exist in the model, and also the case where the line segment can become a single point. There are multiple other options besides the three given, for example, by using the first eigenvector of the points, or the diameter.

In this paper we study the model given by the third option: given a set P of n points in general position, we want to find the shortest line segment q_1q_2 such that all points are within distance r. This model has several advantages: (i) It is a simple model. (ii) It naturally includes the case that no line segment represents the points, or a single point already represents the points. (iii) It guarantees that all points are close to the approximating line segment. (iv) It has desirable properties when the points move: in the first two options, there are cases where the points intuitively remain equally stretched in the direction of the line segment, but points moving orthogonally away from it yields a shorter(!) line segment. This issue does not occur in the chosen model. Moreover, it was studied before in computational geometry, and we can build on existing algorithmic methods and properties.

The first algorithm published that solves the optimization version of the static problem (in fact, the first option) uses $O(n^4 \log n)$ time, for a set of n points [24]. This was improved by Agarwal et al. [1] to $O(n^2\alpha(n)\log^3 n)$, where $\alpha(n)$ is the extremely slowly growing functional inverse of Ackermann's function. The first subquadratic bound was given by Efrat and Sharir [17], who presented an $O(n^{1+\varepsilon})$ time algorithm, for any constant $\varepsilon > 0$. They use the fixed-radius version as a subroutine and then apply parametric search. Their fixed-radius algorithm already has the bound of $O(n^{1+\varepsilon})$, as it uses vertical decompositions of a parameter space in combination with epsilon-nets. They remark that their methods can solve the shortest stabber problem for unit disks within the same time, which is our problem.

In this paper we present an improved static result and new kinetic results. We solve the static version in $O(n \log^3 n)$ time by exploiting the geometry of the situation better, which allows us to avoid the use of parametric search and epsilon-nets. Our new algorithm uses a rotating calipers approach where we predict and handle events using relatively simple data structures. We still use a key combinatorial result from [17] in our efficiency analysis. For the kinetic problem, we are interested in developing a strategy to maintain a "stable" line segment that does not frequently appear and disappear, and whose endpoints move with bounded speed. To accomplish this, we must relax (approximate) the radius around the line segment in which points can be. We show that with constant speeds and a constant factor approximation in radius, the endpoints of the line segment move at a speed bounded by a linear function in r, while also avoiding frequent (dis)appearances of the line segment. These results complement recent results on stability.

Related work. A number of shape descriptors have been proposed over the years. A few popular ones are the *alpha shape* of a point set [15] and the *characteristic shape* [12], both of which generate representative polygons. Another way to generate the shape of a point set is to fit a function to the point set [6, 22, 32]. Bounding boxes and strips are much closer related

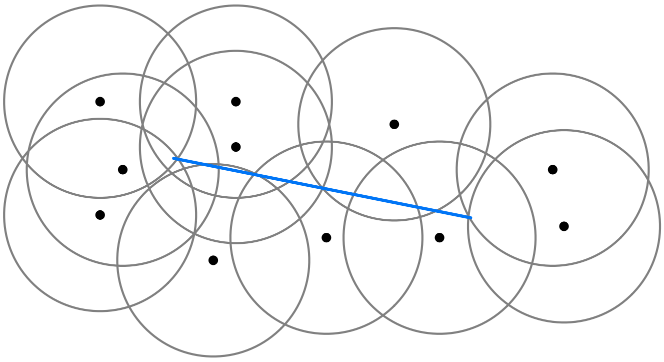


Figure 1 The line segment (blue) must hit every circle of radius r, centered at the points in P.

to the line segment we propose. In the orientation of the first eigenvector, bounding boxes (and strips) have been shown to not capture the dimensions of a point set well [11]. Optimal bounding boxes and strips, of minimum area and width, respectively, align with a convex hull edge and can be computed in linear time given the convex hull [19, 31]. Problems of finding one or more geometric objects that intersect a different set of geometric objects are known as *stabbing* problems [16], and several variants have been studied [5, 10, 29]. As mentioned, stabbing a set of unit circles with the shortest line segment was studied in [17]. The inverse variant, line segments stabbed by one or more circles, has also been studied [7, 25].

Recently, considerable attention has been given to stability of structures under the movement of a set of points or motion of other objects. Stability is a natural concern in, for example, (geo)visualization and automated cartography: In air traffic control planes may be visualized as labeled points on a map, and the labels are expected to smoothly follow the locations of the moving points [8]. Similarly, for interactive maps that allow, for example zooming and panning, labels should not flicker in and out of view [2, 20, 21, 28]. In computational geometry, only the stability of k-center problems was studied [3, 9, 13, 14], until Meulemans et al. introduced a framework for stability analysis [26]. Applying the framework to shape descriptors, they proved that an $O(1)$-approximation of an optimal oriented bounding box or strip moves only a constant-factor faster than the input points [27].

2 Computing the Shortest Representative Segment

Given a set P of n points and a distance bound r, we show how to construct the shortest segment q_1q_2 with maximum distance r to P. See Figure 1 for an example. Omitted proofs can be found in the full version.

Our algorithm uses the rotating calipers approach [31]. We start by finding the shortest representative segment for fixed orientation α, after which we rotate by π while maintaining the line segment, and return the shortest one we encounter. Note that, even though a representative segment does not exist for every orientation, we can easily find an initial orientation α for which it does exist using rotating calipers; these are the orientations at which the rotating calipers have width $\leq 2r$. Although our input point set P can be of any shape, the following lemma shows that it suffices to consider only its convex hull $\mathsf{CH}(P)$.

▶ **Lemma 1.** *If a line segment q_1q_2 intersects all circles defined by the points in the convex hull $\mathsf{CH}(P)$, then q_1q_2 also intersects all circles defined by the points in P.*

We can compute $\mathsf{CH}(P)$ in $O(n \log h)$ time, where h is the size of the convex hull [4].

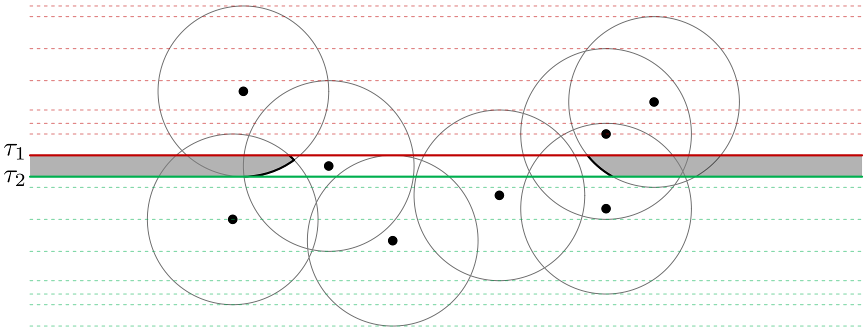


Figure 2 Two extremal tangents τ_1 and τ_2 for horizontal orientation α. The shortest line segment of orientation α that intersects all circles, ends at the boundary of the gray regions.

2.1 Fixed orientation

We describe how to find the shortest representative segment with fixed orientation α. Using rotating calipers [31], we can find all orientations in which a representative segment exists, and pick α such that a solution exists. For ease of exposition and without loss of generality, we assume α to be horizontal. Let the *left* and *right half-circle* of a circle C be the half-circle between $\pi/2$ and $3\pi/2$ and between $3\pi/2$ and $5\pi/2$, respectively. Lemma 1 permits us to consider only points of P on the convex hull, thus for the remainder of this paper we use $\mathcal{C}_P$ to indicate the set of circles of radius r centered at the points of P in $\mathsf{CH}(P)$.

Observe that every horizontal line that lies below the bottom-most top horizontal tangent τ_1 and above the top-most bottom horizontal tangent τ_2 of all circles crosses all circles (see Figure 2). If τ_1 lies below τ_2, then there exists no horizontal line that crosses all circles.

To place q_1q_2 in the strip between τ_1 and τ_2, we can define regions R_1, R_2 in which endpoints q_1 and q_2 must be placed such that q_1q_2 intersects all circles (see Figure 2).

The region R_1 is defined as the set of points below or on τ_1 and above or on τ_2 and right or on the right-most envelope of all left half-circles. The region R_2 is defined analogously using the left envelope of right half-circles. We use S_1 and S_2 to denote the envelope boundary of R_1 and R_2 respectively. Note that S_1 and S_2 are convex and consist of circular arcs from the left and right half-circles respectively. If R_1 and R_2 intersect, then we can place a single point in their intersection at distance at most r from all points in P. Otherwise, note that q_1 and q_2 must be on the convex sequences S_1 and S_2, respectively; otherwise, we can move the endpoint onto the convex sequence, shortening q_1q_2 and still intersecting all circles.

We will show that we can compute S_1 and S_2 in $O(h)$ time. First, we show that the half-circles on a convex sequence appear in order of the convex hull.

▶ **Lemma 2.** *The order of the circular arcs in S_1 or S_2 matches the order of their corresponding centers in* $\mathsf{CH}(P)$.

Now we can compute the convex sequences in linear time, given the tangents τ_1 and τ_2, which can easily be found in linear time.

▶ **Lemma 3.** *Given tangents τ_1 and τ_2, and* $\mathsf{CH}(P)$, *we can construct S_1 and S_2 in $O(h)$.*

Proof. We assume that a solution exists, which can easily be checked in $O(h)$ time. We describe only the construction of S_1, as S_2 can be constructed symmetrically. Without loss of generality, assume that τ_1 denotes the start of S_1 in clockwise order. We can find the first arc on S_1 by checking all intersections between τ_1 and the relevant half-circles, and

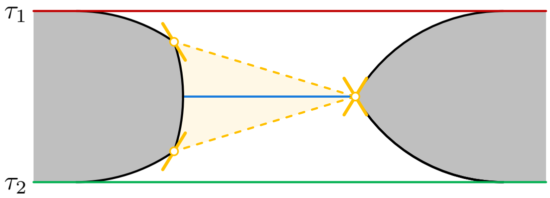


Figure 3 Two convex sequences between τ_1 and τ_2. There are multiple points on the left convex sequence that have the same tangent as the right yellow vertex. Still, there is only one line segment in horizontal orientation for which the tangents of its endpoints are equal (blue).

identifying the most extremal intersection in $O(h)$ time (see Figure 2) We add the part of the circle that lies between τ_1 and τ_2 to S_1. We then process each point p_i along the convex hull in clockwise order from the point defining our initial arc.

Next, let $\frown{c}_1, \ldots, \frown{c}_k$ denote the circular arc pieces for S_1 constructed so far. Let p_i be the next point on the convex hull that we process. Let c_i denote the left half-circle centered at p_i and let c_k be the support left half-circle of $\frown{c}_k$. We find the intersection between c_i and c_k. If there is no intersection, then c_i must lie entirely to the left of c_k and it cannot contribute to S_1. If the intersection point is below τ_2 then between τ_1 and τ_2 we have that c_i lies left of c_k and it cannot contribute to S_1. If the intersection point lies within $\frown{c}_k$ then we update S_1 to switch at the intersection point from c_k to c_i as then c_i must lie right of c_k below the intersection point. If the intersection point lies above $\frown{c}_k$ then the entirety of $\frown{c}_k$ lies to the left of c_i, therefore $\frown{c}_k$ cannot contribute to S_1 and we can discard $\frown{c}_k$. We then continue by comparing c_i to c_{k-1}.

Whenever a half-circle is possibly added it is compared to at most $O(|S_1|)$ arcs. However, when the half-circle is compared to i arcs, then $i-1$ arcs would be removed from S_1. Thus, by an amortization argument, this happens $O(h)$ times. ◀

Next, we must place q_1 and q_2 on S_1 and S_2, respectively, such that q_1q_2 is shortest. We show that q_1q_2 is the shortest line segment of orientation α when the tangents of S_1 at q_1 and S_2 at q_2 are equal. Vertices on S_1 and S_2 have a range of tangents (see Figure 3).

▶ **Lemma 4.** *Let S_1 and S_2 be two convex sequences of circular arcs, and let q_1 and q_2 be points on S_1 and S_2, respectively, such that line segment q_1q_2 has orientation α. If the tangent on S_1 at q_1 is equal to the tangent on S_2 at q_2, then q_1q_2 is minimal.*

Observe that the length of q_1q_2 is unimodal between τ_1 and τ_2. We can hence binary search in $O(\log h)$ time for the optimal placement of q_1 and q_2. By Lemmata 3 and 4 we can compute the shortest representative segment of fixed orientation α in $O(h)$ time.

2.2 Rotation

After finding the shortest line segment for a fixed orientation α, as described in the previous section, we sweep through all orientations α while maintaining τ_1, τ_2, S_1, S_2, and the shortest representative segment q_1q_2 of orientation α. We allow all of these maintained structures to change continuously as the orientation changes, and store the shortest representative segment found. Any time a discontinuous change would happen, we trigger an *event* to reflect these changes. We pre-compute and maintain a number of *certificates* in an event queue, which indicate at which orientation the next event occurs. This way we can perform the continuous motion until the first certificate is violated, recompute the maintained structures, repair the event queue, and continue rotation.

We distinguish five types of events:

1. q_1 or q_2 moves onto/off a vertex of S_1 or S_2;
2. τ_1 or τ_2 is a bi-tangent with the next circle on the convex hull;
3. τ_1 and τ_2 are the same line;
4. τ_1 or τ_2 is tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2;
5. τ_1 or τ_2 is not tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2.

Since the shortest line segment q_1q_2 in orientation α is completely determined by τ_1, τ_2, S_1, and S_2, the above list forms a complete description of all possible events. Thus, we maintain at most two certificates for events of type 1 (one for each convex sequence) and 2 (one for each tangent), and a single type-3 certificate. Additionally, there must be exactly one type-4 or type-5 certificate for each endpoint of S_1 and S_2, so four in total. These are stored in a constant-size event queue Q, ordered by appearance orientation. Insert, remove, and search operations on Q can hence be performed in $O(1)$ time.

We will describe below how all events over a full rotational sweep can be handled in $O(h \log^3 h)$ time in total. Combined with the computation of the convex hull of P this yields the following theorem. Note that in the worst case P is in convex position, and $n = h$.

▶ **Theorem 5.** *Given a point set P consisting of n points and a radius r, we can find the shortest representative segment in $O(n \log h + h \log^3 h)$ time, where $|\mathsf{CH}(P)| = h$.*

Event handling. In the following descriptions, we assume that an event happens at orientation α, and that ε is chosen such that no other events occur between $\alpha - \varepsilon$ and $\alpha + \varepsilon$. We also assume that no two events happen simultaneously, which is a general position assumption. We describe, for each event type, the time complexity of computing a new certificate of that type, the time complexity of resolving the event, and the number of occurrences.

(1) q_1/q_2 moves onto/off of a vertex of S_1/S_2. We describe, without loss of generality, how to handle the event involving q_1 and S_1; the case for q_2 and S_2 is analogous. See Figure 4 for an example of this event. First, observe that we can compute certificates of this type in $O(1)$ time, simply by walking over S_1 to find the next vertex/arc q_1 should move onto.

▶ **Observation 6.** *We can construct a new certificate of type 1 in $O(1)$ time.*

Observe that, since vertices of S_1 cover a range of tangents, there are intervals of orientations at which q_1 remains at a vertex of S_1. As such, we describe two different cases for this event: q_1 moves *onto* or *off* a vertex of S_1.

If q_1 was moving over an arc of S_1 at $\alpha - \varepsilon$ and encounters a vertex at α, then the movement path of q_1 is updated to remain on the encountered vertex. Additionally, we place a new type-1 certificate into the event queue that is violated when q_1 should move off the vertex, when the final orientation covered by the vertex is reached.

If q_1 is at a vertex at $\alpha - \varepsilon$ and orientation α is the final orientation covered by that vertex, then the movement path of q_1 must be updated to follow the next arc on S_1. Additionally, we place a new type-1 certificate into the event queue that is violated when q_1 encounters the next vertex, at the orientation at which this arc of S_1 ends.

▶ **Lemma 7.** *Throughout the full π rotation, type-1 events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(1)$ time.*

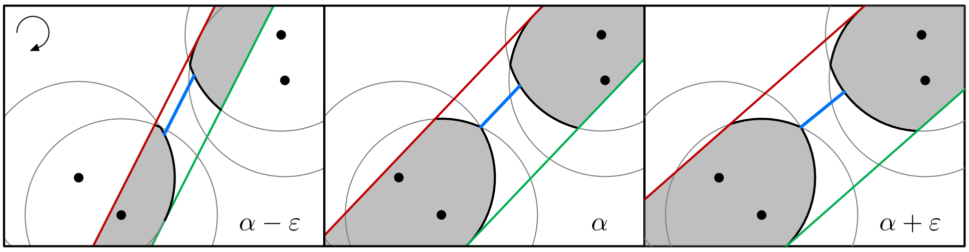


Figure 4 When q_1/q_2 is at a vertex of S_1/S_2, it stops moving.

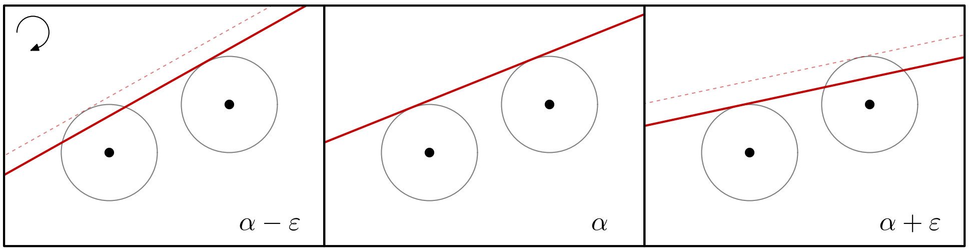


Figure 5 When the defining circle of τ_1/τ_2 changes, τ_1/τ_2 is parallel to a convex hull edge.

(2) τ_1 or τ_2 is bi-tangent with the next circle on the convex hull. We describe, without loss of generality, how to handle the event involving τ_1; handling τ_2 is analogous. See Figure 5 for an illustration. First, observe that we can compute certificates of this type in $O(1)$ time, since these certificates depend only on the orientation of the next convex hull edge.

▶ **Observation 8.** *We can construct a new certificate of type 2 in $O(1)$ time.*

When τ_1 is a bi-tangent of two circles defined by their centers $u, v \in P$ then, by definition of τ_1, u and v must both be the extremal points in the direction θ perpendicular to α. Therefore, (u, v) must be an edge on the convex hull. Suppose that, without loss of generality, u was the previous extremal vertex in direction $\theta - \varepsilon$, then v is extremal in direction $\theta + \varepsilon$. As such, τ_1 belongs to u at $\alpha - \varepsilon$, and to v at $\alpha + \varepsilon$. When this happens, we insert a new type-2 certificate into the event queue that is violated at the orientation of the next convex hull edge. Additionally, we must recompute the certificates of type 3, 4 and 5 that are currently in the event queue, since these are dependent on τ_1.

▶ **Lemma 9.** *Throughout the full π rotation, type-2 events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(\log^2 h)$ time.*

(3) τ_1 and τ_2 are the same line. When this event takes place, τ_1 and τ_2 are the inner bi-tangents of their two respective defining circles. See Figure 6 for an example. First, observe that we can compute certificates of this type in $O(1)$ time by simply finding the inner bi-tangent of the circles corresponding to τ_1 and τ_2.

▶ **Observation 10.** *We can construct a new certificate of type 3 in $O(1)$ time.*

We distinguish two different cases for this event: either there is a solution at $\alpha - \varepsilon$ and no solution at $\alpha + \varepsilon$, or vice versa.

If there was a solution at $\alpha - \varepsilon$ and there is none at $\alpha + \varepsilon$, we simply stop maintaining q_1q_2, S_1 and S_2 until there exists a solution again. As such, we remove all type-1, type-5 and type-4 certificates from the event queue and place a new type-3 certificate into the event queue that is violated at the next orientation where τ_1 and τ_2 are the same line.

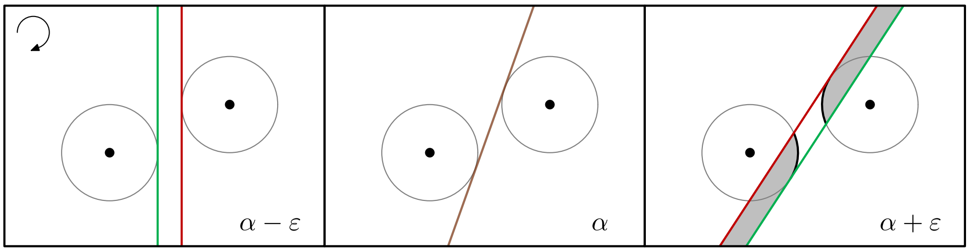


Figure 6 When τ_1 and τ_2 are the same line, they are an inner bi-tangent of their two defining circles.

If there was no solution at $\alpha - \varepsilon$ and there is a solution at $\alpha + \varepsilon$, we must recompute S_1, S_2, and $q_1 q_2$ at orientation α. At orientation α, S_1 and S_2 are single vertices where τ_1 and τ_2 intersect the extremal half-circles of the arrangement. Then, $q_1 q_2$ is the line segment between these single vertices of S_1 and S_2. We place new type-1, type-4 and type-5 certificates into the event queue reflecting the newly found S_1, S_2, q_1 and q_2. Additionally, we insert a new type-3 certificate that is violated at the next orientation where τ_1 and τ_2 are the same line.

▶ **Lemma 11.** *Throughout the full π rotation, type-3 events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(\log^2 h)$ time.*

(4) τ_1 or τ_2 is tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2. We describe, without loss of generality, how to handle the event involving τ_1 and S_1; the case for τ_2 and S_2 is analogous. See Figure 7 for an example of this event. First, observe that we can compute certificates of this type in $O(1)$ time: Let C_i be the circle to which τ_1 is tangent. Then, to construct a certificate, we find the orientation at which τ_1 hits the intersection point of C_i and S_1. Note that this intersection is part of S_1 or will appear at τ_1.

▶ **Observation 12.** *We can construct a new certificate of type 4 in $O(1)$ time.*

Let vertex v be the vertex of the convex chain S_1 that is intersected by τ_1 at orientation α. Then either vertex v is a vertex of S_1 at orientation $\alpha - \varepsilon$ but not at $\alpha + \varepsilon$, or vice versa.

In the prior case, at orientation α the arc to which τ_1 is a tangent is completely removed from S_1. Vertex v becomes the endpoint of S_1 and starts moving along the next arc of S_1. If the affected arc or vertex appeared in a type-1 certificate in the event queue, it is updated to reflect the removal of the arc and the new movement of the vertex. Additionally, we place a new type-5 certificate into the event queue.

In the latter case, at orientation α an arc of the incident circle to τ_1 needs to be added to S_1. If the arc that was previously the outer arc of S_1 appeared in a type-1 certificate in the event queue, it may need to be updated to reflect the addition of the new arc. Additionally, we place a new type-4 certificate into the event queue.

▶ **Lemma 13.** *Throughout the full π rotation, type-4 events happen at most $O(h)$ times, and we can resolve each occurrence of such an event in $O(\log^2 h)$ time.*

(5) τ_1 or τ_2 is not tangent to S_1 or S_2 and rotates over a (prospective) vertex of S_1 or S_2. We describe, without loss of generality, how to handle the event involving τ_1 and S_1; the case for τ_2 and S_2 is analogous. See Figure 8 for an example of this event. The time complexity of constructing a certificate of this type is stated in the following lemma.

▶ **Lemma 14.** *We can construct a new certificate of type 5 in $O(\log^2 h)$ time.*

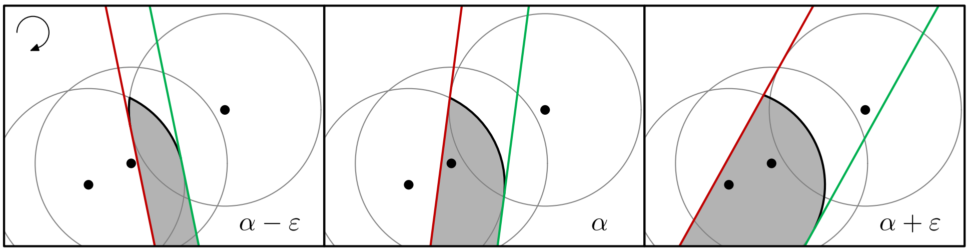


Figure 7 When τ_1/τ_2 hits an intersection of its defining circle that is also on S_1/S_2, an arc is removed from S_1/S_2.

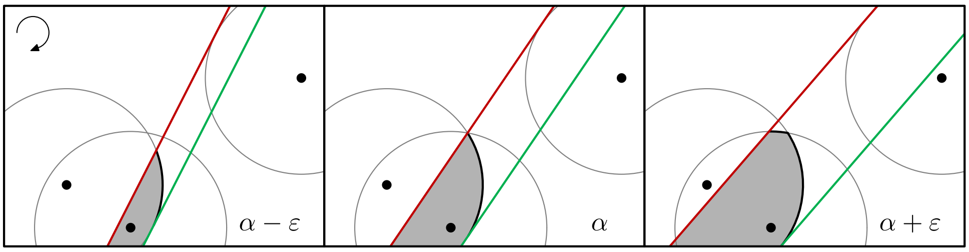


Figure 8 When τ_1/τ_2 hits an intersection of two circles, an arc needs to be added to S_1/S_2.

Additionally, we get the following bounds on handling type-5 events.

▶ **Lemma 15.** *Throughout the full π rotation, τ_1 or τ_2 hits a vertex of S_1 or S_2 at most $O(h \log h)$ times, and we handle each occurrence of this event in $O(\log^2 h)$ time.*

We prove Lemmata 14 and 15 using an additional data structure in the following section.

2.3 Finding and maintaining the convex sequence

In this section, we describe an additional data structure necessary to maintain the convex sequences S_1 and S_2 efficiently. We can use this data structure to construct and handle violations of type-5 certificates efficiently, as well as to find new starting positions of S_1 and S_2 after a type-3 event.

Let $p_1, \ldots, p_h$ be the vertices of the convex hull in clockwise order. At a given orientation α, let p_i be the point corresponding to the circle C_i to which τ_1 is tangent. We use v_τ to denote the intersection point between τ_1 and S_1, if it exists, which is simultaneously an endpoint of S_1. Let p_j be the point corresponding to the circle C_j on which v_τ is located. This implies that the arc on S_1 intersected by τ_1 belongs to circle C_j. Then, during our rotational sweep, v_τ is moving over C_j. A type-5 event takes place when v_τ hits the intersection of C_j with another circle C_k corresponding to point p_k.

If, before a type-5 event, the arc of C_j on S_1 was shrinking due to the movement of v_τ, then C_j is fully removed from S_1 at the event, and v_τ continues moving over S_1. Constructing the certificate in this case is very easy, since all we need to do is walk over S_1 from v_τ to find the next vertex. As such, for the remainder of this section, we consider only the more complicated type-5 event, where the arc of C_j on S_1 is growing due to the movement of v_τ.

In that case, when the type-5 event happens, an arc of C_k is added to S_1, and v_τ starts moving over C_k instead of C_j. As such, to construct a type-5 certificate, we must find the intersection between C_j and another circle C_k belonging to a point $p_k \in P$, such that the intersection between C_j and C_k is the first intersection hit by v_τ. To do this, we will first state some characteristics of C_k and p_k.

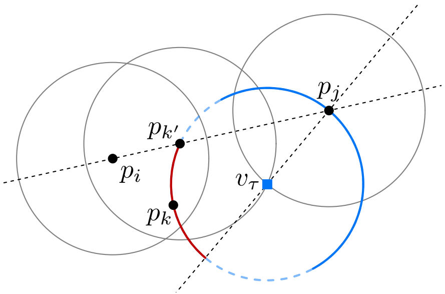


Figure 9 Point p_k is placed on the red arc, which is a subset of the (blue dashed) convex semi-circle centered at v_τ, and disjoint from the (blue solid) concave semi-circle centered at v_τ.

First, observe that finding the first intersecting circle C_k of C_j is not necessarily enough. We are only interested in the semi-circles of all circles in $\mathcal{C}_P$ that have the same "opening direction" as the convex chain S_1 for a given orientation α. As such, let the *convex semi-circle* of a given circle C be the semi-circle of C that is convex with respect to S_1. Conversely, let the *concave semi-circle* of C be the opposite semi-circle of C. Then, circle C_k appears on S_1 after a hit by v_τ at orientation α if v_τ is on the convex semi-circle of C_k at orientation α. If this is not the case, C_k should be skipped. We show that, for this reason, we never have to consider points p_l such that $l < i$ or $j < l$ when constructing a type-5 certificate.

▶ **Lemma 16.** *Let C_i be the circle defining τ_1, and let C_j be the circle on which v_τ is located, for $i \neq j$. Let C_k be the first circle hit by v_τ during rotation at orientation α. If $k < i$ or $j < k$, then at orientation α, v_τ lies on the concave semi-circle of C_k with orientation α.*

Proof. Without loss of generality assume p_i is positioned left of p_j, then assuming that $k < i$ or $j < k$, p_k must lie below the line through p_i and p_j (since $i < j$ and the points are ordered in clockwise order). See Figure 9 for the following construction.

Consider point $p_{k'}$ placed on the line through p_i and p_j, such that v_τ could be the bottom intersection of C_j and a radius-r circle centered at $p_{k'}$. Let p_k be a point placed on the circle of radius r centered at v_τ. If p_k is placed on the other side of the line through v_τ and p_j, compared to $p_{k'}$, then v_τ would enter C_k at orientation α, which does not induce an event. As such, p_k must be on the same side of the line through v_τ and p_j as $p_{k'}$. Additionally, since $k < i$ or $j < k$, p_k must be below the line through p_i and p_j.

It is easy to see that all points on the convex semi-circle of v_τ will have v_τ on their concave semi-circle. Since the arc on which p_k is placed is a strict subset of the concave semi-circle of v_τ, any placement of p_k must have v_τ on its concave semi-circle. ◀

Lemma 16 implies that, while constructing a type-5 certificate, we need to consider only candidate points p_k such that $i < k < j$. Note that, if $i = j$, we get a type-4 event. Then, all that is left is to find the first circle C_k with $i < k < j$ that is intersected by v_τ as it moves over C_j. To this end, we describe a data structure that allows us to perform a circular ray shooting query along C_j from the orientation at which the certificate must be constructed.

Data structure. Our data structure is essentially a balanced binary tree $\mathcal{T}$ on the vertices of the convex hull in clockwise order, where each node stores an associated structure (see Figure 10). For any i, let D_i be the disk bounded by circle C_i. Suppose a node in $\mathcal{T}$ is the root of a subtree with $p_i, \ldots, p_j$ in the leaves. Then its associated structure stores the boundary of $\bigcap_{i \le l \le j} D_l$ as a sorted sequence of circular arcs. Given a range (i, j) we can

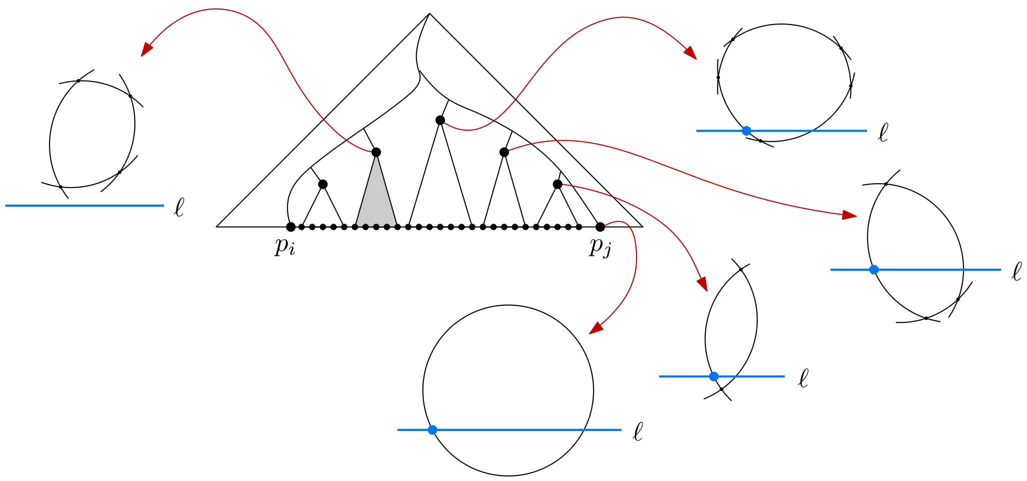


Figure 10 A schematic representation of the data structure $\mathcal{T}$ and a query with ray ℓ.

query this data structure with a (circular) ray in $O(\log^2 h)$ time to find the first intersection of the ray with the boundary of $\bigcap_{i \leq l \leq j} D_l$. A more detailed description of the data structure can be found in the full version of this paper, along with the query algorithms needed to handle certain events and find new certificates.

Event handling. Whenever a certificate of type 5 is violated at orientation α, it means some circle C_k is hit by v_τ at orientation α. It is still possible, however, that this hit happens on the concave semi-circle of C_k. In that case, C_k should not be added to S_1, and v_τ should simply continue moving over its original trajectory. We call these events, where a type-5 certificate is violated but S_1 is not updated, *internal events.* To handle an internal event, we merely need to construct another new type-5 certificate and continue our rotational sweep. In this case, however, we do not have to search the entire range $p_i, \ldots, p_j$ when constructing a new certificate, as shown in the following lemma.

▶ **Lemma 17.** *Let p_i be the point corresponding to τ_1, and let v_τ be at the intersection of circles C_j and C_k, where C_j is the circle that defines the current trajectory of v_τ and where v_τ is on the concave semi-circle of C_k. Then if the next circle hit by v_τ is C_l for $i < l < k$, this circle is hit on its concave semi-circle.*

Proof. Let C_l be the next circle hit by v_τ for $i < l < k$, and see Figure 11 for the following construction. Since v_τ is on the concave semi-circle of p_k, p_k must be on the convex semi-circle of v_τ. Furthermore, as C_k was hit by v_τ, p_k must lie on the same side of the line through p_j and v_τ as p_i. Let the endpoint of the concave semi-circle of v_τ that lies clockwise from p_k be denoted v_c, and observe that $d(p_k, p_j) > d(v_c, p_j)$, where $d(a, b)$ denotes the Euclidean distance between a and b. Additionally, since we consider only points on the convex hull, point p_l must lie above line $p_i p_k$ but below line $p_k p_j$, resulting in a cone with its apex at p_k.

Every point in this cone is further away from p_j than p_k: Since v_τ is part of S_1, placing q_1 at v_τ must yield a valid solution for $q_1 q_2$. This means that all points must be in the Minkowski sum of a radius r disc and the ray with orientation α originating from v_τ, and hence p_k lies below the line $v_c p_i$ (see Figure 12). Finally, p_j lies on the concave semi-circle around v_τ, and p_k lies on the convex semi-circle around v_τ, which shows that the cone lies on the far side of p_k with respect to p_j. This implies that $d(p_l, p_j) > d(p_k, p_j)$.

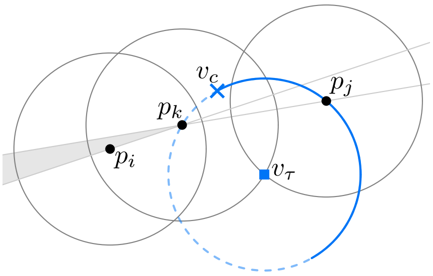


Figure 11 p_l must be located in the gray cone.

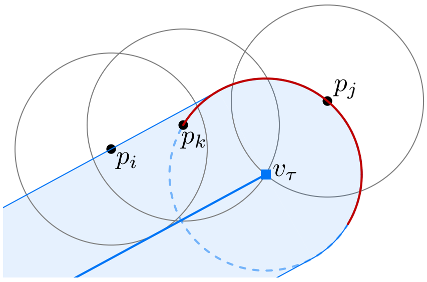


Figure 12 All points in $\mathsf{CH}(P)$ must be in the blue shaded area. When an internal event happens, the red semi-circle is a witness that (p_k, p_j) is conjugate.

During our monotone rotational sweep, v_c must continuously move closer to p_j as we continue our sweep. Therefore, when v_τ intersects C_l, we must have $d(p_l, p_j) > d(v_c, p_j)$: This directly implies that v_c must lie clockwise from p_l on the radius r circle centered at v_τ. Thus, v_τ lies on the concave semi-circle of C_l. ◀

Lemma 17 implies that, after an internal event with circle C_l, it is sufficient to consider only points p_k with $l < k < j$ when constructing a new type-5 certificate.

When a type-5 certificate is violated and v_τ is on the convex semi-circle of C_k, however, we do need to update S_1 to reflect v_τ moving over the intersection between C_k and C_j. Additionally, we must construct new certificates: We possibly need to compute a new type-1 certificate, as well as either a type-4 certificate or a new type-5 certificate using C_k as the new trajectory of v_τ and searching for the next hit with C_l for $i < l < k$. We are now ready to prove Lemmata 14 and 15.

▶ **Lemma 14.** *We can construct a new certificate of type 5 in $O(\log^2 h)$ time.*

Proof. Let p_i be the point corresponding to τ_1, C_j be the circle over which v_τ is currently moving, and α be the current orientation. To construct a type-5 certificate, we find the first circle C_k with $i \leq l < k < j$ that is intersected by v_τ. Here, if this certificate is constructed during an internal event, l is the index of the circle C_l intersected by v_τ during that event. Otherwise, $l = i$. Since v_τ moves over C_j, we can use the data structure described in the full version of this paper to perform a circular ray shooting query on the sequence $C_l, \ldots, C_{j-1}$ with starting point v_τ to find C_k in $O(\log^2 h)$. This gives us the point p_k to include in the certificate, and we can find the orientation at which p_k is hit by drawing the tangent of C_i through the intersection point between C_j and C_k. ◀

▶ **Lemma 15.** *Throughout the full π rotation, τ_1 or τ_2 hits a vertex of S_1 or S_2 at most $O(h \log h)$ times, and we handle each occurrence of this event in $O(\log^2 h)$ time.*

Proof. To show that this event happens $O(h \log h)$ times during a π rotation, we need the following definition. Let an ordered pair (p_g, p_h) of vertices of $\mathsf{CH}(P)$ be *conjugate* if we can place a semi-circle of radius r so that it hits p_g and p_h, p_h lies clockwise from p_g on this semi-circle, and the semi-circle does not intersect the interior of $\mathsf{CH}(P)$. Efrat and Sharir prove that any convex polygon with n vertices has at most $O(n \log n)$ conjugate pairs if the vertices are placed in general position [17]. We charge each occurrence of a type-5 event to a conjugate pair, and prove that each pair is charged only a constant number of times. This immediately yields the bound of $O(h \log h)$ on the number of type-5 events.

Consider an internal type-5 event. We charge these events to the pair (p_k, p_j). To this end, we show that this pair of points must be conjugate. Consider orientation α at which this event takes place. At that point, we can draw a circle of radius r, centered at v_τ, through p_k and p_j. Since, by definition of an internal event, v_τ is on the concave semi-circle of p_k, p_k must be on the convex semi-circle centered around v_τ.

Now again observe that, since v_τ is part of S_1, placing q_1 at v_τ must yield a valid solution for q_1q_2. This means that all points must be in the Minkowski sum of a radius r disc and the ray with orientation α originating from v_τ. See Figure 12. Therefore, the concave semi-circle of radius r centered at v_τ does not intersect $\mathsf{CH}(P)$. If we rotate this semi-circle counter-clockwise until one of its endpoints coincides with p_k, we obtain a semi-circle through p_k and p_j that does not intersect $\mathsf{CH}(P)$. This means it is a witness that (p_k, p_j) is conjugate.

Next, consider a type-5 event that is not internal. The same argument as above holds, except p_k is already on the concave semi-circle of radius r centered at v_τ. Since this semi-circle does not intersect $\mathsf{CH}(P)$, that semi-circle is a direct witness that (p_k, p_j) is conjugate.

Every conjugate pair only induces at most one type-5 event. In order for conjugate pair (p_k, p_j) to induce a second type-5 event, v_τ must again hit the intersection between p_k and p_j while it is moving in the same angular movement direction. This can only happen if we perform a full 2π rotational sweep, or if v_τ first moves over this intersection in the opposite direction. The prior is not possible in a π rotational sweep. The latter is only possible if p_k is first involved in a type-4 event. But then, $k = i$, and by construction p_k can never be involved in another type-5 certificate.

Handling a type-5 event consists of updating S_1 and the movement trajectory of v_τ, which can be done in $O(1)$ time. Additionally, we construct new type-1 certificate and either a type-4 or type-5 certificate, which can be done in $O(1)$ and $O(\log^2 h)$ time by Observations 6 and 12 and Lemma 14. ◀

After analyzing all events that occur during the rotational sweep, we can prove Theorem 5.

▶ **Theorem 5.** *Given a point set P consisting of n points and a radius r, we can find the shortest representative segment in $O(n \log h + h \log^3 h)$ time, where $|\mathsf{CH}(P)| = h$.*

Proof. We initialize the algorithm by computing the convex hull $\mathsf{CH}(P)$. By Lemma 1, it is sufficient to consider only points in $\mathsf{CH}(P)$ to find a representative segment of P. We use rotating calipers to check that the shortest representative segment is not a point, and find an orientation α in which a solution exists. For this fixed α we find τ_1 and τ_2, compute S_1 and S_2, as well as the shortest line segment q_1q_2 in orientation α. Computing the convex hull can be done in $O(n \log h)$ [4]. By Lemma 3, S_1 and S_2 can be initialized in $O(h)$, and we can initialize q_1q_2 in $O(\log h)$ time. As such, initialization of the algorithm can be done in $O(n \log h)$ time in total.

Next, we rotate orientation α over π in total, maintaining τ_1, τ_2, S_1, and S_2, as well as q_1q_2. Note that a rotation of π is sufficient, since we consider orientations, which identify opposite directions of a 2π rotation. Throughout the rotation we maintain the shortest

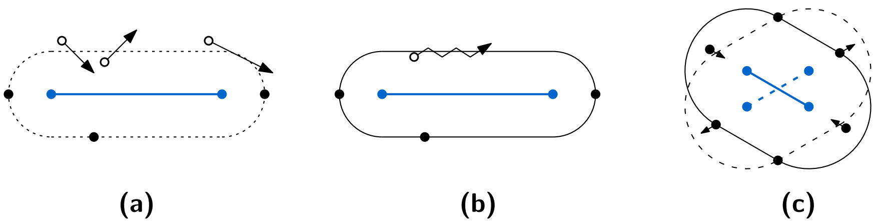


Figure 13 Examples of representative segments (blue) for moving points. These show that a representative segment may appear and disappear frequently **(a)** due to the motion of two points (top left) or one point (top right) or **(b)** due to minor oscillations in a movement path. **(c)** Small movements can trigger a discrete change of the representative segment.

representative segment, and return the shortest such line segment found over all orientations. Over the full rotation, we encounter five different types of events. By Lemmata 7–15, these events can be handled in $O(h \log^3 h)$ time in total.

As we mentioned in Section 2.1, the shortest representative segment is defined by only τ_1, τ_2, S_1, and S_2. Each tangent is defined by a circle. Hence, τ_1 and τ_2 can only change when they are defined by a new circle. Thus, by Lemma 9, we correctly maintain τ_1 and τ_2 throughout the full π rotation. Furthermore, S_1 and S_2 can only exist when τ_1 and τ_2 appear in the correct order. By Lemma 11 we correctly maintain when S_1 and S_2 exist. Finally, S_1 and S_2 can only make a discrete change as the tangents τ_1 and τ_2 hit intersections between circles in $\mathcal{C}_P$. If the tangents do not touch a vertex, then S_1 and S_2 must change continuously along the arcs that τ_1 and τ_2 cross. By Lemmata 13 and 15, we correctly maintain S_1 and S_2 when they exist. In conclusion, we correctly maintain τ_1, τ_2, S_1, and S_2 throughout the full π rotation. Since we maintain the shortest line segment between S_1, and S_2 in any orientation using Lemma 7, we also maintain the shortest representative segment for any orientation. ◀

3 Stable Representative Segments for Moving Points

In this section, we consider maintaining representative segment q_1q_2 while the points in P move. We first show what can happen if we would maintain the optimal solution explicitly under continuous motion of the points.

There are examples where a representative segment exists for a value of r for an arbitrarily short duration. This can happen because one point moves towards a hippodrome shape and another point moves out of it, giving a brief moment with a valid hippodrome. The same effect can be caused by a single linearly moving point that grazes the hippodrome at a join point. These examples are illustrated in Figure 13(a). It can also happen that a minor oscillating movement of a point causes a quick sequence of changes between a valid and no valid segment, see Figure 13(b).

Now, consider the point set P in which the points form a regular k-gon (see Figure 13(c)). When r is equal to half the width of the k-gon, then we can force a discrete change in the placement of q_1 and q_2, with very slow movement of points in P. In this case there is always a valid representative segment, but its endpoints make a jump.

We see that continuously maintaining the optimal solution has two artifacts that are undesirable to a human observer: (1) the segment can appear and disappear frequently within an arbitrarily short time frame, and (2) a segment may jump to a new location, leading to infinitely high speeds of the endpoints even if the points themselves move slowly.

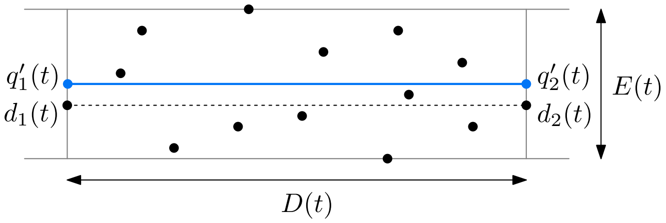


Figure 14 The prospective segment $q'_1q'_2$ in relation to D, E, and diametrical line d_1d_2 at time t.

Our goal is therefore to ensure that the segment movement is *stable* over time: We want q_1 and q_2 to move continuously and with bounded speed, preventing discrete or (near) instantaneous changes. We accomplish this as follows. First, we will sample the positions of the moving points only at integer moments, and then decide immediately to show or not show a solution in the next time unit. This implies that existence and non-existence of a solution lasts at least one time unit, and we avoid the solution (dis)appearing frequently. Second, we make the assumption that the maximum speed of the points is unit. We have to make some bounded speed assumption otherwise we cannot hope to get a bounded speed of the endpoints either. Third, we allow more flexibility in when we have a solution. We do this using a less strict regime on r, and by assuming that $r \geq 1$. Whenever the real solution exists (with the actual r), then we guarantee that we also have a solution. Whenever there is no solution even for a radius of $2\sqrt{2} \cdot r + 4$, then we never give a solution. When the radius is in between these bounds, we may have a solution or not. Our algorithm can then ensure that the speeds of the endpoints are bounded, and the length of our chosen segment always approximates the true optimum (when it exists), at any moment in time, also between the integer sampling moments.

So we assume that a point $p \in P$ is described by a trajectory that is sampled at integer timestamps. That is, each point in P is described by $p(i) \to \mathbb{R}^2$ where $i \in \mathbb{Z}$ is the timestamp. We also assign each endpoint of the representative segment a position as a function of $t \in \mathbb{R}$, which means that the representative segment at time t is now defined by $q_1(t)q_2(t)$. Furthermore, we use $D(t)$ and $W(t)$ as the *diameter* and *width* of the point set, which are respectively the maximum pairwise distance of points in P and the width of the thinnest strip containing P. Let $d_1(t), d_2(t) \in P$ be a pair of points defining the diameter. Lastly, we also use the *extent* $E(t)$ of P in the direction orthogonal to the line segment $d_1(t)d_2(t)$. For all of the above definitions, we omit the dependence on t when it is clear from the context.

In the following, we describe how to specify $q_1(t)$ and $q_2(t)$ such that they move with bounded speed, and such that the length of the segment $q_1(t)q_2(t)$ as well as the proximity of the segment to P can be bounded at any time t. In particular, we define such a segment $q_1(t)q_2(t)$ as an *approximating* segment, and prove that the length of an optimal representative segment is approximated by an additive term l, and at the same time the maximum distance from any point in P to the segment $q_1(t)q_2(t)$ is at most $h \cdot r$, for some constant h.

Algorithm. Our algorithm $A(t)$ is *state-aware*. This means that the output of $A(t)$ is dependent only on the input at or before time t, but it has no knowledge of the input after time t. At every integer timestamp $i \in \mathbb{Z}$, we compute a *canonical solution* $q'_1(i)q'_2(i)$. The endpoints $q'_1(i)$ and $q'_2(i)$ of this canonical solution are placed on the lines orthogonal to the diametric line through $d_1(i)$ and $d_2(i)$, respectively, such that $q'_1(i)q'_2(i)$ lies in the middle of the narrowest strip containing P in the diametric orientation, see Figure 14.

Our algorithm is now a special kind of state-aware algorithm called a *chasing algorithm* [27]: At at every integer time step $i \in \mathbb{Z}$, the algorithm computes the canonical solution $q'_1(i)q'_2(i)$ and then linearly interpolates from $q'_1(i-1)q'_2(i-1)$ to $q'_1(i)q'_2(i)$, arriving there at time $i+1$. At that point, $q'_1(i+1)q'_2(i+1)$ can be computed, and we continue in a similar manner.

However, if the maximum distance from P to the canonical solution becomes too large, we no longer want the algorithm to output any solution. In this case, no (optimal) representative solution exists, and we do not produce an approximating segment either.

Formally, for any timestamp $t \in (i, i+1)$, we linearly interpolate q_1 and q_2 between their previous canonical placements as follows. We define $\alpha = (t-i)$ and set $q_j(t) = \alpha \cdot q'_j(i-1) + (1-\alpha) \cdot q'_j(i)$, for $j \in \{1,2\}$. Then, the output of our algorithm is $A(t) = q_1(t)q_2(t)$ if $E(\lfloor t \rfloor) \leq 2r\sqrt{2}+2$ and $\varnothing$ otherwise.

The above algorithm yields the bounds stated in the following theorem. Detailed proofs for these bounds can be found in the full version of this paper.

▶ **Theorem 18.** *Given a set P of points moving with at most unit speed, algorithm A yields a stable approximating segment with $l = 2r + 4$ and $h = 2\sqrt{2} + 4$, for which speed of the endpoints is bounded by $(2r+1)\sqrt{2} + 2$.*

4 Conclusion

In this paper, we presented an $O(n \log h + h \log^3 h)$ time algorithm to find the shortest representative segment of a point set, improving the previous $O(n^{1+\varepsilon})$ time solution. Additionally, we showed how to maintain an approximation of the shortest representative segment in a stable manner, such that its endpoints move with a speed bounded by a linear function in r.

There may be possibilities for improving the running time of our static solution to $O(n \log h + h \log^2 h)$, or even $O(n \log h)$. The $O(h \log^3 h)$ term comes from having to handle $O(h \log h)$ type-5 events in $O(\log^2 h)$ time each. However, it may be possible to show that there are at most $O(h)$ type-5 events, since the conjugate pairs used to bound the number of internal events each have a unique starting point. Additionally, it may be possible to improve the query time of the data structure described in the full version of this paper to $O(\log h)$ time using ideas like fractional cascading, but there is no straightforward way to make this work for the circular query.

References

1 Pankaj K. Agarwal, Alon Efrat, Micha Sharir, and Sivan Toledo. Computing a segment center for a planar point set. *Journal of Algorithms*, 15(2):314–323, 1993. doi:10.1006/jagm.1993.1043.

2 Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry: Theory and Applications*, 43(3):312–328, 2010. doi:10.1016/j.comgeo.2009.03.006.

3 Sergei Bespamyatnikh, Binay Bhattacharya, David Kirkpatrick, and Michael Segal. Mobile facility location. In *Proc. 4th Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, pages 46–53, 2000. doi:10.1145/345848.345858.

4 Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996. doi:10.1007/BF02712873.

5 Timothy M. Chan, Thomas C. van Dijk, Krzysztof Fleszar, Joachim Spoerhase, and Alexander Wolff. Stabbing rectangles by line segments–how decomposition reduces the shallow-cell complexity. In *Proc. 29th International Symposium on Algorithms and Computation (ISAAC)*, pages 61:1–61:13, 2018. doi:10.4230/LIPICS.ISAAC.2018.61.

6 Danny Z. Chen and Haitao Wang. Approximating points by a piecewise linear function. *Algorithmica*, 66(3):682–713, 2013. doi:10.1007/s00453-012-9658-y.

7 Merce Claverol, Elena Khramtcova, Evanthia Papadopoulou, Maria Saumell, and Carlos Seara. Stabbing circles for sets of segments in the plane. *Algorithmica*, 80:849–884, 2018. doi:10.1007/s00453-017-0299-z.

8 Mark de Berg and Dirk H. P. Gerrits. Labeling moving points with a trade-off between label speed and label overlap. In *Proc. 21st European Symposium on Algorithms (ESA)*, pages 373–384, 2013. doi:10.1007/978-3-642-40450-4_32.

9 Mark de Berg, Marcel Roeloffzen, and Bettina Speckmann. Kinetic 2-centers in the black-box model. In *Proc. 29th Symposium on Computational Geometry (SoCG)*, pages 145–154, 2013. doi:10.1145/2462356.2462393.

10 José Miguel Díaz-Báñez, Matias Korman, Pablo Pérez-Lantero, Alexander Pilz, Carlos Seara, and Rodrigo I. Silveira. New results on stabbing segments with a polygon. *Computational Geometry*, 48(1):14–29, 2015. doi:10.1016/j.comgeo.2014.06.002.

11 Darko Dimitrov, Christian Knauer, Klaus Kriegel, and Günter Rote. Bounds on the quality of the PCA bounding boxes. *Computational Geometry*, 42(8):772–789, 2009. doi:10.1016/j.comgeo.2008.02.007.

12 Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern recognition*, 41(10):3224–3236, 2008. doi:10.1016/j.patcog.2008.03.023.

13 Stephane Durocher and David Kirkpatrick. The steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. *International Journal of Computational Geometry & Applications*, 16(04):345–371, 2006. doi:10.1142/S0218195906002075.

14 Stephane Durocher and David Kirkpatrick. Bounded-velocity approximation of mobile Euclidean 2-centres. *International Journal of Computational Geometry & Applications*, 18(03):161–183, 2008. doi:10.1142/S021819590800257X.

15 Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on information theory*, 29(4):551–559, 1983. doi:10.1109/TIT.1983.1056714.

16 Herbert Edelsbrunner, Hermann A. Maurer, Franco P. Preparata, Arnold L. Rosenberg, Emo Welzl, and Derick Wood. Stabbing line segments. *BIT Numerical Mathematics*, 22:274–281, 1982. doi:10.1007/BF01934440.

17 Alon Efrat and Micha Sharir. A near-linear algorithm for the planar segment-center problem. *Discrete & Computational Geometry*, 16(3):239–257, 1996. doi:10.1007/BF02711511.

18 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996. doi:10.5555/3001460.3001507.

19 Herbert Freeman and Ruth Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7):409–413, 1975. doi:10.1145/360881.360919.

20 Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Sliding labels for dynamic point labeling. In *Proc. 23rd Canadian Conference on Computational Geometry (CCCG)*, pages 205–210, 2011.

21 Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent labeling of rotating maps. *Journal of Computational Geometry*, 7(1):308–331, 2016. doi:10.20382/jocg.v7i1a15.

22 S. Louis Hakimi and Edward F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graphical Models and Image Processing*, 53(2):132–136, 1991. doi:10.1016/1049-9652(91)90056-P.

23 John A. Hartigan and Manchek A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society.*, 28(1):100–108, 1979. doi:10.2307/2346830.

24 Hiroshi Imai, DT Lee, and Chung-Do Yang. 1-segment center problems. *ORSA Journal on Computing*, 4(4):426–434, 1992. doi:10.1287/ijoc.4.4.426.

25 Konstantin Kobylkin. Stabbing line segments with disks: complexity and approximation algorithms. In *Proc. 6th International Conference on Analysis of Images, Social Networks and Texts (AIST)*, pages 356–367, 2017. doi:10.1007/978-3-319-73013-4_33.

26 Wouter Meulemans, Bettina Speckmann, Kevin Verbeek, and Jules Wulms. A framework for algorithm stability and its application to kinetic euclidean MSTs. In *Proc. 13th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 805–819, 2018. doi:10.1007/978-3-319-77404-6_58.

27 Wouter Meulemans, Kevin Verbeek, and Jules Wulms. Stability analysis of kinetic orientation-based shape descriptors. *arXiv preprint*, 2019. arXiv:1903.11445.

28 Martin Nöllenburg, Valentin Polishchuk, and Mikko Sysikaski. Dynamic one-sided boundary labeling. In *Proc. 18th International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 310–319, 2010. doi:10.1145/1869790.1869834.

29 Lena Marie Schlipf. *Stabbing and Covering Geometric Objects in the Plane.* PhD thesis, FU Berlin, 2014.

30 Ariana Strandburg-Peshkin, Damien R. Farine, Margaret C. Crofoot, and Iain D. Couzin. Habitat and social factors shape individual decisions and emergent group structure during baboon collective movement. *eLife*, 6:e19505, 2016. doi:10.7554/eLife.19505.

31 Godfried T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. 2nd Mediterranean Electrotechnical Conference (MELECON)*, volume 1, page A10, 1983.

32 Der Perng Wang. A new algorithm for fitting a rectilinear x-monotone curve to a set of points in the plane. *Pattern Recognition Letters*, 23(1-3):329–334, 2002. doi:10.1016/S0167-8655(01)00130-1.

Polynomial Calculus for Quantified Boolean Logic: Lower Bounds Through Circuits and Degree

Olaf Beyersdorff
Friedrich Schiller University Jena, Germany

Tim Hoffmann
Friedrich Schiller University Jena, Germany

Kaspar Kasche
Friedrich Schiller University Jena, Germany

Luc Nicolas Spachmann
Friedrich Schiller University Jena, Germany

Abstract

We initiate an in-depth proof-complexity analysis of polynomial calculus ($\mathcal{Q}$-PC) for Quantified Boolean Formulas (QBF). In the course of this we establish a tight proof-size characterisation of $\mathcal{Q}$-PC in terms of a suitable circuit model (polynomial decision lists). Using this correspondence we show a size-degree relation for $\mathcal{Q}$-PC, similar in spirit, yet different from the classic size-degree formula for propositional PC by Impagliazzo, Pudlák and Sgall (1999).

We use the circuit characterisation together with the size-degree relation to obtain various new lower bounds on proof size in $\mathcal{Q}$-PC. This leads to incomparability results for $\mathcal{Q}$-PC systems over different fields.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases proof complexity, QBF, polynomial calculus, circuits, lower bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.27

Funding *Olaf Beyersdorff*: Carl-Zeiss Foundation and DFG grant BE 4209/3-1
Tim Hoffmann: Carl-Zeiss Foundation
Kaspar Kasche: Carl-Zeiss Foundation

1 Introduction

Proof complexity studies the problem to understand the minimal size of proofs of specific formulas in various formal proof systems. The field bears deep connections to computational complexity [28,40], logic – mainly through the correspondence to bounded arithmetic [8,27,40] – and has practical significance due to intricate relations to SAT solving [23]. In fact, proof complexity is the main theoretical framework to assess the strength and limitations of solvers.

While traditionally proof complexity concentrated on propositional logic, there has been intense work in the past two decades on proof complexity for further logics, most notably for *Quantified Boolean Formulas* (QBF) [9], but also for other non-classical logics such as modal and intuitionistic logics [19,36,46]. For QBF, one of the main drivers for the field has been significant advances in QBF solving [18,44]. As in the propositional case, QBF proof complexity provides the theoretical tools to model, assess and guide QBF solving [12,21,38].

In propositional proof complexity, various proof systems have been studied intensively, including resolution, Frege systems, algebraic and geometric systems [40]. While resolution has arguably received most attention – and underpins modern SAT solving in the form of CDCL [2,5,42] – algebraic proof complexity has enjoyed a boost of interest in the past decade with many strong results shown for Nullstellensatz, polynomial calculus (PC), sum of squares (SOS), and very strong systems such as the ideal proof system (cf. e.g. [26,29,31,32,34,35,43]).

Algebraic proof systems typically work with polynomials and the central system of polynomial calculus [25] is a refutational proof system demonstrating that a given set of polynomial equations does not admit a common solution.

Similarly, in QBF proof complexity there are many results on various QBF resolution systems [4,9,11,14]. Yet, in stark contrast to the propositional case, information on algebraic proof systems for QBF is rather scarce. A version of polynomial calculus for QBF – called $\mathcal{Q}$-PC here – is straightforward to define [13] as there is a general framework how to lift a line-based propositional proof system P – fulfilling some modest closure properties – to a quantified system $\mathcal{Q}$-P by adding just one rule of universal reduction that allows to substitute a universal variable u from a formula F (under the condition that u is quantified rightmost in F) [13]. This system $\mathcal{Q}$-PC naturally works with polynomials as lines and provides a succinct way to prove the falsity of QBFs. Hence we view this algebraic system as a refutational system for QBFs. The existential and universal variables are therefore propositional and take $0/1$ values in accordance with the QBF semantics, while intermediate values computed by the polynomials can be arbitrary field elements, making proofs more succinct.

So far, the only information on proof size in $\mathcal{Q}$-PC stems from the general semantic technique of cost through the size-cost-capacity theorem from [10] which allows to obtain lower bounds for QBF proof systems of bounded capacity (which applies to $\mathcal{Q}$-PC as well as to most QBF resolution systems). With the cost technique, QBFs become hard to prove whenever the universal player needs large winning strategies (measured as the number of different answers of the universal player in the game interpretation of QBFs) and these lower bounds simultaneously hold in all QBF systems to which this technique is applicable. Hence this method does not allow to separate QBF resolution from $\mathcal{Q}$-PC, for example.

One key motivation to study algebraic proof systems in the propositional case is their recently emerging connection to algebraic circuit complexity [31, 35, 43]. In general, a correspondence between progress for lower bounds for circuit and proof size has often been postulated (e.g. [6]), but formal connections for propositional proofs could not yet be established outside the algebraic domain. In fact, it could be argued that this correspondence perfectly works in the QBF setting: for QBF resolution – tightly corresponding to a version of decision lists [11] – and for QBF Frege systems where proof size is characterised by circuit size in Boolean circuits [13]. This is quite fruitful as it allows a direct transfer of known circuit lower bounds to proof complexity, e.g. from $\mathsf{AC}^0[p]$ to the corresponding system of $\mathcal{Q}$-$\mathsf{AC}^0[p]$-Frege [13,47]. A similar transfer in the propositional case remains wide open.

Curiously, an analogous relation between algebraic circuits and algebraic QBF systems is missing, whereas exactly in this algebraic case, some connections are known propositionally [31,35,43], as mentioned above.

Our aim in this paper is to initiate a comprehensive analysis of the algebraic system $\mathcal{Q}$-PC. In the course of this investigation we achieve a circuit characterisation for $\mathcal{Q}$-PC. This leads to new lower bound techniques for proof size in $\mathcal{Q}$-PC, which we apply to show a number of new proof size lower bounds for this system.

Our contributions

A. Circuit characterisation for $\mathcal{Q}$-PC. Our first result is a tight circuit characterisation of $\mathcal{Q}$-PC proof size by circuit size in an appropriate circuit model. The circuit model in question is a generalisation of decision lists [45], which are lists of simple statements of the form: If *(condition on existential variables)* Then *(assignment to universal variables).*

The decision lists – termed PDLs here for *p*olynomial *d*ecision *l*ists – have polynomial equations in existential variables as conditions and compute a complete assignment to the universal variables. Semantically, a PDL for a quantified set of polynomial equations Φ computes a countermodel for Φ in the two-player game semantics of QBFs.

We show that the minimal proof size for Φ (of bounded quantifier complexity) in $\mathcal{Q}$-PC is polynomially equivalent to the minimal size of a PDL for Φ. In fact, we show a more general result that applies to a whole class of QBF proof systems with bounded capacity [10] (and fulfilling some closure properties). The result is parameterised by the lines of the proof system, which in turn correspond to the conditions in the decision lists. This generalises a result for Q-Resolution [11] and lifts it to $\mathcal{Q}$-PC.

B. Size-degree relation for $\mathcal{Q}$-PC. Having the PDL characterisation in place, we can obtain a size-degree result, relating minimal proof size in $\mathcal{Q}$-PC to the minimal degree of polynomials in the refutation. This is similar in spirit to the size-degree method known for propositional PC [37], albeit the actual relation is different and includes the quantifier depth of the QBF. Technically, the result is shown via the degree-preserving transfer from $\mathcal{Q}$-PC to PDLs and back explained above, together with an additional size-degree relation that we show for PDLs. The technique is similar to a prior size-width result for Q-Resolution [11].

C. New lower bounds for $\mathcal{Q}$-PC. Having both the PDL characterisation and size-degree relation at hand opens the door to new lower bounds for degree and size in $\mathcal{Q}$-PC.

Specifically, we show that the parity and more generally the modulo k functions mod_n^k on n variables as well as the majority function maj_n all require high-degree PDLs over all subfields of $\mathbb{C}$. Using a general construction from [13, 14] we can turn any Boolean function f into a QBF $\mathcal{Q}$-f that has f as its only countermodel. Together with our results above this implies that the $\mathcal{Q}$-mod_n^k and $\mathcal{Q}$-maj_n QBFs require both linear degree and exponential monomial size in $\mathcal{Q}$-PC.

In addition to using the size-degree method to prove lower bounds for PDLs and hence for $\mathcal{Q}$-PC proofs, we show that for finite fields of characteristic p, PDLs can be efficiently transformed into $\mathsf{AC}^0[p]$ circuits. This allows to directly transfer circuit lower bounds of [47] into $\mathcal{Q}$-PC proof lower bounds. As a result, either if F and G are both finite fields of different characteristics, or if F is finite and G is a subfield of $\mathbb{C}$, then the systems $\mathcal{Q}$-PC over F and G are incomparable.

In fact, all our lower bounds are very strong as they apply to a succinct model of QBF proof systems were propositional sub-derivations – for PC comprised of additions and multiplications of polynomials – can be abbreviated as semantic entailment steps that are checked with an NP oracle [17]. This implies that all our lower bounds and incomparability results also hold in the traditional proof model with “unfolded” computations, but remain even valid in the mentioned stronger NP oracle model.

Due to space constraints, some proofs are omitted.

2 Preliminaries

We assume familiarity with basic notions from computational complexity, cf. [1], as well as from logic, cf. [39], and algebra, cf. [41], but define all specific concepts needed in this paper.

We consider propositional formulas φ built from constants $0, 1$, connectors $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and propositional variables. A literal is a variable v or its negation $\overline{v}$. A clause is a disjunction of literals, and a formula is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses. When V is a set of variables, a (partial) assignment to V is a (partial) function $\alpha : V \to \{0,1\}$. We write $\langle V \rangle$ for the set of all complete assignments to V, and $\mathrm{vars}(\varphi)$ or $\mathrm{vars}(\alpha)$ for the set of all variables occurring in φ or α. For $V' \subseteq V$, we denote by $\alpha|_{V'}$ the restriction of α to the variables in V'. We denote by $\varphi[\alpha]$ the formula φ where each variable $v \in \mathrm{vars}(\alpha)$ has been substituted by $\alpha(v)$, and by $\phi[v_1/\theta_1, \ldots, v_k/\theta_k]$ the formula φ where variables v_i have been substituted by formulas θ_i.

Circuit classes. We recall the definitions of standard circuit classes used in this paper. The class AC^0 contains all languages computable by polynomial-size circuits with gates $\neg$, $\vee$, $\wedge$ with bounded depth and unbounded fan-in. The class $\mathsf{AC}^0[p]$ additionally uses MOD_p gates determining whether the sum of the inputs is 0 modulo p. $\mathsf{P/poly}$ uses circuits of polynomial size but arbitrary depth. For an in-depth account on circuit complexity we refer to [48].

Proof systems. We refer to [28] for a formal definition of proof systems and only illuminate certain restrictions. We only consider *line-based proof systems* where proofs consist of a sequence of lines (in our case polynomials) that are derived with certain rules. A propositional *base system* is a line-based proof system with certain very natural restrictions, formally defined in [10]. All propositional proof systems discussed in this paper are base systems.

Polynomial Calculus. Polynomial Calculus (PC) [25] is an algebraic proof system showing that a set of polynomials does not have common roots. For variables $V = \{v_1, \ldots, v_n\}$ over a field F, its lines are polynomial equations $0 = \sum_{i=1}^{m} c_i \prod_{v \in V_i} v$ with $m \in \mathbb{N}, c_i \in F, V_i \subseteq V$. A PC *refutation* starts with a set of polynomials, derives linear combinations of previous polynomials, and ends with the contradiction $0 = 1$. The size of a polynomial is its number of monomials, and the size of a PC refutation is the sum of the sizes of its polynomials.

Here, we view PC as a propositional proof system, and allow only the values 0 and 1 for each variable. This is ensured by including the Boolean axioms $v^2 - v = 0$ for each $v \in V$. In order to represent literals and clauses compactly, we introduce *complementary variables* $\overline{v}$ that are required to have the value $1 - v$, and introduce axioms $v + \overline{v} - 1 = 0$.

Perhaps counterintuitively, a variable that is 0 in a polynomial corresponds to a propositional variable that is true, and 1 corresponds to false. We recall that a polynomial equation is true if its polynomial equals 0. This way, a monomial corresponds to a clause containing the same literals, and a CNF can be efficiently encoded as a set of monomial equations.

When p is a polynomial, v a variable, and $c \in \{0,1\}$, we denote by $p[v/c]$ the polynomial p where v has been replaced by c and $\overline{v}$ by $1 - c$.

Quantified Boolean Formulas. A (closed prenex) *Quantified Boolean Formula* (QBF) is a formula $Q\,\phi$ where ϕ is a propositional formula and Q is the quantifier prefix that quantifies all variables v in ϕ either existentially as $\exists v$ or universally as $\forall v$. We typically use x_i for existentially quantified variables and u_i for universally quantified variables. When a system includes complementary variables $\overline{v}$ as in PC, those do not occur in the quantifier prefix. Their values are determined by the corresponding variables v instead.

Whether a QBF is true or not can be defined recursively. The formula $\forall u\, Q\, \varphi$ is true if both $Q\,\varphi[u = 0]$ and $Q\,\varphi[u = 1]$ are true. The formula $\exists x\, Q\, \varphi$ is true if at least one of $Q\,\varphi[x = 0]$ and $Q\,\varphi[x = 1]$ is true. When a QBF proof system is derived from an algebraic system such as PC, it nonetheless has these Boolean semantics.

In a fully quantified prenex QBF, the quantifier prefix determines a total order of the variables. Given a variable v, we will sometimes refer to the variables preceding v in the prefix as variables *left* of v; analogously we speak of the variables *right* of v.

A QBF $Q_1x_1 \cdots Q_kx_k\,\phi$ can be seen as a game between two players: *universal* ($\forall$) and *existential* ($\exists$). In the i-th step of the game, the player Q_i assigns a value to the variable x_i. The existential player wins if ϕ evaluates to 1 under the assignment constructed in the game. The universal player wins if ϕ evaluates to 0. Given a universal variable u with index i, a *strategy for u* is a function from all variables of index $< i$ to $\{0, 1\}$. A QBF is false if and only if there exists a *winning strategy* for the universal player, that is if the universal player has a strategy for all universal variables that wins any possible game [1, 33].

3 A general characterisation of $\mathcal{Q}$-P proof size by decision lists

We start by characterising proof size in $\mathcal{Q}$-PC by a suitable circuit model. In fact, we will show a more general result that applies to a class of QBF proof systems which are lifted from a propositional base system P to the QBF system $\mathcal{Q}$-P. This lifting is done by adding the $\forall$red rule to the rules of P. The $\forall$red rule allows to derive $l[\mu]$ from a line l and a propositional assignment μ to universal variables, as long as $\mathrm{vars}(\mu)$ occur after all existential variables in l in the quantifier prefix [13].

However, lower bounds for the resulting $P + \forall$red system are trivial to obtain from lower bounds for P by existentially quantifying all variables. We are not too interested in such bounds, but in "genuine" QBF lower bounds that arise from quantifier alternation (cf. [17, 24] for a longer discussion and details). In other words, we want to filter out any propositional hardness in a QBF by ignoring purely propositional sub-derivations. For this, we introduce the Sem rule for semantic steps. It can derive a line l from a line r if $r \models l$. In general this inference step cannot be checked efficiently, but needs an NP oracle call [17]. Using Sem steps also removes the need for any other propositional inference rules as these can be carried out by Sem. We call the resulting system $\mathcal{Q}$-P.

▶ **Definition 1** ($\mathcal{Q}$-P). *Let P be a propositional base system. The system $\mathcal{Q}$-P is a refutational proof system for QBF that has the same lines as P, and rules $\forall$red and Sem.*

The system we are most interested in is $\mathcal{Q}$-PC where the lines are polynomials. We briefly review the semantics of this system. As specified in Definition 1, its only rules are $\forall$red and Sem. Its lines are polynomial equations with coefficients from a field F. The $\forall$red rule can be applied to a polynomial p to obtain $p[u/0]$ (which is p with variable u set to 0 and $\overline{u}$ set to 1) or $p[u/1]$ (which is p with variable u set to 1 and $\overline{u}$ set to 0) as long as u is quantified universally right of all existential variables in p. This also means that u cannot be a complementary variable $\overline{v}$. The Sem rule allows to derive polynomial equations that semantically follow from previous equations, the Boolean axioms, and the $v + \overline{v} = 1$ axioms. Semantically, the variables can only take the values 0 or 1, and $\overline{v}$ must always take the value $1 - v$. The propositional rules of PC can be added, but are not strictly needed and do not shorten proofs in the presence of Sem.

We now define the circuit model that we will use for the characterisation of $\mathcal{Q}$-P proof size. The model is a variant of decision lists [10, 45].

▶ **Definition 2** (P-UDL). *Let P be a base system and X, U sets of variables. A P-UDL of length k is a sequence $L = (p_1, \mu_1), (p_2, \mu_2), \ldots, (p_k, \mu_k)$ where $(\overline{p_i})$ is a line of P, $\mathrm{vars}(p_i) \subseteq X$, $\mu_i \in \langle U \rangle$ for each $i \in [k]$ and $p_k = \top$. It computes a function $f_L : \langle X \rangle \to \langle U \rangle$ with $f_L(\alpha) = \mu_j$ for the smallest j such that $\alpha \models p_j$.*

Intuitively, a P-UDL checks conditions p_j, which are negations of lines of P using only existential variables and outputs a full assignment μ_j to the universal variables.

Again the main instantiation of this definition for us is to choose P as PC. To ease notation we abbreviate PC-UDL by PDL for *p*olynomial *d*ecision *l*ists. The lines of PDLs are then polynomials. As lines in PC are polynomials p with the implicit meaning of $p = 0$, conditions in a PDL check that the polynomial is *not* zero. Hence a line p in a PDL becomes active, if the p does not evaluate to 0 under the assignment.

We use P-UDLs to compute countermodels for QBFs. More formally, we say that a P-UDL L is *correct* for a QBF $\mathcal{Q}.\varphi$ if it has all of the following properties:

- All variables in X are existential variables of $\mathcal{Q}$.
- All universal variables of $\mathcal{Q}$ are in U.
- Let $\alpha, \beta \in \langle X \rangle, u \in U$. If $f_L(\alpha)$ and $f_L(\beta)$ disagree on u, then α and β disagree on a variable $x \in X$ that occurs before u in $\mathcal{Q}$.
- For every $\alpha \in \langle X \rangle$, $\alpha \wedge f_L(\alpha)$ falsifies φ.

For P-UDL $L = (p_1, \mu_1), \ldots, (p_k \mu_k)$, we define the *size* of L, $|L| = \sum_{i=1}^{k} size_P(\overline{p_i})$, where $size_P$ corresponds to the respective size measure in the base system P. Additionally, the length of L is defined as $\text{len}(L) = k$.

Our goal is to use P-UDLs to characterise the hardness of $\mathcal{Q}$-P proofs on QBFs of constant alternation depth, i.e. to show that there exists a short P-UDL for a QBF Φ if and only if there exists a short $\mathcal{Q}$-P proof of Φ.

From the definition of P-UDL, it is apparent that the size of P-UDL for a QBF Φ is always at least as big as the size of the smallest countermodel of Φ, since each line always assigns all universal variables. As such, P-UDL cannot possibly characterise proof size in $\mathcal{Q}$-P for all base systems P. In fact, we need three restrictions on $\mathcal{Q}$-P for the characterisation to work. Firstly, the negations of the lines of P must allow to succinctly represent assignments to variables. Secondly, the base system P must be *closed under disjunction*, i.e. for two arbitrary lines l and p of P, the disjunction $l \vee p$ is also a valid line in P with size $\mathcal{O}(|l| \cdot |p|)$.

Thirdly, we require $\mathcal{Q}$-P to have limited *capacity.* Capacity is a measure introduced in [10], which counts how many different answers the universal player needs at most to respond to one line in a $\mathcal{Q}$-P proof π. In particular, we require that the capacity of $\mathcal{Q}$-P is at most polynomial in the size of π for all proofs π. For the formal definition of capacity, we refer to [10]. We remark that $\mathcal{Q}$-PC and $\mathcal{Q}$-Res (as well as the QBF cutting planes system $\mathcal{Q}$-CP) all have bounded capacity [10].

We are now ready to characterise proof size in $\mathcal{Q}$-P by the size of P-UDLs.

▶ **Theorem 3.** *Let P be a line-based, propositional proof system, where the lines are closed under disjunction, such that $\mathcal{Q}$-P has at most polynomial capacity. Then for QBFs Φ with bounded quantifier alternation depth, we can efficiently transform a P-*UDL *for Φ into a $\mathcal{Q}$-P refutation for Φ and vice versa. In particular, the minimal size of a P-*UDL *for Φ is polynomially equivalent to the size of the minimal $\mathcal{Q}$-P refutation for Φ.*

Proof sketch. The proof outline follows [11]. *From $\mathcal{Q}$-P to P-*UDL. Let a QBF Φ of alternation depth d and a $\mathcal{Q}$-P refutation π of Φ be given. Using bounded capacity, we can show that w.l.o.g. π always uses universal reductions on entire blocks of universal variables. Employing the paradigm of strategy extraction [3, 11, 30], we can extract a set of P-decision lists computing a countermodel of Φ from π. Each of these decision lists computes the universal strategy for one of the universal blocks and has size at most $|\pi|$. We combine these decision lists to a single decision list, whose size is at most the product of the sizes of the original lists, using the *direct product* construction from [11]. The resulting decision list computes the correct function and is a P-UDL. Since there are d decision lists of size $|\pi|$, the computed P-UDL has size $\mathcal{O}(|\pi|^d)$.

From P-UDL to $\mathcal{Q}$-P. Let a QBF Φ of alternation depth d and a P-UDL L computing a countermodel of Φ be given. We define the *entailment sequence* $\mathcal{E}$ of L recursively in d.

- if $d = 1$, $\mathcal{E}(L) := \overline{\varepsilon_1} \vee \overline{\mu_1}, \dots, \overline{\varepsilon_s} \vee \overline{\mu_s}$
- if $d \geq 2$, for each $i \in [s]$ define L_i as the list obtained from L by replacing the first $i - 1$ existential terms by their X_1 components and setting all U_1 components to $\mu_i|_{U_1}$. $\mathcal{E}(L)$ is the sequence $\pi_1, \dots, \pi_s$, where $\pi_i := (\overline{\varepsilon_i}|_{X_1} \vee \overline{\mu_i}|_{U_1}) \otimes \mathcal{E}(L_i[\alpha])$ and α is some arbitrary but fixed assignment on $X_1 \cup U_1$ satisfying $\varepsilon_i|_{X_1} \wedge \mu_i|_{U_1}$.

Here, the $\otimes$-operator between a line of P and the entailment sequence defines an elementwise disjunction between the line and each element in the entailment sequence, i.e. $\ell \otimes (c_1, c_2, \dots, c_r) = (\ell \vee c_1, \ell \vee c_2, \dots, \ell \vee c_r)$. For a line ℓ, we call $\mathrm{red}(\ell)$ the line obtained by using universal reduction on ℓ by some specific universal assignment. Since negations of lines of P can succinctly represent assignments, $\overline{\mu_i}$ can be represented in P. If $\mathcal{E}(L) = (c_1, \dots, c_r)$, then $(c_1, \mathrm{red}(c_1), c_2, \mathrm{red}(c_2), \dots, c_r, \mathrm{red}(c_r))$ is a $\mathcal{Q}$-P refutation of Φ. ◀

This reproves a characterisation of QBF resolution by decision lists from [11]. The main application for us is polynomial calculus (PC) for which this result is new. PC has a capacity of $\sqrt{n}$, where n is the size of the proof [10]. Additionally, PC is closed under disjunctions as the disjunction of two lines can be expressed as the product of the respective polynomials. The size of the disjunction is the product of the sizes of the original lines.

▶ **Corollary 4.** *For QBFs of bounded quantifier depth, the minimal size of $\mathcal{Q}$-PC proofs and the minimal PDL sizes are polynomially equivalent.*

4 Size-degree bounds for polynomial calculus in QBF

Using the connection between $\mathcal{Q}$-PC and PDLs from Corollary 4, we now aim to show lower bounds for $\mathcal{Q}$-PC by proving lower bounds for PDLs. The latter task will be simplified by a relation between PDL size and PDL degree, which is measured as the maximal degree of the polynomial conditions in the PDL. As these polynomials are just defined in existential variables, it makes sense to define the *existential degree* for PDLs and $\mathcal{Q}$-PC refutations. This is in line with an analogous definition of existential width for Q-Resolution [11, 15].

▶ **Definition 5** (Existential degree of a $\mathcal{Q}$-PC refutation). *Let $f = \exists X_1 \forall U_1 \cdots \exists X_{d+1}.\varphi$, $\pi = (\pi_1, \dots, \pi_s)$ be a $\mathcal{Q}$-PC refutation of f and $X = \bigcup_{i=1}^{d} X_i$. The* existential degree $\deg_\exists$ *of f is defined as $\deg_\exists(\pi) = \max_{i \in [s]} \deg(\pi_i|_X)$.*

Analogously, the degree of a PDL L is defined as the maximal degree of all queries in L. We can show a size-degree relation for PDLs.

▶ **Theorem 6.** *Let f be a multi-output Boolean function with n input variables. If f is computed by a PDL of size s, it is also computed by a PDL of degree $O(\sqrt{n \log s})$.*

In essence, the proof of this theorem is the same as the original size-degree result for propositional polynomial calculus by Impagliazzo, Pudlák and Sgall [37] (cf. also [7, 22] for similar arguments). Equivalently, a function f that can only be computed by PDLs of degree at least k needs PDLs of size $\exp(\Omega(\frac{k^2}{n}))$.

We can use this size-degree relation for PDLs to obtain a size-degree relation on $\mathcal{Q}$-PC.

▶ **Theorem 7.** *Let f be a QBF with n variables of alternation depth d such that every $\mathcal{Q}$-PC proof has existential degree at least k. Then every $\mathcal{Q}$-PC proof has size at least $\exp(\Omega(\frac{k^2}{d^3 n}))$.*

Proof. Let $f = \exists X_1 \forall U_1 \cdots \exists X_{d+1}.\varphi$, $X = \bigcup_{i=1}^{d} X_i$ the set of existential variables except the last block and $|X| = v$. Additionally, let π be a shortest $\mathcal{Q}$-PC refutation of f. Using Corollary 4, π can be transformed into a PDL L of size at most $|\pi|^d$. With Theorem 6, L can then be transformed into a PDL M with degree at most $\mathcal{O}(\sqrt{dv\log|\pi|})$. Transforming M back into a $\mathcal{Q}$-PC proof with Corollary 4 results in a proof π' with degree at most $k = \mathcal{O}(d\sqrt{dv\log|\pi|})$. Solving this equation for $|\pi|$ yields the result. ◀

The proof is similar to Theorem 6.2 in [11]. In contrast to the size-degree relation from [37], Theorem 7 includes the quantifier depth of the QBF, but not the initial degree of the QBF.

In the rest of this section, we will explore specific lower bounds for the degree of PDLs. We will first show degree lower bounds for PDLs computing specific functions and then turn this into $\mathcal{Q}$-PC size lower bounds for related QBFs.

4.1 Parity and mod functions require PDLs of high degree

Let $\mathrm{par}_n(x_1, \ldots, x_n) = \bigoplus_{i=1}^{n} x_n$ be the parity function. Lower bounds for this function only hold for PDLs over certain fields; in fact, par_n is just the sum of input variables in fields of characteristic 2, and is therefore trivial for PDLs over those fields. However, it seems to be hard in fields of characteristic 0. We prove a lower bound for $\mathbb{C}$ and its subfields.

▶ **Proposition 8.** *A PDL with polynomials over a subfield of* $\mathbb{C}$ *computing* par_n *has degree at least* $\frac{n}{2}$.

Proof. We consider the first line of the PDL and assume without loss of generality that it has output 1.[1]

Let p be the first line's polynomial and d its degree. Let $X = \{x_1, \ldots, x_n\}$. We can assume that there is a $w \in \langle X \rangle$ with $p(w) \neq 0$ or we could omit the first line. However, to avoid giving any wrong answers, for every $a \in \langle X \rangle$ with $\mathrm{par}_n(a) = 0$, it must hold that $p(a) = 0$.

We compute the complex conjugate p^* by conjugating every coefficient. Because we only evaluate the polynomials on real numbers (specifically 0 and 1), we know that $p^*(a) = (p(a))^*$ for every $a \in \langle X \rangle$. We define $q := p \cdot p^*$ and note that $\deg(q) \leq 2d$ and for all $a \in \langle X \rangle$, $q(a) = p(a) \cdot p(a)^* \in \mathbb{R}^{\geq 0}$.

For a polynomial r, define the function

$$s(r) := \sum_{\substack{a \in \langle X \rangle \\ \mathrm{par}(a)=1}} r(a) - \sum_{\substack{a \in \langle X \rangle \\ \mathrm{par}(a)=0}} r(a)$$

which is linear with respect to r. If r is a monomial that does not contain the variable x,

$$\begin{aligned} s(r) &= \sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ b \in \langle \{x\} \rangle \\ \mathrm{par}(a,b)=1}} r(a,b) - \sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ b \in \langle \{x\} \rangle \\ \mathrm{par}(a,b)=0}} r(a,b) = \sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ b \in \langle \{x\} \rangle \\ \mathrm{par}(a,b)=1}} r(a) - \sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ b \in \langle \{x\} \rangle \\ \mathrm{par}(a,b)=0}} r(a) \\ &= \left(\sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ \mathrm{par}(a)=1}} r(a) + \sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ \mathrm{par}(a)=0}} r(a) \right) - \left(\sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ \mathrm{par}(a)=0}} r(a) + \sum_{\substack{a \in \langle X \setminus \{x\} \rangle \\ \mathrm{par}(a)=1}} r(a) \right) \\ &= 0. \end{aligned}$$

[1] If it has output 0, we can invert all outputs in the PDL and replace every occurence of x_1 with $1 - x_1$. This does not change its degree, and the resulting PDL computes $\neg\mathrm{par}_n(\overline{x_1}, x_2, \ldots, x_n) = \mathrm{par}_n(x_1, \ldots, x_n)$.

Because s is linear, $s(r) = 0$ also holds for any polynomial r of degree $< n$.

To obtain a value of $s(q)$, we use that

$$\sum_{\substack{a\in\langle X\rangle\\ \mathrm{par}(a)=1}} q(a) > 0 \quad \text{and} \quad \sum_{\substack{a\in\langle X\rangle\\ \mathrm{par}(a)=0}} q(a) = 0$$

(in the left equation, none of the summands are negative; one of them is $q(w) = |p(w)|^2 > 0$). Consequently, $s(q) > 0$ and $\deg(q) \geq n$, so using $\deg(q) \leq 2d$ we conclude that $d \geq \frac{n}{2}$. ◀

To turn this lower bound into a $\mathcal{Q}$-PC bound we need QBFs based on the parity function. For this we describe a general transformation originating from [13, 14] to construct QBFs that are false, but force the universal player to use a unique strategy by computing a particular function f. We define the QBF $\mathcal{Q}$-f:

▶ **Definition 9** ($\mathcal{Q}$-f). *Let $f : \langle X\rangle \to \langle U\rangle$ be a function that is computed by a* P/poly *circuit C. Then $\mathcal{Q}$-$f := \exists X\forall U\exists T\,\varphi$ where φ is the Tseitin transformation of the circuit $U \neq C(X)$, and T is the corresponding set of auxiliary Tseitin variables.*

In our case above, f is par_n and C is a simple P/poly circuit for par_n. The existential player wins if and only if the circuit $U \neq C(X)$ yields true, after the assignments to X and U were chosen by the respective players. The universal player therefore has the unique winning strategy of playing $U = f(X)$.

From Proposition 8 together with the size-degree relation for PDLs (Theorem 6) and the efficient transformation into $\mathcal{Q}$-PC (Theorem 3) we obtain:

▶ **Corollary 10.** *$\mathcal{Q}$-PC refutations over a subfield of $\mathbb{C}$ of $\mathcal{Q}$-par_n require size $\exp(\Omega(n))$.*

We can generalise this to the modulo k functions. Let $\mathrm{mod}_n^k(x_1, \ldots, x_n) = 1$ if and only if $\sum_{i=1}^n x_i \equiv 0 \mod k$ (otherwise it is 0). With a similar, but somewhat more technical proof than for Proposition 8 we can show:

▶ **Proposition 11.** *A PDL with polynomials over a subfield of $\mathbb{C}$ computing mod_n^k has degree at least $\frac{1}{2}\left\lfloor \frac{n}{(k-1)} \right\rfloor$.*

Consequently, the $\mathcal{Q}$-mod_n^k QBFs are hard for $\mathcal{Q}$-PC over $\mathbb{C}$.

▶ **Corollary 12.** *$\mathcal{Q}$-PC refutations over a subfield of $\mathbb{C}$ of $\mathcal{Q}$-mod_n^k have size $\exp(\Omega(\frac{n}{k}))$.*

4.2 Majority PDLs have high degree

Next we want to show degree lower bounds for PDLs that compute the majority functions $\mathrm{maj}_n(x_1, \ldots, x_n)$, which evaluate to one, if and only if $\sum_{i=1}^n x_i \geq \frac{n}{2}$. In contrast to the par_n and mod_n^k functions, this lower bound does not depend on the underlying field. We start by proving a useful lemma about PDLs:

▶ **Lemma 13.** *Let $X = \{x_1, \ldots, x_n\}$, $\alpha \in \langle X\rangle$ a complete assignment and p a polynomial with variables in X that is not the constant 0. There is an assignment $\beta \in \langle X\rangle$ such that α and β only differ in at most $\deg(p)$ variables, and $p(\beta) \neq 0$.*

Proof. We start by taking p and constructing an equivalent polynomial q that contains no variables according to their polarity in α: when $\alpha(x) = 0$, we replace $\overline{x}$ by $1 - x$, and when $\alpha(x) = 1$, we replace x by $1 - \overline{x}$. Let m be one of the lowest-degree monomials in q, and β the assignment that satisfies every literal in m, and assigns every other variable according to α. Note that α and β only differ in at most $\deg(m) \leq \deg(q) = \deg(p)$ variables. Because m has minimal degree, every other monomial in q contains a variable $v \notin \mathrm{vars}(m)$, so $\beta(v) = \alpha(v)$. Because v only occurs in the polarity opposite of $\alpha(v)$, β does not satisfy that other monomial. Therefore, $0 \neq m(\beta) = q(\beta) = p(\beta)$. ◀

With this, we can show the lower bound fairly easily:

▶ **Proposition 14.** *A PDL computing* $\mathrm{maj}_n(x_1, \dots, x_n)$ *has degree at least* $\frac{n}{2}$.

Proof. Let L be such a PDL, d its degree, and p its first polynomial. We also define $X := \{x_1, \dots, x_n\}$. If the first line of L has output 1, let α be the assignment that assigns all $x_i = 0$, and apply Lemma 13. Because $p(\beta) \neq 0$ and L is correct, we obtain $\mathrm{maj}_n(\beta) = 1$. So β has to assign 1 to at least $\frac{n}{2}$ variables and thus differs from α in at least $\frac{n}{2}$ variables. Therefore, $\frac{n}{2} \leq \deg(p) \leq d$. If the first line has output 0 instead, let α be the assignment that assigns all $x_i = 1$. Lemma 13 yields an assignment β with $p(\beta) \neq 0$, so $\mathrm{maj}_n(\beta) = 0$ and β has to assign 0 to at least $\frac{n}{2}$ variables. Therefore, $\frac{n}{2} \leq \deg(p) \leq d$. ◀

▶ **Corollary 15.** *Q-PC refutations of* $\mathcal{Q}$-maj_n *have size* $\exp(\Omega(n))$.

4.3 Limits of the size-degree method

While the size-degree technique is useful for showing several lower bounds as demonstrated above, it does not capture all nuances of PDL size. We will illustrate this by constructing two functions, each having n^2 variables and requiring PDLs of degree n. For these values, Theorem 6 does not yield any useful lower bound. Indeed, one of the functions requires PDLs of size $\exp(\Omega(n))$, while for the other one, size $O(n)$ is sufficient.

▶ **Theorem 16.** *Given a function f, the minimal size of its PDLs is not solely determined by its number of variables and minimal PDL degree. In particular, there are families of functions f_n and g_n, each with n^2 input variables and minimal PDL degree n, such that f_n has PDLs of size $O(n)$ and g_n requires PDLs of size* $\exp(\Omega(n))$.

In order to prove this, we introduce examples for those functions g and f. We define the square-majority function as $\mathrm{sqm}_n(x_1, \dots, x_{n^2}) = 1$ if and only if $\sum_{i=1}^{n^2} x_i \geq n$, otherwise 0.

▶ **Lemma 17.** *The* sqm_n *function can be computed by PDLs of degree n, but not by PDLs of smaller degree. It requires PDLs of size* $\exp(\Omega(n))$.

The lower bound on the size of PDLs for sqm already cannot be shown by Theorem 6. This raises the question of whether our size-degree relation is simply too weak, and whether we could obtain a stronger one that can capture the complexity of the sqm function. It turns out that this is not the case: there are functions with the same degree and number of variables, but smaller decision lists. The complexity of the sqm function cannot be derived from its degree and number of variables alone.

To obtain such a function, let $X = \{x_i^j \mid i, j \in \{1, \dots, n\}\}$ and $f_n(X) := \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n} x_i^j$.

▶ **Lemma 18.** *The function f_n defined above can be computed by PDLs of degree n and size $O(n)$, but not by PDLs of smaller degree.*

Proof of Theorem 16. Let f_n be the function defined above, and $g_n = \mathrm{sqm}_n$. The statement follows directly from Lemmas 17 and 18. ◀

5 Converting PDLs into Boolean circuits

We now establish a different lower bound method for $\mathcal{Q}$-PC that directly imports circuit lower bounds. For this we show a connection between PDLs over finite fields and $AC^0[p]$ circuits. First we convert PDLs over a finite field to PDLs over a prime-order finite field.

▶ **Lemma 19.** *Let p be a prime, $k, l, m \in \mathbb{N}^{\geq 1}$ and L a PDL of length l and size m over the finite field $\mathbb{F}_{p^k}$. Then L can be converted into a PDL of length $k \cdot l$ and size $k \cdot m$ over the finite field $\mathbb{F}_p$ that computes the same function.*

This lemma is based on the fact that $\mathbb{F}_{p^k}$ has the structure of a k-dimensional $\mathbb{F}_p$ vector space with respect to addition. An equation in $\mathbb{F}_{p^k}$ that does not use multiplication can therefore be split up into k equations in $\mathbb{F}_p$. The only multiplication that occurs in the polynomials of a PDL is between coefficients and the corresponding unit monomials, and those unit monomials can only be 1 or 0. We can use this to convert each line of a PDL over $\mathbb{F}_{p^k}$ into k lines over $\mathbb{F}_p$.

We will now efficiently convert PDLs over $\mathbb{F}_p$ for primes p into $AC^0[p]$ circuits.

▶ **Proposition 20.** *Let $f : \{0,1\}^n \to \{0,1\}^s$ be a function that is computed by a PDL of size m over a finite field $\mathbb{F}_p$. Then f can be computed by an $\mathsf{AC}^0[p]$ circuit of depth 6 with only $O(pm + s)$ AND or OR gates.*

Proof. We construct the circuit iteratively starting at the inputs and ending at the output. At each layer, we describe the semantics of the newly-added circuit gates.

- Start with v input gates for the variables.
- Add v NOT gates, each negating one of the variables. All literals are now represented in the circuit.
- Consider each monomial $c \cdot \prod_i t_i$ where $c \in \mathbb{F}_p$ and the t_i are literals. Add c identical AND gates with all the t_i as inputs. The sum of the outputs of these gates will be equivalent to the value of the monomial. Because each $c < p$, the total number of these gates is smaller than pm.
- For each line, add a MOD_p gate over all the AND gates of its monomials. The sum of its inputs will be equivalent to the sum of the values of its monomials, so its output indicates whether the polynomial equation holds in $\mathbb{F}_p$.
- For each line, add a NOT gate negating the MOD_p gate.
- For each line, add an AND gate that checks if all polynomial equations of previous lines hold, but the equation of the current line does not. Its output indicates whether this line's output value is active.
- Finally, for each output variable z add an OR gate connecting the AND gates of those lines whose output sets $z = 1$. Its output indicates whether the PDL sets $z = 1$, it is the output of the circuit for variable z.

This circuit has depth 6 and at most $O(pm + s)$ AND or OR gates. ◀

Using Lemma 19 we can extend this result to fields $\mathbb{F}_{p^k}$:

▶ **Corollary 21.** *Let $f : \{0,1\}^n \to \{0,1\}^s$ be a function that is computed by a PDL of size m over a finite field $\mathbb{F}_{p^k}$. Then f can be computed by an $\mathsf{AC}^0[p]$ circuit of depth 6 with only $O(pmk + s)$ AND or OR gates.*

This corollary allows us to lift lower bounds for $\mathsf{AC}^0[p]$ circuits to PDLs over finite fields. We cite such a circuit lower bound by Smolensky [47]. Let p be a prime and r not a power of p. An $\mathsf{AC}^0[p]$ circuit of depth k that computes mod_n^r requires $\exp(\Omega(n^{\frac{1}{2k}}))$ AND or OR gates. Using Corollary 21 we can apply this result to PDLs:

▶ **Corollary 22.** *Let p, r be distinct primes. A PDL over the finite field $\mathbb{F}_{p^k}$ that computes* mod_n^r *needs size* $\exp(\Omega(\sqrt[12]{n} - \log(pk)))$*.*

We also want to apply these results to another class of functions that we call balance. $\mathrm{balance}_{2n}(x_1, x_2, \ldots, x_{2n}) = 1$ if and only if $\sum x_i = n$, else it is zero.

We now show that PDLs for the balance function can be transformed into $AC^0[p]$ circuits for the function mod_n^q with arbitrary q.

▶ **Proposition 23.** *If there is a PDL of size m over $\mathbb{F}_p$ that computes* $\mathrm{balance}_{2n}$*, then* mod_n^q *can be computed by an* $\mathsf{AC}^0[p]$ *circuit of depth 8 that uses only $O(pnm)$* AND *or* OR *gates.*

Proof. Let $R = \{q \cdot i \mid i \in \mathbb{Z}, 0 \le q \cdot i \le n\} = q\mathbb{Z} \cap [0; n]$, and note that $|R| = \left\lceil \frac{n+1}{q} \right\rceil$. When exactly k of the variables $x_1, \ldots, x_n$ are true, then $\mathrm{mod}_n^q(x_1 \ldots, x_n) = 1$ if and only if $k \in R$.

Using Proposition 20 we can obtain a circuit that computes $\mathrm{balance}_{2n}$ with $pm + l$ AND gates and one OR gate. We instantiate this circuit once for each $k \in R$, replacing the $2n$ inputs with n actual inputs $x_1, \ldots, x_n$, as well as k constants 0 and $n - k$ constants 1. The output of such an instance is 1 if and only if k of the actual inputs are 1. We use a single OR gate over all these outputs to obtain $\mathrm{mod}_n^q(x_1 \ldots, x_n)$. The total number of AND or OR gates is $|R|(pm + l) + 1 = \left\lceil \frac{n+1}{q} \right\rceil (pm + l) + 1 \le (n + 1)(p + 1)m + 1$. ◀

▶ **Corollary 24.** *The balance formulas require exponential-sized PDLs over finite fields.*

We can now show that many of the systems $\mathcal{Q}$-PC over different fields are incomparable.

▶ **Theorem 25.** *Let F be a finite field, and G either a finite field with different characteristic, or a subfield of $\mathbb{C}$. Then the $\mathcal{Q}$-PC systems over F and G are incomparable.*

Proof. Because PC systems and PDLs are polynomially equivalent, we can use exponential separations between the respective PDLs to obtain the result. Let p and q be the characteristics of F and G, respectively. The function mod_n^p is trivial for PDLs over F and requires only linear size. However, it requires exponential-sized PDLs over G, either due to Corollary 12 (if G is a finite field) or due to Corollary 22 (if G is a subfield of $\mathbb{C}$).

If G is a finite field, the function mod_n^q requires exponential-size PDLs over F and only linear-size PDLs over G. If G is a subfield of $\mathbb{C}$, the $\mathrm{balance}_n$ functions can be computed in linear size by PDLs over G, but require exponential-size PDLs over F by Corollary 24. ◀

6 Conclusion

While we concentrated on $\mathcal{Q}$-PC in this paper, it is interesting to explore the wider consequences of our P-UDL characterisation in Theorem 3 for further proof systems $\mathcal{Q}$-P. Besides PC and Res, the most studied base systems are arguably *Cutting Planes* (CP) and the various $\mathcal{C}$-Frege systems, where $\mathcal{C}$ is some circuit class, e.g. $\mathsf{AC}^0, \mathsf{NC}^1$ or P/poly.

While $\mathcal{Q}$-CP has polynomial capacity (the capacity is 1) [10], the lines are not closed under disjunction. Hence we cannot use Theorem 3 to obtain a CP-UDL characterisation. We leave open, whether the result itself does not hold or if it only requires a different proof.

For $\mathcal{Q}$-$\mathcal{C}$-Frege, we cannot use Theorem 3 either. Here the lines are closed under disjunction, but the capacity is exponential. However, it is known that the circuit class $\mathcal{C}$ itself, which is a strictly stronger model than $\mathcal{C}$-UDL, tightly characterises $\mathcal{Q}$-$\mathcal{C}$-Frege [20]. As such, the equivalence between $\mathcal{C}$-UDLs and $\mathcal{Q}$-$\mathcal{C}$-Frege fails.

Interestingly, if we consider treelike $\mathcal{Q}$-$\mathcal{C}$-Frege systems, the $\mathcal{C}$-UDL characterisation does hold, even though the capacity is still superpolynomial. Intuitively, this could be explained by the fact that limited capacity is only required for blockwise reductions, and such reductions can (possibly) be obtained by combining reductions along a path in the proof-tree. We prove the result even without using Theorem 3, as a direct proof is straightforward to obtain using a previous characterisation of treelike $\mathcal{Q}$-$\mathfrak{C}$-Frege from [16]. We just state the result.

▶ **Theorem 26.** *For a circuit class* $\mathfrak{C}$ *and a QBF family, the minimal sizes of* $\mathcal{Q}$-$\mathfrak{C}$-$\mathsf{Frege}_{\text{tree}}$ *proofs and the minimal* $\mathfrak{C}$-UDL *sizes are polynomially bounded by each other.*

Interestingly, this result even holds for QBFs with unbounded quantifier alternation. We leave open whether similar characterisations can be obtained for daglike systems $\mathcal{Q}$-P on formulas with unbounded alternation depth.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

2 Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. `doi:10.1613/jair.3152`.

3 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Form. Methods Syst. Des.*, 41(1):45–65, 2012. `doi:10.1007/s10703-012-0152-6`.

4 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Proc. Theory and Applications of Satisfiability Testing (SAT)*, pages 154–169, 2014. `doi:10.1007/978-3-319-09284-3_12`.

5 Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *J. Artif. Intell. Res.*, 22:319–351, 2004. `doi:10.1613/jair.1410`.

6 Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, and future. In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Entering the 21st Century*, pages 42–70. World Scientific Publishing, 2001.

7 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow - resolution made simple. *J. ACM*, 48(2):149–169, 2001. `doi:10.1145/375827.375835`.

8 Olaf Beyersdorff. On the correspondence between arithmetic theories and propositional proof systems – a survey. *Mathematical Logic Quarterly*, 55(2):116–137, 2009. `doi:10.1002/malq.200710069`.

9 Olaf Beyersdorff. Proof complexity of quantified Boolean logic – a survey. In Marco Benini, Olaf Beyersdorff, Michael Rathjen, and Peter Schuster, editors, *Mathematics for Computation (M4C)*, pages 353–391. World Scientific, Singapore, 2022.

10 Olaf Beyersdorff, Joshua Blinkhorn, and Luke Hinde. Size, cost, and capacity: A semantic technique for hard random QBFs. *Logical Methods in Computer Science*, 15(1), 2019. `doi:10.23638/LMCS-15(1:13)2019`.

11 Olaf Beyersdorff, Joshua Blinkhorn, Meena Mahajan, and Tomás Peitl. Hardness characterisations and size-width lower bounds for QBF resolution. *ACM Trans. Comput. Log.*, 24(2):10:1–10:30, 2023. `doi:10.1145/3565286`.

12 Olaf Beyersdorff and Benjamin Böhm. Understanding the relative strength of QBF CDCL solvers and QBF resolution. *Log. Methods Comput. Sci.*, 19(2), 2023. `doi:10.46298/lmcs-19(2:2)2023`.

13 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Jan Pich. Frege systems for quantified Boolean logic. *J. ACM*, 67(2):9:1–9:36, 2020. `doi:10.1145/3381881`.

14 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. New resolution-based QBF calculi and their proof complexity. *ACM Transactions on Computation Theory*, 11(4):26:1–26:42, 2019. `doi:10.1145/3352155`.

15 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Are short proofs narrow? QBF resolution is *not* so simple. *ACM Trans. Comput. Log.*, 19(1):1:1–1:26, 2018. doi:10.1145/3157053.

16 Olaf Beyersdorff and Luke Hinde. Characterising tree-like Frege proofs for QBF. *Inf. Comput.*, 268, 2019. doi:10.1016/j.ic.2019.05.002.

17 Olaf Beyersdorff, Luke Hinde, and Ján Pich. Reasons for hardness in QBF proof systems. *ACM Transactions on Computation Theory*, 12(2):10:1–10:27, 2020. doi:10.1145/3378665.

18 Olaf Beyersdorff, Mikolás Janota, Florian Lonsing, and Martina Seidl. Quantified Boolean formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 1177–1221. IOS Press, 2021. doi:10.3233/FAIA201015.

19 Olaf Beyersdorff and Oliver Kutz. Proof complexity of non-classical logics. In N. Bezhanishvili and V. Goranko, editors, *Lectures on Logic and Computation - ESSLLI 2010 /ESSLLI 2011, Selected Lecture Notes*, pages 1–54. Springer-Verlag, Berlin Heidelberg, 2012. doi:10.1007/978-3-642-31485-8_1.

20 Olaf Beyersdorff and Ján Pich. Understanding Gentzen and Frege systems for QBF. In *Proc. ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016. doi:10.1145/2933575.2933597.

21 Benjamin Böhm, Tomás Peitl, and Olaf Beyersdorff. QCDCL with cube learning or pure literal elimination - what is best? In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1781–1787. ijcai.org, 2022. doi:10.24963/ijcai.2022/248.

22 Nader H. Bshouty. A subexponential exact learning algorithm for DNF using equivalence queries. *Inf. Process. Lett.*, 59(1):37–39, 1996. doi:10.1016/0020-0190(96)00077-4.

23 Sam Buss and Jakob Nordström. Proof complexity and SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, pages 233–350. IOS Press, 2021. doi:10.3233/FAIA200990.

24 Hubie Chen. Proof complexity modulo the polynomial hierarchy: Understanding alternation as a source of hardness. *ACM Transactions on Computation Theory*, 9(3):15:1–15:20, 2017. doi:10.1145/3087534.

25 Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proc. 28th ACM Symposium on Theory of Computing*, pages 174–183, 1996. doi:10.1145/237814.237860.

26 Jonas Conneryd, Susanna F. de Rezende, Jakob Nordström, Shuo Pang, and Kilian Risse. Graph colouring is hard on average for polynomial calculus and Nullstellensatz. In *64th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–11. IEEE, 2023. doi:10.1109/FOCS57990.2023.00007.

27 Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity.* Cambridge University Press, 2010.

28 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979. doi:10.2307/2273702.

29 Susanna F. de Rezende, Mika Göös, Jakob Nordström, Toniann Pitassi, Robert Robere, and Dmitry Sokolov. Automating algebraic proof systems is NP-hard. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 209–222. ACM, 2021. doi:10.1145/3406325.3451080.

30 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In *Proc. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 291–308, 2013. doi:10.1007/978-3-642-45221-5_21.

31 Michael A. Forbes, Amir Shpilka, Iddo Tzameret, and Avi Wigderson. Proof complexity lower bounds from algebraic circuit complexity. *Theory Comput.*, 17:1–88, 2021. URL: https://theoryofcomputing.org/articles/v017a010/.

32 Nicola Galesi, Joshua A. Grochow, Toniann Pitassi, and Adrian She. On the algebraic proof complexity of tensor isomorphism. In Amnon Ta-Shma, editor, *38th Computational Complexity Conference (CCC)*, volume 264 of *LIPIcs*, pages 4:1–4:40. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CCC.2023.4.

33 Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In *IJCAI*, pages 546–553, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-099.

34 Nashlen Govindasamy, Tuomas Hakoniemi, and Iddo Tzameret. Simple hard instances for low-depth algebraic proofs. In *63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 188–199. IEEE, 2022. doi:10.1109/FOCS54457.2022.00025.

35 Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6):37:1–37:59, 2018. doi:10.1145/3230742.

36 Pavel Hrubeš. On lengths of proofs in non-classical logics. *Ann. Pure Appl. Logic*, 157(2–3):194–205, 2009. doi:10.1016/j.apal.2008.09.013.

37 Russell Impagliazzo, Pavel Pudlák, and Jirí Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Comput. Complex.*, 8(2):127–144, 1999. doi:10.1007/s000370050024.

38 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015. doi:10.1016/j.tcs.2015.01.048.

39 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*, volume 60 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1995.

40 Jan Krajíček. *Proof complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, 2019.

41 Serge Lang. *Algebra (3. ed.)*. Addison-Wesley, 1993.

42 Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011. doi:10.1016/j.artint.2010.10.002.

43 Toniann Pitassi and Iddo Tzameret. Algebraic proof complexity: progress, frontiers and challenges. *SIGLOG News*, 3(3):21–43, 2016. doi:10.1145/2984450.2984455.

44 Luca Pulina and Martina Seidl. The 2016 and 2017 QBF solvers evaluations (QBFEVAL'16 and QBFEVAL'17). *Artif. Intell.*, 274:224–248, 2019. doi:10.1016/j.artint.2019.04.002.

45 Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987. doi:10.1007/BF00058680.

46 Sarah Sigley and Olaf Beyersdorff. Proof complexity of modal resolution. *J. Autom. Reason.*, 66(1):1–41, 2022. doi:10.1007/s10817-021-09609-9.

47 R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 77–82. ACM Press, 1987. doi:10.1145/28395.28404.

48 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999. doi:10.1007/978-3-662-03927-4.

Generalized Completion Problems with Forbidden Tournaments

Zeno Bitter
Hamburg University of Technology, Research Group for Theoretical Computer Science, Germany

Antoine Mottet
Hamburg University of Technology, Research Group for Theoretical Computer Science, Germany

Abstract

A recent result by Bodirsky and Guzmán-Pro gives a complexity dichotomy for the following class of computational problems, parametrized by a finite family $\mathcal{F}$ of finite tournaments: given an undirected graph, does there exist an orientation of the graph that avoids every tournament in $\mathcal{F}$? One can see the edges of the input graphs as constraints imposing to find an orientation. In this paper, we consider a more general version of this problem where the constraints in the input are not necessarily about pairs of variables and impose local constraints on the global oriented graph to be found. Our main result is a complexity dichotomy for such problems, as well as a classification of such problems where the yes-instances have bounded treewidth duality. As a consequence, we obtain a streamlined proof of the result by Bodirsky and Guzmán-Pro using the theory of smooth approximations due to Mottet and Pinsker.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Theory of computation → Complexity theory and logic

Keywords and phrases Tournaments, completion problems, constraint satisfaction problems, homogeneous structures, polymorphisms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.28

1 Introduction

1.1 Completion Problems

For every fixed finite family $\mathcal{F}$ of tournaments, the $\mathcal{F}$-free orientation problem consists in deciding whether an undirected graph G admits an orientation G^* such that no tournament from $\mathcal{F}$ is contained in G^*. The complexity of such problems was systematically studied recently in [5], where it is shown that every such problem is solvable in polynomial time or NP-complete.

We extend this result by considering the following variation of the problem. Fix an $r \geq 2$ and let R be a set of tournaments on r vertices, labelled with the numbers $1, \ldots, r$. An input in our problem is a set V of vertices where some r-tuples are marked. The yes-instances to this problem are those where there exists an $\mathcal{F}$-free digraph on V such that whenever $(x_1, \ldots, x_r)$ is marked, then the labelled subdigraph induced on $\{x_1, \ldots, x_r\}$ is isomorphic to an element of R.

This is an extension of the $\mathcal{F}$-free orientation problem, as can be seen by taking $r = 2$ and R consisting of the two possible tournaments on 2 (labelled) vertices. We call this the $(\mathcal{F}, R)$-orientation problem, see Figure 1 for an example. One can view $\mathcal{F}$ as imposing *global* constraints (find a directed graph on V that globally avoids every tournament in $\mathcal{F}$), and R as imposing *local* constraints (find a directed graph on V that satisfies the local restrictions on the marked tuples). Our more general result is a P versus NP-complete complexity dichotomy for such problems.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 28; pp. 28:1–28:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

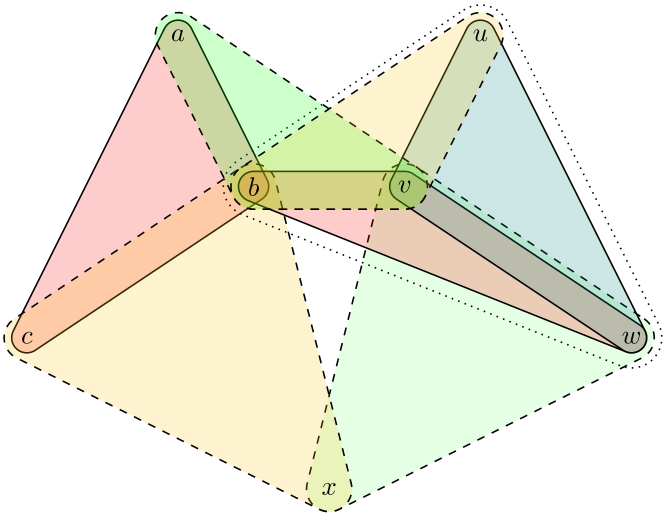


Figure 1 Example of an instance of $(\mathcal{F}, R)$-orientation where $\mathcal{F}$ consists of the transitive tournament on 4 vertices, and R contains the oriented graphs on 3 vertices inducing a transitive tournament. The markings are given by hyperedges. Note that due to the symmetry of R we do not need to specify the order of the vertices in the hyperedges. To solve such an instance, one needs to determine whether there exists an oriented graph on the given vertices, not inducing a transitive tournament on any four vertices, while all marked triples of vertices induce a transitive tournament.

▶ **Theorem 1.** *Let $\mathcal{F}$ be a finite set of finite tournaments and let R be a set of labelled r-vertex tournaments. Then the $(\mathcal{F}, R)$-orientation problem is solvable in polynomial time or NP-complete.*

1.2 Constraint Satisfaction Problems

Given a finite family $\mathcal{F}$ of tournaments, Fraïssé's theorem asserts that there exists a countably infinite digraph $D_{\mathcal{F}}$ whose finite subgraphs are exactly the oriented graphs avoiding every tournament in $\mathcal{F}$. Moreover, $D_{\mathcal{F}}$ satisfies a certain model-theoretic condition called *homogeneity*, which defines $D_{\mathcal{F}}$ uniquely up to isomorphism (see Section 2 for the detailed definitions). The following observation by Bodirsky and Guzmán-Pro in [5] is crucial for their complexity result. Consider the symmetric closure of $D_{\mathcal{F}}$, i.e., the undirected graph $H_{\mathcal{F}}$ obtained by forgetting about the directions of arcs in $D_{\mathcal{F}}$. Then it can be observed that given an undirected graph G, one has a homomorphism $G \to H_{\mathcal{F}}$ if, and only if, G admits an $\mathcal{F}$-free orientation. Thus, the $\mathcal{F}$-free orientation problem coincides with the *constraint satisfaction problem* for the graph $H_{\mathcal{F}}$, denoted by $\mathrm{CSP}(H_{\mathcal{F}})$. The complexity of this CSP can then be investigated using standard methods.

In order to obtain a generalization of [5] for the $(\mathcal{F}, R)$-orientation problem, we start with a similar observation. In $D_{\mathcal{F}}$, consider the subset $R' \subseteq V^r$ containing all tuples $(v_1, \ldots, v_r)$ inducing in $D_{\mathcal{F}}$ a tournament from R. The structure $\mathbb{A} = (V; R')$ is a so-called *first-order reduct* of $D_{\mathcal{F}}$. Note that if R consists of the two labelled tournaments on 2 vertices, then $\mathbb{A}$ thus defined coincides with $H_{\mathcal{F}}$. An input to the $(\mathcal{F}, R)$-orientation problem can then be seen as a structure in the same signature as $\mathbb{A}$. One obtains that an input to the $(\mathcal{F}, R)$-orientation problem admits a solution if, and only if, it admits a homomorphism to the structure $\mathbb{A}$. The structure $\mathbb{A}$ obtained in this way moreover satisfies two important properties:

- Provided R contains more than one tournament, the symmetric closure U of the edge relation of $D_{\mathcal{F}}$ is invariant under the higher-order symmetries of $\mathbb{A}$, called *polymorphisms*. In particular, the group of automorphisms of $\mathbb{A}$ is a subgroup of $\mathrm{Aut}(H_{\mathcal{F}})$.
- $\mathbb{A}$ has a particular type of binary injective polymorphism that we call an *injective projection*.

Theorem 1 then follows from the following technical result. The notion that "$\mathbb{A}$ pp-constructs every finite structure" appearing in the statement is defined in Section 2; for now, the reader can view this as a natural condition allowing to reduce the boolean satisfiability problem to the constraint satisfaction problem of $\mathbb{A}$, which is then NP-hard.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a finite set of tournaments, and let $\mathbb{A}$ be a first-order reduct of $D_{\mathcal{F}}$ that admits injective projections and such that U is invariant under* $\mathrm{Pol}(\mathbb{A})$. *One of the following holds:*

- $\mathbb{A}$ *pp-constructs every finite structure and* $\mathrm{CSP}(\mathbb{A})$ *is NP-complete, or*
- $\mathrm{CSP}(\mathbb{A})$ *is solvable in polynomial time.*

The proof follows the *smooth approximations* approach introduced in [25] and uses the refinements thereof from [22, 26]. This provides a streamlined proof and an extension of the dichotomy result from [5].

Another consequence of our proof is the following. Say that the class of $\mathcal{F}$-free orientable graphs has *bounded treewidth duality* if there exists a set $\mathcal{G}$ of undirected graphs of bounded treewidth such that for every finite graph G, there exists an $\mathcal{F}$-free orientation of G if, and only if, no graph from $\mathcal{G}$ admits a homomorphism to G. This notion can be extended to structures with an $(\mathcal{F}, R)$-orientation by generalizing the notion of treewidth for relational structures in general, we refer the interested reader to [15] for precise definitions. Any class of structures corresponding to a CSP and having bounded treewidth duality can be recognized in polynomial time by a Datalog program, giving a particularly simple algorithm recognizing the class.

As a by-product of our proof, we obtain a characterization of the sets $\mathcal{F}, R$ for which the class of structures with an $(\mathcal{F}, R)$-orientation has bounded treewidth duality. As above, this characterization is phrased in terms of the algebraic properties of a first-order reduct of $D_{\mathcal{F}}$. Using the recent results from [22], this also allows us to obtain a bounded on the treewidth in a possible duality depending on the size of the tournaments in $\mathcal{F}$ and R.

▶ **Theorem 3.** *Let $\mathcal{F}$ be a finite set of tournaments, and let $\mathbb{A}$ be a first-order reduct of $D_{\mathcal{F}}$ that admits injective projections and such that U is invariant under* $\mathrm{Pol}(\mathbb{A})$. *Let r be the maximal arity of a relation of $\mathbb{A}$ and ℓ be the maximal size of a tournament in $\mathcal{F}$. Then, the following are equivalent:*

1. $\mathrm{Pol}(\mathbb{A})$ *contains an* $\mathrm{Aut}(D_{\mathcal{F}})$*-canonical pseudo-majority polymorphism modulo* $\overline{\mathrm{Aut}(D_{\mathcal{F}})}$*,*
2. *The class of finite structures admitting a homomorphism to $\mathbb{A}$ has a duality of treewidth at most* $\max(6, r, \ell)$.

In particular, when the class of $\mathcal{F}$-free orientable graphs has a bounded treewidth duality, then Theorem 3 implies that the duality $\mathcal{G}$ can be chosen to consist of graphs of treewidth at most $\max(6, \ell)$.

1.3 Related Work

The approach we follow here shows an equivalence between $(\mathcal{F}, R)$-completion problems and a subclass of constraint satisfaction problems (CSPs). A P/NP-complete complexity dichotomy is known for CSPs with a finite template [14, 29], while for CSPs with infinite templates in general it is known that the complexity varies greatly [4, 17, 18]. For a subclass of infinite structures, so-called *first-order reducts of finitely bounded homogeneous structures*, Bodirsky and Pinsker have conjectured that a dichotomy similar to that of finite-domain CSPs exists. The templates $D_{\mathcal{F}}$ studied here are in the scope of that conjecture. The conjecture has been proved for a variety of finitely bounded homogeneous structures (e.g., all homogeneous undirected graphs [8], certain homogeneous hypergraphs [26], the universal homogeneous tournament [25], $(\mathbb{Q}; <)$ [7], and the universal homogeneous poset [20]). There exist natural subclasses of the Bodirsky-Pinsker class for which the conjecture is still open; this is the case for example for first-order reducts of finitely bounded homogeneous directed graphs, or for first-order reducts of *homomorphically bounded* homogeneous structures. It can be shown that if D is a homogeneous homomorphically bounded homogeneous directed graph, then $D = D_{\mathcal{F}}$ for some finite family $\mathcal{F}$ of finite tournaments. Thus, our result can be seen as making progress on the Bodirsky-Pinsker conjecture for both mentioned subclasses.

1.4 Organization of the paper

We recall some elementary notions from graph theory and the universal-algebraic approach to the complexity of infinite-domain CSPs in Section 2. In Section 3, we prove some elementary properties of the templates $\mathbb{A}$ arising from $(\mathcal{F}, R)$-completion problems. Due to space restrictions, we only describe a high-level proof strategy for Theorems 2 and 3 in Section 4, focusing on making the presentation accessible to a non-expert.

2 Definitions and Notations

2.1 Elementary model-theoretic notions

For the purposes of this paper, a *structure* is a tuple $\mathbb{A} = (A; R_1^{\mathbb{A}}, \dots, R_k^{\mathbb{A}})$ consisting of a set A (the *domain*) together with finitely many relations $R_i^{\mathbb{A}} \subseteq A^{r_i}$ on A. The *signature* of $\mathbb{A}$ is the list $(r_1, \dots, r_k)$ containing the arities of the relations of $\mathbb{A}$. We assume the reader is familiar with the standard notions of homomorphisms, embeddings, and isomorphisms between structures. As is standard in model theory, all substructures and subgraphs in this paper are *induced* substructures and subgraphs.

An oriented graph is a directed graph $G = (V, E)$ where at most one of $(u, v) \in E$ or $(v, u) \in E$ holds for all $u, v \in V$, and where $(u, u) \notin E$ for all $u \in V$. A tournament is an oriented graph such that the symmetric closure of its edge relation induces a complete graph. For a (finite) set $\mathcal{F}$ of finite tournaments we say that an oriented graph is $\mathcal{F}$-free if it contains no $F \in \mathcal{F}$ as an induced subgraph.

Let $\mathcal{F}$ be a finite set of finite tournaments. Let $\mathcal{C}_{\mathcal{F}}$ be the class of all $\mathcal{F}$-free oriented graphs. It can be seen that this class has the so-called *amalgamation property*: given two $\mathcal{F}$-free oriented graphs, their union is also $\mathcal{F}$-free due to $\mathcal{F}$ consisting of tournaments only. By the classical result of Fraïssé [16], there exists an oriented graph $D_{\mathcal{F}} = (V, E)$ on a countable set whose finite subgraphs are exactly the graphs isomorphic to a graph in $\mathcal{C}_{\mathcal{F}}$. Moreover, this graph is *homogeneous*, in the sense that for every finite subset $S \subseteq V$ and every partial isomorphism $f \colon S \to V$, there exists an automorphism α of $D_{\mathcal{F}}$ such that $f|_S = \alpha|_S$. These two properties describe $D_{\mathcal{F}}$ uniquely up to isomorphism. We write $H_{\mathcal{F}} = (V, U)$ for the undirected graph whose edge set is the symmetric closure of E. We note that $H_{\mathcal{F}}$ is in general not homogeneous, and in fact we will focus in this paper on the case where $H_{\mathcal{F}}$ is *not* homogeneous for reasons that are made clear below.

A *first-order reduct* of $\mathbb{A}$ is a structure $\mathbb{B}$ with the same domain as $\mathbb{A}$ and whose relations all have a first-order definition in $\mathbb{A}$. For example, $H_{\mathcal{F}}$ is a first-order reduct of $D_{\mathcal{F}}$, because the symmetric closure U of E is definable by $(v, w) \in E \vee (w, v) \in E$, which is a first-order definition of U in $D_{\mathcal{F}}$. An *expansion* of $\mathbb{A}$ is a structure $\mathbb{B}$ obtained from $\mathbb{A}$ by adding new relations.

An example of a natural expansion of $D_{\mathcal{F}}$ is obtained as follows. Consider the class $\mathcal{C}_{\mathcal{F}}^{<}$ of all structures $(V', E', \prec)$ where $(V', E') \in \mathcal{C}_{\mathcal{F}}$ and where $\prec$ is an arbitrary linear order on V'. This class also has the amalgamation property as described above, and its Fraïssé limit can be taken to be $(V, E, <)$, an expansion of $D_{\mathcal{F}}$ by a linear order.

Let $\mathscr{G}$ be a group of permutations on a set A. Let $\equiv_k$ be the equivalence relation on A^k containing the pairs (a, b) of tuples such that there exists $\alpha \in \mathscr{G}$ such that $\alpha(a_i) = b_i$ for all $i \in \{1, \ldots, k\}$. We call the equivalence classes of $\equiv_k$ the *orbits* of $\mathscr{G}$. The orbit of a pair $(a, b) \in V^2$ under $\mathrm{Aut}(D_{\mathcal{F}})$ is given, by the homogeneity of $D_{\mathcal{F}}$, by the isomorphism type of the labelled graph induced by $\{a, b\}$. There are therefore 4 orbits, corresponding to whether $a = b$, whether $a \neq b$ are connected by an edge, and if they are whether $(a, b) \in E$ or $(b, a) \in E$. We denote the orbit containing the pairs (a, b) such that $a \neq b$ and $(a, b) \notin U$ by N, the orbit of all $(a, b) \in E$ by $\rightarrow$, and the orbit of all (a, b) such that $(b, a) \in E$ by $\leftarrow$.

Let $\mathbb{A}, \mathbb{B}$ be structures with $B = A^n$. We say $\mathbb{B}$ is a *pp-power* of A if every relation in $\mathbb{B}$ is definable by a primitive positive formula over $\mathbb{A}$, that is a formula only using $\exists$ and $\wedge$. For arbitrary structures $\mathbb{A}, \mathbb{B}$ we say $\mathbb{A}$ *pp-constructs* $\mathbb{B}$ if there is a pp-power $\mathbb{C}$ of $\mathbb{A}$ such that $\mathbb{B}$ and $\mathbb{C}$ are homomorphically equivalent.

2.2 Clones, naked and affine sets

A relation $R \subseteq A^m$ is said to be *invariant* under a function $f\colon A^n \to A$ if for every $a^1, \ldots, a^n \in R$, then $f(a^1, \ldots, a^n)$, the tuple obtained by applying f componentwise to the tuples $a^1, \ldots, a^n$, is also in R. A function $f\colon A^n \to A$ is a *polymorphism* of a structure $\mathbb{A}$ if all the relations of $\mathbb{A}$ are invariant under f. In particular, every automorphism and endomorphism of $\mathbb{A}$ is a polymorphism of $\mathbb{A}$. The set $\mathrm{Pol}(\mathbb{A})$ of all polymorphisms of $\mathbb{A}$ forms a *clone*: it contains all the projections $p_i^k\colon (a_1, \ldots, a_k) \mapsto a_i$ for $1 \leq i \leq k$ and it is closed under composition. We write $\mathscr{P}$ for the clone consisting of only the projections on the set $\{0, 1\}$. This clone is relevant in the theory of constraint satisfaction because it is exactly the clone of polymorphisms of a structure $\mathbb{S}$ with domain $\{0, 1\}$ and having all ternary relations on $\{0, 1\}$ as its relations, whose CSP corresponds to the problem CNF-3SAT.

If $S \subseteq A^k$ is invariant under a clone $\mathscr{C}$, and θ is an equivalence relation on S that is also invariant under $\mathscr{C}$, then the operations in $\mathscr{C}$ naturally induce a clone of functions on the set S/θ, where $f \in \mathscr{C}$ of arity n induces the function $([s_1], \ldots, [s_n]) \mapsto [f(s_1, \ldots, s_n)]$. We use the notation $\mathscr{C} \curvearrowright S/\theta$ to denote this clone and if θ has at least two equivalence classes we call (S, θ) a *subfactor* of $\mathscr{C}$. A subfactor is *minimal* if for all $a, b \in S$ that are not θ-equivalent, the smallest $\mathscr{C}$-invariant set containing a, b is equal to S.

A subfactor (S, θ) of $\mathscr{C}$ is called a *naked set* if $\mathscr{C} \curvearrowright S/\theta$ only consists of projections. Similarly, if S/θ is finite and can be endowed with a structure of a module over a ring R, in a way that $\mathscr{C} \curvearrowright S/\theta$ consists of *affine* functions, of the form $(x_1, \ldots, x_n) \mapsto \sum \lambda_i \cdot x_i$ for $\lambda_1, \ldots, \lambda_n \in R$ such that $\sum \lambda_i = 1$, then we call (S, θ) an *affine set* for $\mathscr{C}$. Note that every naked set is a particular example of an affine set: if (S, θ) is a naked set of $\mathscr{C}$, then the maps induced by $\mathscr{C}$ on S/θ have exactly one non-zero λ_i, which is equal to 1.

For a clone $\mathscr{C}$ on a 2-element set (in this paper, the 2-element set is $\{\leftarrow, \rightarrow\}$), the notions of having a naked set or an affine set can be rephrased using Post's classification of such clones [28]:

- $\mathscr{C}$ has a naked set if, and only if, every operation in $\mathscr{C}$ is essentially unary, of the form $(x_1, \dots, x_n) \mapsto e(x_i)$ for some permutation e of $\{\leftarrow, \rightarrow\}$ and $i \in \{1, \dots, n\}$. If $\mathscr{C}$ does not have a naked set, then it contains an operation f satisfying the equation

  $$f(x, y, z) = f(y, z, x)$$

  for all $x, y, z \in \{\leftarrow, \rightarrow\}$. Such an f is called a *ternary cyclic operation.*
- $\mathscr{C}$ has an affine set if, and only if, every operation in $\mathscr{C}$ is of the form

  $$(x_1, \dots, x_n) \mapsto \sum \lambda_i \cdot x_i + \mu$$

  where $\lambda_1, \dots, \lambda_n, \mu \in \{0, 1\}$ are such that $\sum \lambda_i = 1$, addition and multiplication are understood modulo 2, and after choosing an arbitrary bijection between $\{\leftarrow, \rightarrow\}$ and $\{0, 1\}$. If $\mathscr{C}$ does not have an affine set, then it contains an operation m satisfying the identities

  $$m(x, x, y) = m(x, y, x) = m(y, x, x) = x$$

  for all $x, y \in \{\leftarrow, \rightarrow\}$ or a binary operation s that is associative, commutative, and satisfies $s(x, x) = x$ for all $x \in \{\leftarrow, \rightarrow\}$. Such an m is called a *majority* operation, while s is called a *semilattice.*

For an ω-categorical structure $\mathbb{A}$ it is known that the complexity of CSP($\mathbb{A}$) is captured by the polymorphisms of $\mathbb{A}$. For our purposes, we need the following two special cases:

▶ **Theorem 4** ([21]). *Let $\mathbb{A}$ be an ω-categorical structure. The following hold:*

- *If* Pol($\mathbb{A}$) *has a naked set, then $\mathbb{A}$ pp-constructs every finite structure and* CSP($\mathbb{A}$) *is NP-hard.*
- *If* Pol($\mathbb{A}$) *has an affine set, then the class of structures admitting a homomorphism to $\mathbb{A}$ does not have bounded treewidth duality.*

2.3 Canonical Functions

Let $S \subseteq A^k$ be invariant under a function $f\colon A^n \to A$. We say that $f\colon A^n \to A$ is *canonical on S with respect to a permutation group $\mathscr{G}$* if whenever $a^1 \equiv_k b^1, \dots, a^n \equiv_k b^n$ for $a^1, b^1, \dots, a^n, b^n \in S$, then $f(a^1, \dots, a^n) \equiv_k f(b^1, \dots, b^n)$. In other words, f is canonical on S if the restriction of $\equiv_k$ to S is invariant under f; in particular, the set of functions f that are canonical on S with respect to $\mathscr{G}$ forms a clone $\mathscr{C}$, and this clone has an action $\mathscr{C} \curvearrowright S/{\equiv_k}$. If $f(a^1, \dots, a^n) \equiv_k f(\alpha a^1, \dots, \alpha a^n)$ holds for all $\alpha \in \mathrm{Aut}(\mathbb{A})$ and all $a^1, \dots, a^n$, we call f *diagonally* canonical. In order to simplify notation and to make the dependence on the group clearer, we write $S/\mathscr{G}$ for the set of orbits of S induced by $\mathscr{G}$.

Let $k \geq 1$. We write $I_k \subseteq V^k$ for the set of tuples with pairwise distinct entries, and $K \subseteq I_k$ for the set of tuples whose components induce a clique in $H_{\mathcal{F}}$. We will be considering the following clones:

- $\mathscr{C}^K_{(D_{\mathcal{F}},<)}$ is the clone of polymorphisms of $\mathbb{A}$ that are canonical on K with respect to $\mathrm{Aut}(D_{\mathcal{F}}, <)$.
- $\mathscr{C}^K_{D_{\mathcal{F}}}$ the clone of polymorphisms of $\mathbb{A}$ that are canonical on K with respect to $\mathrm{Aut}(D_{\mathcal{F}})$,
- $\mathscr{C}^I_{\mathbb{A}}$ the clone of polymorphisms of $\mathbb{A}$ that are, for all $k \geq 1$, canonical on I_k with respect to $\mathrm{Aut}(\mathbb{A})$.

Note that since canonicity is considered with respect to different groups, the clones above are not necessarily comparable with respect to inclusion.

2.4 Smooth approximations

Central to our proof of Theorem 2 is the theory of smooth approximations, further developed in [23]. It relies on comparing two clones $\mathscr{C} \subseteq \mathscr{D}$ and whether a naked set (resp. affine set) for $\mathscr{C}$ can be lifted to a naked set (resp. affine set) for $\mathscr{D}$. The lifting is formalized by smooth approximations.

▶ **Definition 5** (Smooth Approximations). *Let A be a set, $k \geq 1$, and $\sim$ be an equivalence relation on $S \subseteq A^k$. An equivalence relation η on some set S' with $S \subseteq S' \subseteq A^k$ approximates $\sim$ if the restriction of η to S is a possible (non-proper) refinement of $\sim$; in that case, η is an* approximation *of $\sim$.*

Suppose that the $\sim$-equivalence classes as well as η are invariant under a group $\mathscr{G}$ of permutations on A. We say that the approximation is

- presmooth *with respect to a group $\mathscr{G}$ if each equivalence class C of $\sim$ intersects some equivalence class C' of η such that $C \cap C'$ contains two disjoint tuples in the same $\mathscr{G}$-orbit;*
- very smooth *with respect to a group $\mathscr{G}$ if $\equiv_k$ is a (possibly non-proper) refinement of η; in other words, if any two k-tuples in the same orbit must be η-equivalent.*

The equivalence relations $\sim$ for which we want to find approximations come from subfactors of $\mathscr{C}$. If $\mathscr{D}$ contains $\mathscr{C}$, it might not act at all on $S/{\sim}$ (if S or $\sim$ is not invariant under $\mathscr{D}$), and even in the case that it does, its action might contain operations that are not from $\mathscr{C} \curvearrowright S/{\sim}$. However, the theory of smooth approximations gives us that we can (under certain conditions) find a $\mathscr{D}$-invariant set $S' \supseteq S$ and an equivalence relation η on S' that approximates $\sim$, and such that $\mathscr{D} \curvearrowright S'/\eta$ is not "richer" than $\mathscr{C} \curvearrowright S/{\sim}$.

One of the central results from [25] is the so-called *loop lemma of smooth approximations*. We do not give the general formulation of the loop lemma here and rather phrase it directly the way we apply it in our proof.

▶ **Theorem 6** (Consequence of Theorem 11 and Lemma 14 in [25]). *Let $k \geq 1$, and suppose that $\mathscr{C}^I_{\mathbb{A}} \curvearrowright I_k/\mathrm{Aut}(\mathbb{A})$ has a naked (resp. affine) set. Then there exists a naked (resp. affine) set $(S, \sim)$ of $\mathscr{C}^I_{\mathbb{A}}$ with $S \subseteq I_k$ and $\mathrm{Aut}(\mathbb{A})$-invariant $\sim$-classes such that one of the following holds:*

1. *$\sim$ is approximated by a $\mathrm{Pol}(\mathbb{A})$-invariant equivalence relation that is presmooth with respect to $\mathrm{Aut}(D_{\mathcal{F}})$;*
2. *there exists $f \in \mathrm{Pol}(\mathbb{A})$ such that $f(a,b) \sim f(b,a)$ holds for all disjoint injective tuples $a, b \in V^k$ such that $f(a,b), f(b,a) \in S$.*

▶ **Lemma 7.** *Let $f\colon V^n \to V$ be an arbitrary operation defined on V. There exists $g\colon V^n \to V$ that is canonical with respect to $\mathrm{Aut}(D_{\mathcal{F}}, <)$ and that is* locally interpolated *by f, i.e., for every finite $S \subseteq V$ there exist $\alpha_1, \ldots, \alpha_n, \beta \in \mathrm{Aut}(D_{\mathcal{F}}, <)$ such that $g(a_1, \ldots, a_n) = \beta f(\alpha_1 a_1, \ldots, \alpha_n a_n)$ holds for all $a_1, \ldots, a_n \in S$.*

Proof. By the theorem of Nešetřil and Rödl [27], the class of all $\mathcal{F}$-free oriented graphs endowed with a linear order is a so-called *Ramsey class*. The conclusion is then obtained by applying [13, Lemma 14], see also [12, Theorem 5] for an alternative presentation. ◀

In particular, if $\mathbb{A}$ is a first-order reduct of $D_{\mathcal{F}}$, then for every $f \in \mathrm{Pol}(\mathbb{A})$ there exists $g \in \mathscr{C}^K_{(D_{\mathcal{F}},<)}$ that is locally interpolated by f. Similarly, for every $f\colon V^n \to V$, there exists $g\colon V^n \to V$ that is diagonally canonical with respect to $\mathrm{Aut}(D_{\mathcal{F}}, <)$ and that is *diagonally interpolated* by f, that is, for every finite $S \subseteq V$ there exist $\alpha, \beta \in \mathrm{Aut}(D_{\mathcal{F}}, <)$ such that $g(a_1, \ldots, a_n) = \beta f(\alpha a_1, \ldots, \alpha a_n)$ holds for all $a_1, \ldots, a_n \in S$.

3 Injective Projections and Preliminary Results

We define here the notion of injective projections appearing in Theorem 2.

▶ **Definition 8** (Injective projections). *An* injective projection *is a function $q_i^n \colon V^n \to V$ such that for all $a_1, \ldots, a_n \in V^2$, at least one of which is not a constant pair, the following hold:*

- *$q_i^n(a_1, \ldots, a_n) \in \mathrm{N}$ if $a_j \notin U$ for some $j \in \{1, \ldots, n\}$,*
- *$q_i^n(a_1, \ldots, a_n)$ and a_i are in the same orbit under $\mathrm{Aut}(D_{\mathcal{F}})$ otherwise.*

If $\mathbb{A}$ is a reduct of $D_{\mathcal{F}}$ and $q_i^n \in \mathscr{C}_{\mathbb{A}}^I$ for all $1 \leq i \leq n$ then we say $\mathbb{A}$ has injective projections.

Note that injective projections are canonical with respect to $D_{\mathcal{F}}$. Indeed, suppose that $a_1, b_1, \ldots, a_n, b_n$ are tuples of the same length m such that a_j and b_j induce the same graph in $D_{\mathcal{F}}$, for all $j \in \{1, \ldots, n\}$. Then by definition, $q_i^n(a_1, \ldots, a_n)$ and $q_i^n(b_1, \ldots, b_n)$ induce the same directed graph in $D_{\mathcal{F}}$, and thus they are in the same orbit under $\mathrm{Aut}(D_{\mathcal{F}})$ by homogeneity of $D_{\mathcal{F}}$.

▶ **Proposition 9.** *Let $\mathcal{F}$ be a finite set of finite tournaments, and let $\mathbb{A}$ be a first-order reduct of $D_{\mathcal{F}}$ whose relations only contain tuples inducing tournaments in $D_{\mathcal{F}}$. Then $q_i^n \in \mathrm{Pol}(\mathbb{A})$ for all $1 \leq i \leq n$. If one of the relations of $\mathbb{A}$ contains tuples inducing different tournaments, then U is invariant under $\mathrm{Pol}(\mathbb{A})$.*

Proof. Let R be a relation of $\mathbb{A}$ that contains two tuples inducing different tournaments in $D_{\mathcal{F}}$. In particular the arity n of R must be at least 2. Since R contains tuples a, b inducing different tournaments, there exist distinct $i, j \in \{1, \ldots, n\}$ such that (a_i, a_j) and (b_i, b_j) induce different tournaments, i.e., they form edges in opposite directions. Thus, the projection of R onto the coordinates i, j is equal to U, and it follows that U is invariant under $\mathrm{Pol}(\mathbb{A})$.

Let $1 \leq i \leq n$. We define a directed graph $D = (V^n, E')$ by $(x, y) \in E'$ if, and only if, $(x_j, y_j) \in U$ for all $j \in \{1, \ldots, n\}$ and $(x_i, y_i) \in E$.

We prove that D is $\mathcal{F}$-free. Assume for contradiction otherwise. Then there is a finite $V' \subseteq V^n$ inducing a tournament from $\mathcal{F}$ in D. By definition, the projection

$$\{v \in V \mid \exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \in V : (x_1, \ldots, x_{i-1}, v, x_{i+1}, \ldots, x_n) \in V'\}$$

of V' onto its ith coordinate induces the same tournament in $D_{\mathcal{F}}$. But this contradicts the fact that $D_{\mathcal{F}}$ is $\mathcal{F}$-free. Thus, since $D_{\mathcal{F}}$ is universal for the class of $\mathcal{F}$-free digraphs, there exists an embedding $q_i^n \colon D \hookrightarrow D_{\mathcal{F}}$.

We can then view q_i^n as an n-ary function on V. We prove that it is a polymorphism of $\mathbb{A}$. Let R be a relation of $\mathbb{A}$, and let $r_1, \ldots, r_n \in R$. By assumption on R, all $r_1, \ldots, r_n$ induce tournaments in $D_{\mathcal{F}}$. Thus, $q_i^n(r_1, \ldots, r_n)$ induce the same tournament as r_i, and therefore $q_i^n(r_1, \ldots, r_n) \in R$. ◀

As a consequence of Proposition 9, we obtain that all templates $\mathbb{A}$ arising from a $(\mathcal{F}, R)$-completion problem in the way outlined in the introduction satisfy the assumptions of Theorem 2.

We now prove that the injective projections are canonical with respect to $\mathrm{Aut}(\mathbb{A})$, for any first-order reduct $\mathbb{A}$ of $D_{\mathcal{F}}$ such that $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_{\mathcal{F}})$. The proof of this fact is the only place where the classification of first-order reducts of $D_{\mathcal{F}}$ due to Agarwal and Kompatscher [1] is needed.

▶ **Lemma 10.** *Let $\mathbb{A}$ be a first-order reduct of $D_{\mathcal{F}}$ with $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_{\mathcal{F}})$. The injective projections are canonical with respect to $\mathrm{Aut}(\mathbb{A})$ and are therefore elements of $\mathscr{C}_{\mathbb{A}}^I$.*

Proof. Let $1 \leq k \leq n$ and let $a^1, b^1, \ldots, a^n, b^n$ be such that a^i and b^i are injective tuples that are in the same orbit under $\mathrm{Aut}(\mathbb{A})$, for all $i \in \{1, \ldots, n\}$. We need to show that $q_k^n(a^1, \ldots, a^n)$ and $q_k^n(b^1, \ldots, b^n)$ are in the same orbit under $\mathrm{Aut}(\mathbb{A})$.

The $\mathrm{Aut}(D_{\mathcal{F}})$-orbit of $q_k^n(a^1, \ldots, a^n)$ is the orbit of n-tuples c such that for all $i \neq j$, either $(c_i, c_j) \in \mathrm{N}$ whenever for some ℓ, $(a_i^\ell, a_j^\ell) \in \mathrm{N}$, and otherwise (c_i, c_j) is in the same $\mathrm{Aut}(D_{\mathcal{F}})$-orbit as (a_i^k, a_j^k). Since a^ℓ and b^ℓ are in the same orbit under $\mathrm{Aut}(\mathbb{A})$ for all $\ell \in \{1, \ldots, n\}$, for all $i \neq j$ we have $(a_i^\ell, a_j^\ell) \in \mathrm{N}$ if, and only if, $(b_i^\ell, b_j^\ell) \in \mathrm{N}$. Therefore, the indices i, j where the restrictions of the tuples $q_k^n(a^1, \ldots, a^n)$ and $q_k^n(b^1, \ldots, b^n)$ are in N coincide.

By the classification of all first-order reducts $\mathbb{A}$ of $D_{\mathcal{F}}$ such that $\mathrm{Aut}(D_{\mathcal{F}}) \leq \mathrm{Aut}(\mathbb{A}) \leq \mathrm{Aut}(H_{\mathcal{F}})$ due to Agarwal and Kompatscher [1], the tuple b^k can be obtained from a^k by a sequence of *switching* steps and *reversing* steps defined as follows. Given a directed graph, a switching step consists in choosing a vertex of the graph and reversing the direction of every edge incident to that vertex; a reversing step consists in reversing the direction of all edges. Note that if the directed graph induced by b can be obtained by finitely many such operations starting from the directed graph induced by a, then the same is true if one removes in a and b edges at the same position. It follows that $q_k^n(a^1, \ldots, a^k)$ and $q_k^n(b^1, \ldots, b^k)$ are in the same orbit under $\mathrm{Aut}(\mathbb{A})$. ◀

4 Description of the proof strategy

We describe here the strategy for the proof of Theorem 2 on a relatively high level.

4.1 Preprocessing of the Reducts of $D_{\mathcal{F}}$

A structure $\mathbb{A}$ is a *model-complete core* if for every finite $S \subseteq A$ and every endomorphism $f\colon \mathbb{A} \to \mathbb{A}$, there exists an automorphism $\alpha \in \mathrm{Aut}(\mathbb{A})$ such that $f|_S = \alpha|_S$. It is often very convenient for studying the complexity of $\mathrm{CSP}(\mathbb{A})$ and the polymorphisms of $\mathbb{A}$ to work with a structure that is a model-complete core; as an example of an important application for us, if $\mathbb{A}$ is a model-complete core, then for all $n \geq 1$ and $a \in A^n$, the orbit of a under $\mathrm{Aut}(\mathbb{A})$ is invariant under all the polymorphisms of $\mathbb{A}$.

While not every structure is a core, it is known that every ω-categorical structure, and in particular every first-order reduct $\mathbb{A}$ of $D_{\mathcal{F}}$, is homomorphically equivalent to a structure $\mathbb{B}$ that is a model-complete core, i.e., such that there exist homomorphisms $\mathbb{A} \to \mathbb{B}$ and $\mathbb{B} \to \mathbb{A}$ [3, 2]. Moreover, this structure is unique up to isomorphism and is called the *model-complete core of* $\mathbb{A}$.

If $\mathbb{A}$ is a first-order reduct of $D_{\mathcal{F}}$, it is *a priori* not guaranteed that the model-complete core of $\mathbb{A}$ is a first-order reduct of $D_{\mathcal{F}}$. The following statement that we prove first establishes this property and relies on a recent result by Mottet and Pinsker [24].

▶ **Lemma 11.** *Let $\mathbb{A}$ be a first-order reduct of $D_{\mathcal{F}}$, and let $\mathbb{A}'$ be the model-complete core of $\mathbb{A}$. Then $\mathbb{A}'$ is either isomorphic to $\mathbb{A}$, or a 1-element structure, or a first-order reduct of a homogeneous undirected graph or the universal homogeneous tournament.*

In the following statement, a function $g\colon A \to A$ with respect to a group $\mathscr{G}$ of permutations on A is *range-rigid* if for every $\alpha \in \mathscr{G}$ and every finite $S \subseteq A$, there exists $\beta \in \mathscr{G}$ such that $g|_S = \beta \circ g \circ \alpha \circ g|_S$. In words, this means that g essentially behaves like a retraction (modulo elements of $\mathscr{G}$) on every orbit of $\mathscr{G}$ that intersects the range of g.

While the first outcome in Theorem 6 only yields a presmooth approximation, it is a very smooth approximation that is needed in Proposition 21. As in [25], the step from presmooth to very smooth is achieved by leveraging a certain primitivity property of the automorphism group of the base structure under consideration, in this case $D_{\mathcal{F}}$.

For $n \geq 1$, we call a permutation group $\mathscr{G}$ acting on a set A *n-primitive* if for every orbit $O \subseteq A^n$ of $\mathscr{G}$, every $\mathscr{G}$-invariant equivalence relation on O containing a pair (a, b) where a, b are disjoint is full. We say an ω-categorical structure $\mathbb{A}$ has *no algebraicity* if none of its elements are first-order definable using other elements as parameters.

▶ **Lemma 12.** *For all $n \geq 1$,* $\mathrm{Aut}(D_{\mathcal{F}})$ *is n-primitive and has no algebraicity.*

Proof. Let $n \geq 1$ and O an orbit of n-tuples under $\mathrm{Aut}(D_{\mathcal{F}})$ and $a \sim b$ for some equivalence relation $\sim$ on O with a, b disjoint, and c, d arbitrary tuples in O. We define a digraph $\mathbb{X}$ on $5n$ vertices, partitioned into five n-tuples $x, y, z, u, v \in V^n$ such that the entries of x, u and v, z induce the same graph as the entries of a, b in $D_{\mathcal{F}}$. Similar let u, y and y, v induce b, a. Finally let x, z induce c, d. Then $\mathbb{X}$ is $\mathcal{F}$-free as all induced tournaments in $\mathbb{X}$ contain only vertices belonging to at most two tuples which are neighbors in the cycle x, u, y, v, z, x. By definition these two tuples induce a graph isomorphic to the graph induced by a, b or c, d in $D_{\mathcal{F}}$ respectively, so $\mathbb{X}$ is $\mathcal{F}$-free. Then there is an embedding $f \colon \mathbb{X} \to D_{\mathcal{F}}$ with $f(x) = c$ and $f(z) = d$ by homogeneity. Also by transitivity of $\sim$ we have $c = f(x) \sim f(z) = d$.

The class $\mathcal{C}$ of $\mathcal{F}$-free oriented graphs has the *free amalgamation property*, i.e., given any two such oriented graphs $D = (W, F), D' = (W', F')$ inducing the same directed graph on $W \cap W'$, then the union $(W \cup W', F \cup F')$ is an $\mathcal{F}$-free oriented graph. This implies (see, e.g., [19]) that the Fraïssé limit of $\mathcal{C}$, which is exactly $D_{\mathcal{F}}$, has no algebraicity. ◀

▶ **Lemma 13.** *Let $\mathbb{A}$ be a first-order reduct of $D_{\mathcal{F}}$ that is a model-complete core and such that* $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_{\mathcal{F}})$*. Then $\neq$ and* N *are invariant under* $\mathrm{Pol}(\mathbb{A})$*.*

Proof. Since $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_{\mathcal{F}})$, N is a single orbit under $\mathrm{Aut}(\mathbb{A})$ and is therefore invariant under $\mathrm{Pol}(\mathbb{A})$ since $\mathbb{A}$ is a model-complete core. Now, let O be another orbit of injective pairs. Then every pair (a, b) with $a \neq b$ satisfied the formula $\varphi(x, y) := \exists z((x, z) \in O \wedge (y, z) \in \mathrm{N})$. This is a primitive positive formula, and since both O and N are invariant under $\mathrm{Pol}(\mathbb{A})$, then so is the relation defined by φ. ◀

Let us call a first-order reduct $\mathbb{A}$ of $D_{\mathcal{F}}$ a *proper reduct* if the following conditions are satisfied:

- $\mathbb{A}$ is a model-complete core, i.e., if every homomorphism $\mathbb{A} \to \mathbb{A}$ locally resembles an automorphism an automorphism of $\mathbb{A}$ (the precise definition is given in Section 4.1),
- $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_{\mathcal{F}})$,
- if $\mathrm{Aut}(\mathbb{A}) = \mathrm{Aut}(H_{\mathcal{F}})$, then $H_{\mathcal{F}}$ is not homogeneous.

Our next step is to show that $\mathbb{A}$ can without loss of generality be assumed to be proper. Indeed, if $\mathbb{A}$ is not a model-complete core, then by Lemma 11 the model-complete core of $\mathbb{A}$ is a 1-element structure, or is a first-order reduct of a homogeneous graph, or is a first-order reduct of the universal homogeneous tournament; Theorem 2 is known to hold for all such structures [8, 25]. Since replacing $\mathbb{A}$ by its model-complete core does not change the outcome of Theorem 2, we are immediately done if $\mathbb{A}$ is not a model-complete core. If $\mathrm{Aut}(\mathbb{A}) \not\subseteq \mathrm{Aut}(H_{\mathcal{F}})$, then by Theorem 2.2(i) of [1] we have $\mathrm{Aut}(H_{\mathcal{F}}) \subsetneq \mathrm{Aut}(\mathbb{A})$, in which case either $H_{\mathcal{F}}$ is a homogeneous undirected graph, or $H_{\mathcal{F}}$ is not homogeneous and then

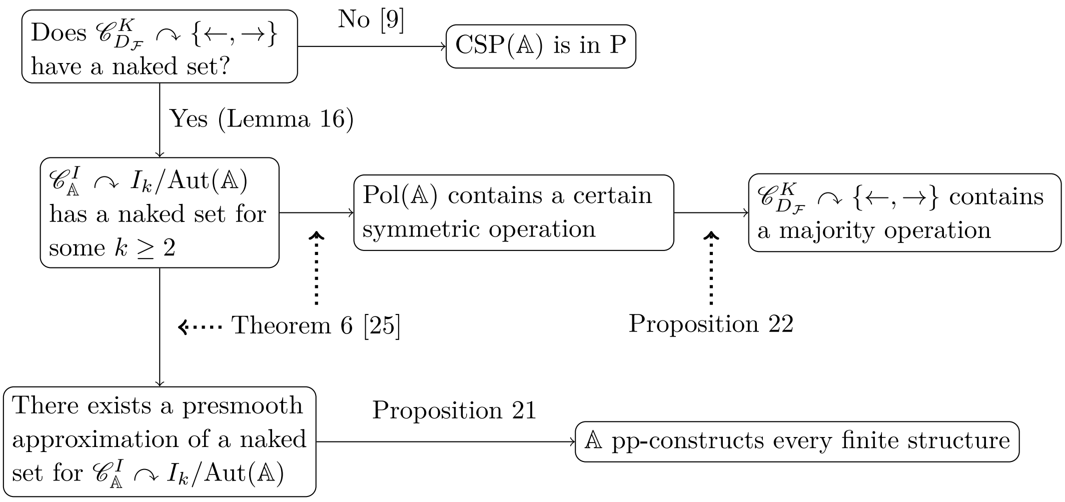

Does $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ have a naked set?
No [9]
CSP($\mathbb{A}$) is in P
Yes (Lemma 16)
$\mathscr{C}^I_\mathbb{A} \curvearrowright I_k/\mathrm{Aut}(\mathbb{A})$ has a naked set for some $k \geq 2$
Pol($\mathbb{A}$) contains a certain symmetric operation
$\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ contains a majority operation
Theorem 6 [25]
Proposition 22
There exists a presmooth approximation of a naked set for $\mathscr{C}^I_\mathbb{A} \curvearrowright I_k/\mathrm{Aut}(\mathbb{A})$
Proposition 21
$\mathbb{A}$ pp-constructs every finite structure

Figure 2 A simplified overview of the proof strategy of Theorem 2 after the preprocessing step.

$\mathrm{Aut}(\mathbb{A})$ is the full symmetric group by Theorem 2.2(iii) of [1]. Both of these cases can be handled by [25].[1] Finally, if $\mathrm{Aut}(\mathbb{A}) = \mathrm{Aut}(H_\mathcal{F})$ and $H_\mathcal{F}$ is homogeneous, then we are again done by [25].

4.2 An Algebraic Dichotomy for Proper Reducts

After this "preprocessing" step, the main technical result is the following.

▶ **Theorem 14.** *Let $\mathcal{F}$ be a finite set of finite tournaments. Let $\mathbb{A}$ be a proper reduct of $D_\mathcal{F}$ that admits injective projections and such that U is invariant under* Pol($\mathbb{A}$)*. Then exactly one of the following holds:*

1. Pol($\mathbb{A}$) *has a naked set,*
2. Pol($\mathbb{A}$) *contains a ternary operation f that is canonical with respect to* $\mathrm{Aut}(D_\mathcal{F})$ *and $u, v \in \overline{\mathrm{Aut}(D_\mathcal{F})}$ such that $u \circ f(x,y,z) = v \circ f(y,z,x)$ holds for all $x,y,z \in V$.*

Note that Theorem 14 is indeed a refined version of Theorem 2: the first item of Theorem 14 implies the first item of Theorem 2 by [11], and the second item of Theorem 14 implies the second item of Theorem 2 by [9].

The proof strategy is represented in Figure 2 and is based on distinguish upon whether $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$, which is a clone of functions on the two-element set $\{\leftarrow,\rightarrow\}$, has a naked set. If $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ does not have a naked set, then we show that Pol($\mathbb{A}$) contains an operation f as in the second item of Theorem 14.

▶ **Proposition 15.** *Let $\mathbb{A}$ be a proper reduct of $D_\mathcal{F}$ that admits injective projections. The following hold:*

1. *If $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ does not have a naked set, then there exists $f \in$* Pol($\mathbb{A}$) *and $u, v \in \overline{\mathrm{Aut}(D_\mathcal{F})}$, canonical with respect to* $\mathrm{Aut}(D_\mathcal{F})$*, and such that the identity*

$$u \circ f(x,y,z) = v \circ f(y,z,x)$$

holds for all $x,y,z \in V$;

[1] The first proof in the first case was given in [8], while the first proof in the case of the full symmetric group is due to [6].

2. *If* $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ *does not have an affine set, then there exists* $f \in \mathrm{Pol}(\mathbb{A})$ *such that for all* $n \geq 1$ *and* $a, b \in V^n$*, the tuples* $a, f(a,a,b), f(a,b,a), f(b,a,a)$ *are all in the same orbit under* $\mathrm{Aut}(D_\mathcal{F})$*.*

Proof. If $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ does not have a naked set, then it contains a ternary function f' that acts as a cyclic operation on $\{\leftarrow, \rightarrow\}$. Then $f(x,y,z) := f'(q^3_1(x,y,z), q^3_1(y,z,x), q^3_1(z,x,y))$ is a canonical polymorphism of $\mathbb{A}$ which is pseudo-cyclic modulo $\overline{\mathrm{Aut}(D_\mathcal{F})}$. If $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ does not have an affine set, then it contains a ternary function m that induces a majority operation on $\{\leftarrow, \rightarrow\}$ (note that any binary operation $f \in \mathscr{C}^K_{D_\mathcal{F}}$ induces on $\{\leftarrow, \rightarrow\}$ a function such that $f(\rightarrow, \leftarrow) \neq f(\leftarrow, \rightarrow)$, so that $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ cannot contain a semilattice operation). Then $f(x,y,z) := m(q^3_1(x,y,z), q^3_1(y,z,x), q^3_1(z,x,y))$ is a canonical polymorphism of $\mathbb{A}$ that satisfies the statement. ◀

Then CSP($\mathbb{A}$) can be solved in polynomial time by reducing it to a tractable finite-domain CSP [10]. Otherwise, $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ has a naked set and one can prove in this case that $\mathscr{C}^I_\mathbb{A} \curvearrowright I_k/\mathrm{Aut}(\mathbb{A})$ also has a naked set for some $k \geq 1$.

▶ **Lemma 16.** *Let* $\mathbb{A}$ *be a proper reduct of* $D_\mathcal{F}$ *that admits injective projections and such that* U *is invariant under* $\mathrm{Pol}(\mathbb{A})$*. Assume* $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ *has a naked (resp. affine) set. Then* $\mathscr{C}^I_\mathbb{A} \curvearrowright I_k/\mathrm{Aut}(\mathbb{A})$ *has a naked (resp. affine) set for some* $k \geq 2$*.*

In other words, there exist $S \subseteq I_k$ invariant under $\mathscr{C}^I_\mathbb{A}$ and a $\mathscr{C}^I_\mathbb{A}$-invariant equivalence relation $\sim$ on S with $\mathrm{Aut}(\mathbb{A})$-invariant equivalence classes such that $\mathscr{C}^I_\mathbb{A} \curvearrowright S/{\sim}$ only contains projections. The loop lemma of smooth approximations applies (Theorem 6), giving us two possible outcomes.

4.2.1 First Case: Presmooth Approximation

In the first case, there exists a presmooth approximation for a naked set of $\mathscr{C}^I_\mathbb{A} \curvearrowright I/\mathrm{Aut}(\mathbb{A})$. We first show how to "upgrade" this approximation into a very smooth approximation, applying general principles from the theory of smooth approximations.

▶ **Proposition 17.** *Let* $\mathbb{A}$ *be a first-order reduct of* $D_\mathcal{F}$ *that is a model-complete core and such that* $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_\mathcal{F})$*. If* $(S, \sim)$ *is a minimal subfactor of* $\mathscr{C}^I_\mathbb{A}$ *such that* $\sim$ *has* $\mathrm{Aut}(\mathbb{A})$*-invariant classes, and* η *is a presmooth approximation of* $\sim$ *with respect to* $\mathrm{Aut}(\mathbb{A})$*, then* η *is very smooth with respect to* $\mathrm{Aut}(D_\mathcal{F})$*.*

Proof. We show that η is presmooth with respect to $\mathrm{Aut}(D_\mathcal{F})$. Let C be an equivalence class of $\sim$. By assumption, there exists an equivalence class C' of η and $a, b \in C \cap C'$ that are disjoint. By Lemma 12, $\mathrm{Aut}(D_\mathcal{F})$ has no algebraicity. Thus, there exists an automorphism $\alpha \in \mathrm{Aut}(D_\mathcal{F}, a)$ such that $\alpha(b)$ and b are disjoint. Note that b and $\alpha(b)$ are $\sim$-equivalent, since the equivalence classes of $\sim$ are $\mathrm{Aut}(\mathbb{A})$-invariant. Moreover, b and $\alpha(b)$ are η-equivalent, since $(a,b) \in \eta$ and η is $\mathrm{Aut}(\mathbb{A})$-invariant. Thus, we have disjoint elements $b, \alpha(b)$ in $C' \cap C$ and in the same orbit under $\mathrm{Aut}(D_\mathcal{F})$, i.e., η is presmooth with respect to $\mathrm{Aut}(D_\mathcal{F})$.

By Lemma 13, $\neq$ is invariant under $\mathrm{Pol}(\mathbb{A})$, and by Lemma 12, $\mathrm{Aut}(D_\mathcal{F})$ is n-primitive for all n. By [25, Lemma 8], we obtain that η is very smooth with respect to $\mathrm{Aut}(D_\mathcal{F})$. ◀

We show that this can be used to obtain a naked set for $\mathrm{Pol}(\mathbb{A})$, which implies by Theorem 4 that $\mathbb{A}$ pp-constructs every finite structure. We are in the situation where the original theorem from [25, Theorem 13] used to extend a naked set (or affine set) does not apply. Indeed, this result would require that $\mathscr{C}^I_\mathbb{A}$ be locally interpolated by $\mathrm{Pol}(\mathbb{A})$, which we do not have.

However, we do have that $\mathscr{C}^K_{D_\mathcal{F}}$ *is* locally interpolated by $\mathrm{Pol}(\mathbb{A})$. Indeed, we already know that any $f \in \mathrm{Pol}(\mathbb{A})$ locally interpolates an operation g that is canonical with respect to $\mathrm{Aut}(D_\mathcal{F}, <)$. If g is not an element of $\mathscr{C}^K_{D_\mathcal{F}}$, we show that $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ must contain a majority operation, a contradiction to our assumption that $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ has a naked set. The proof of the following lemma is similar to the proof of Lemma 34 in [25].

▶ **Lemma 18.** *Let $\mathbb{A}$ be a proper reduct of $D_\mathcal{F}$ such that U is invariant under* $\mathrm{Pol}(\mathbb{A})$*. Suppose that:*

- $\mathrm{Pol}(\mathbb{A})$ *contains a binary injection acting like a projection on $\{\leftarrow, \rightarrow\}$,*
- *there is a function in $\mathscr{C}^K_{(D_\mathcal{F},<)}$ that is not in $\mathscr{C}^K_{D_\mathcal{F}}$.*

Then $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ contains a majority operation and in particular does not have an affine set.

We can then proceed as in [26] and use Lemma 18 and the injective projections to circumvent the original limitation from [25, Theorem 13]. In the following, the "lifting" operation, which gives us that $\mathrm{Pol}(\mathbb{A})$ has a naked set, is performed by exhibiting a *uniformly continuous clone homomorphism* from $\mathrm{Pol}(\mathbb{A})$ to $\mathscr{C}^I_\mathbb{A} \curvearrowright S/{\sim}$, i.e., an arity-preserving map ξ such that $\xi(f \circ (g_1, \ldots, g_k)) = \xi(f) \circ (\xi(g_1), \ldots, \xi(g_k))$ for all $f, g_1, \ldots, g_k \in \mathrm{Pol}(\mathbb{A})$ of appropriate arities, and where the value of $\xi(f)$ for an n-ary $f \in \mathrm{Pol}(\mathbb{A})$ is only determined by the restriction $f|_S$ of f onto a finite subset S of A that does not depend on f. The existence of such a map shows that $\mathrm{Pol}(\mathbb{A})$ is not "richer" than $\mathscr{C}^I_\mathbb{A} \curvearrowright S/{\sim}$; in particular, if $\mathscr{C}^I_\mathbb{A} \curvearrowright S/{\sim}$ has a naked (resp. affine) set, then so does $\mathrm{Pol}(\mathbb{A})$. We refer the reader to [11] and in particular to Theorem 1 therein for more details.

▶ **Corollary 19.** *Let $\mathbb{A}$ be a proper reduct of $D_\mathcal{F}$ that admits injective projections and such that U is invariant under* $\mathrm{Pol}(\mathbb{A})$*. If $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow, \rightarrow\}$ has an affine set, we have $\mathscr{C}^K_{(D_\mathcal{F},<)} \subseteq \mathscr{C}^K_{D_\mathcal{F}}$. In particular, every $f \in$* $\mathrm{Pol}(\mathbb{A})$ *locally interpolates a function in $\mathscr{C}^K_{D_\mathcal{F}}$.*

Proof. Lemma 18 applies since $\mathbb{A}$ admits injective projections and U is invariant under $\mathrm{Pol}(\mathbb{A})$. For the second part, we know by Lemma 7 that every $f \in \mathrm{Pol}(\mathbb{A})$ locally interpolates an operation that is canonical with respect to $(D_\mathcal{F}, <)$. Such operations are in particular in $\mathscr{C}^K_{(D_\mathcal{F},<)}$. Since $\mathscr{C}^K_{(D_\mathcal{F},<)} \subseteq \mathscr{C}^K_{D_\mathcal{F}}$, we are done. ◀

▶ **Lemma 20.** *Let $\mathbb{A}$ be a proper reduct of $D_\mathcal{F}$ such that U is invariant under $\mathscr{C}^I_\mathbb{A}$. Let $(S, \sim)$ be an affine set of $\mathscr{C}^I_\mathbb{A}$ such that $S \subseteq I_k$ and where the $\sim$-equivalence classes are* $\mathrm{Aut}(\mathbb{A})$*-invariant. Then for all $a, b \in S$, a and b induce the same undirected graph in $H_\mathcal{F}$.*

Proof. Suppose first that there exist $a, b \in S$ with $a \not\sim b$ and such that a, b induce the same undirected graph in $H_\mathcal{F}$. Since both N and U are invariant under $\mathscr{C}^I_\mathbb{A}$, the set generated by $\{a, b\}$ under $\mathscr{C}^I_\mathbb{A}$ only consists of tuples all inducing the same undirected graph as a and b in $H_\mathcal{F}$. By minimality of $(S, \sim)$, such a set must be equal to S itself, so we are done.

Otherwise, whenever $a \not\sim b$, then a, b induce different undirected graphs. In other words, any tuples in S inducing the same undirected graph are in the same $\sim$-equivalence class. By assumption, $(S, \sim)$ is an affine set for $\mathscr{C}^I_\mathbb{A}$, and therefore there exists a ring R and a finite R-module M on $S/{\sim}$ such that all operations in $\mathscr{C}^I_\mathbb{A} \curvearrowright S/{\sim}$ are of the form $(x_1, \ldots, x_n) \mapsto \sum \lambda_i \cdot x_i$, where $\lambda_1, \ldots, \lambda_n \in R$ are such that $\sum \lambda_i = 1$. Let $a \in S$ be a tuple whose $\sim$-equivalence class is an arbitrary non-zero element of the module M, and let $b \in S$ be a tuple whose $\sim$-equivalence class is the zero element of M. Since M is finite, a (more precisely, its $\sim$-equivalence class $[a]$), has a finite order $n \geq 2$, that is, $n[a] = [b]$. Note that the tuples $q^n_1(a, b, \ldots, b), q^n_1(b, a, b, \ldots, b), \ldots, q^n_1(b, \ldots, b, a)$ all induce the same undirected graph, and they are all in S since $q^n_1 \in \mathscr{C}^I_\mathbb{A}$ by Lemma 10. By our assumption, all these tuples are in the same $\sim$-equivalence class. Since $\mathscr{C}^I_{\mathbb{A}} \curvearrowright S/{\sim}$ is affine, the action of q^n_1 on $S/{\sim}$ is an affine function of the form $(x_1, \ldots, x_n) \mapsto \sum \lambda_i \cdot x_i$, where $\lambda_1, \ldots, \lambda_n$ are elements of R and such that $\sum \lambda_i = 1$. If a is in the ith position, then $q^n_1(b, \ldots, b, a, b, \ldots, b)$ is $\sim$-equivalent to $\lambda_i \cdot [a]$. Thus, we get that $\lambda_i \cdot [a] = \lambda_j \cdot [a]$ for all $i, j \in \{1, \ldots, n\}$. We call this element $\lambda \cdot [a]$. Thus, the equivalence class of $q^n_1(a, \ldots, a)$ is $\sum_{i=1}^n \lambda \cdot [a] = n(\lambda \cdot [a]) = \lambda \cdot (n \cdot [a]) = \lambda \cdot [b] = [b]$. However, q^n_1 being affine, we also know that $q^n_1(a, \ldots, a)$ is $\sim$-equivalent to a, contradicting our assumption that $a \not\sim b$. ◀

▶ **Proposition 21.** *Let $\mathbb{A}$ be a proper reduct of $D_{\mathcal{F}}$ that admits injective projections and such that U is invariant under* $\mathrm{Pol}(\mathbb{A})$*. Assume that $\mathscr{C}^K_{D_{\mathcal{F}}} \curvearrowright \{\leftarrow, \rightarrow\}$ has an affine set. Suppose that there exists an affine set $(S, \sim)$ of $\mathscr{C}^I_{\mathbb{A}}$ with $S \subseteq I_k$ and $\sim$ having* $\mathrm{Aut}(\mathbb{A})$*-invariant classes. Suppose further that $\sim$ admits a* $\mathrm{Pol}(\mathbb{A})$*-invariant very smooth approximation with respect to* $\mathrm{Aut}(D_{\mathcal{F}})$*. Then* $\mathrm{Pol}(\mathbb{A})$ *admits a uniformly continuous clone homomorphism to $\mathscr{C}^I_{\mathbb{A}} \curvearrowright S/{\sim}$.*

Proof. Without loss of generality, we can assume that S is minimal with the property of intersecting two equivalence classes of $\sim$. Let η be a presmooth approximation of $\sim$ with respect to $\mathrm{Aut}(D_{\mathcal{F}})$. By [25, Lemma 8] and Lemmas 12 and 13, every $\mathrm{Pol}(\mathbb{A})$-invariant presmooth approximation of this naked set must be very smooth with respect to $\mathrm{Aut}(D_{\mathcal{F}})$.

Let $f \in \mathrm{Pol}(\mathbb{A})$ an n-ary function. Let $f' \in \mathscr{C}^K_{D_{\mathcal{F}}}$ be locally interpolated by f; such an operation exists by Corollary 19. Let q^n_i be the i-th injective projection. By Lemma 10, $q^n_i \in \mathscr{C}^I_{\mathbb{A}}$. We define $f''(x_1, \ldots, x_n) = f'(q^n_1(x_1, \ldots, x_n), \ldots, q^n_n(x_1, \ldots, x_n))$ and show that $f'' \in \mathscr{C}^I_{\mathbb{A}}$. For this let $a_1, a'_1, \ldots, a_n, a'_n$ be injective tuples of an arbitrary length such that a_i, a'_i are in the same $\mathrm{Aut}(\mathbb{A})$-orbit for all $1 \leq i \leq n$. Since $\mathrm{Aut}(\mathbb{A}) \subseteq \mathrm{Aut}(H_{\mathcal{F}})$, every pair a_i, a'_i induce the same undirected graph in $H_{\mathcal{F}}$. As $q^n_i \in \mathscr{C}^I_{\mathbb{A}}$ we know that $b_i := q^n_i(a_1, \ldots, a_n)$ and $b'_i := q^n_i(a'_1, \ldots, a'_n)$ are in the same $\mathrm{Aut}(\mathbb{A})$-orbit, too, and for the same reason as above, each pair b_i, b'_i induce the same undirected graph in $H_{\mathcal{F}}$. Moreover, the pairs of coordinates where the projection of b_i is in N are exactly the pair of coordinates where the projection of b'_i is in N. Now let $c := f''(a_1, \ldots, a_n) = f'(b_1, \ldots, b_n)$ and $c' := f''(a'_1, \ldots, a'_n) = f'(b'_1, \ldots, b'_n)$. Since $f' \in \mathscr{C}^K_{D_{\mathcal{F}}}$, since $\mathrm{Pol}(\mathbb{A})$ preserves N, and since the operation induced by f' on $\{\leftarrow, \rightarrow\}$ is an affine map, we know that c and c' are in the same orbit under $\mathrm{Aut}(\mathbb{A})$. Thus, $f'' \in \mathscr{C}^I_{\mathbb{A}}$ and therefore it acts on $I_k/\mathrm{Aut}(\mathbb{A})$.

We define $\xi(f)$ as the action of f'' on $S/{\sim}$. As in [25, Theorem 13], for all $a_1, \ldots, a_n \in S$, and any f' that is locally interpolated by f, we have $f(a_1, \ldots, a_n)(\eta \circ \sim)f'(a_1, \ldots, a_n)$. It follows that

$$f(q^n_1(a_1, \ldots, a_n), \ldots, q^n_n(a_1, \ldots, a_n))(\eta \circ \sim)f''(a_1, \ldots, a_n)$$

holds for all $a_1, \ldots, a_n \in S$. In particular, the definition of $\xi(f)$ does not depend on the choice of f' in the construction. Moreover, if $a_1, \ldots, a_n$ induce the same undirected graph in $H_{\mathcal{F}}$, then $q^n_i(a_1, \ldots, a_n)$ and a_i are in the same orbit under $\mathrm{Aut}(D_{\mathcal{F}})$, for all $i \in \{1, \ldots, n\}$. It follows that $f''(a_1, \ldots, a_n)$ and $f'(a_1, \ldots, a_n)$ are in the same orbit with respect to $\mathrm{Aut}(D_{\mathcal{F}})$, as f' is canonical with respect to $\mathrm{Aut}(D_{\mathcal{F}})$. Since $\mathrm{Aut}(D_{\mathcal{F}}) \subseteq \mathrm{Aut}(\mathbb{A})$, they are in the same orbit with respect to $\mathrm{Aut}(\mathbb{A})$, and therefore there are $\sim$-equivalent. Finally, this implies that $f(a_1, \ldots, a_n)(\eta \circ \sim)f''(a_1, \ldots, a_n)$ holds, whenever all $a_1, \ldots, a_n$ induce the same undirected graph in $H_{\mathcal{F}}$, which is the case for all $a_1, \ldots, a_n \in S$ by Lemma 20.

Now we show that ξ is a uniformly continuous clone homomorphism. It clearly preserves arities so we need to show it also preserves compositions. Let $f \in \mathrm{Pol}(\mathbb{A})$ be n-ary, $g_1, \ldots, g_n \in \mathrm{Pol}(\mathbb{A})$ be m-ary. Let $u_1, \ldots, u_m \in S$. Since $g_i(u_1, \ldots, u_m)(\eta \circ \sim)g''_i(u_1, \ldots, u_m)$ for all i, there exists $v_i \in S$ such that $g_i(u_1, \ldots, u_m)\ \eta\ v_i$. Then

$$\begin{aligned} f(g_1(u_1,\ldots,u_m),\ldots,g_n(u_1,\ldots,u_m)) \;\eta\; & f(v_1,\ldots,v_n) \\ & (\eta\circ\sim) f''(v_1,\ldots,v_n) \\ & (\eta\circ\sim) f''(g_1(u_1,\ldots,u_m),\ldots,g_n(u_1,\ldots,u_m)). \end{aligned}$$

from which we conclude that

$$\xi(f(g_1,\ldots,g_n))(u_1,\ldots,u_m) = \xi(f)(\xi(g_1)(u_1,\ldots,u_m),\ldots,\xi(g_n)(u_1,\ldots,u_m)).$$

To prove that ξ is uniformly continuous fix $n \in \mathbb{N}$ and let $a \not\sim b$ be any two k-ary tuples in S and define $F = \bigcup_{0\leq i\leq k}\{a_i,b_i\}^n$. Observe that if f,g agree on all $x \in \{a_i,b_i\}^n$ for each $i \leq k$, then they also agree on all $y \in \{a,b\}^n$ as $f(y)_i$ and $g(y)_i$ are fully determined by the values of f,g on a certain subset of $\{a_i,b_i\}^n$. Therefore they induce the same action on $S/{\sim}$ as in that case $f(x) \sim g(x)$ for all $x \in \{a,b\}^n$. Then it is clear that if f,g agree on F we also have $\xi(f) = \xi(g)$. This completes the proof. ◀

4.2.2 Second Case: Weakly Commutative Polymorphism

In the second case of Theorem 6, $\mathrm{Pol}(\mathbb{A})$ contains a well-behaved binary operation, which implies that $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ contains a majority operation; this contradicts our assumption that $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ has a naked set. The proof of Proposition 22 below is similar to that of Lemma 39 in [25] and is omitted due to space restrictions.

▶ **Proposition 22.** *Let $\mathbb{A}$ be a proper reduct of $D_\mathcal{F}$ with injective projections and such that U is invariant under* $\mathrm{Pol}(\mathbb{A})$. *Let $(S,\sim)$ be a minimal affine set of $\mathscr{C}^I_\mathbb{A}$ with $S \subseteq I_k$ and where $\sim$ has* $\mathrm{Aut}(\mathbb{A})$*-invariant equivalence classes. Suppose that there exists a binary $f \in$* $\mathrm{Pol}(\mathbb{A})$ *such that $f(a,b) \sim f(b,a)$ holds for all disjoint injective tuples $a,b \in V^k$ with $f(a,b), f(b,a) \in S$. Then $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ contains a majority operation.*

4.2.3 Classifying Problems with Bounded Treewidth Dualities

Finally, we briefly describe the proof strategy for Theorem 3. It is very similar to the one outlined above and shares all the intermediate steps. Only the starting distinction changes, where we distinguish between whether $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ has an affine set or not.

If $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ does not have an affine set, then the second item in Proposition 15 states that $\mathrm{Pol}(\mathbb{A})$ contains a so-called *canonical pseudo-majority operation*, and $\mathbb{A}$ has bounded relational width by [9]. Moreover, since $D_\mathcal{F}$ is homogeneous in a binary language, [22, Theorem 2] entails that $\mathbb{A}$ has relational width at most $(4, \max(6,\ell))$, where ℓ is the maximal size of a tournament in $\mathcal{F}$. It follows from general principles [15] that there exists a duality for the class of structures admitting a homomorphism to $\mathbb{A}$ that has treewidth bounded by $\max(6,\ell,r)$, where r is the maximal arity of a relation of $\mathbb{A}$.

If $\mathscr{C}^I_\mathbb{A} \curvearrowright I_k/\mathrm{Aut}(\mathbb{A})$ has an affine set for some $k \geq 1$ (by Lemma 16), either we get an affine set for $\mathrm{Pol}(\mathbb{A})$ (by Proposition 21) or $\mathscr{C}^K_{D_\mathcal{F}} \curvearrowright \{\leftarrow,\rightarrow\}$ contains a majority operation (by Proposition 22), again a contradiction since a majority operation cannot be represented as an affine map over any module.

References

1 Lovkush Agarwal and Michael Kompatscher. $2^{\aleph_0}$ pairwise nonisomorphic maximal-closed subgroups of Sym($\mathbb{N}$) via the classification of the reducts of the henson digraphs. *J. Symb. Log.*, 83(2):395–415, 2018. doi:10.1017/JSL.2017.74.

2 Libor Barto, Michael Kompatscher, Miroslav Olšák, Van Trung Pham, and Michael Pinsker. Equations in oligomorphic clones and the constraint satisfaction problem for ω-categorical structures. *J. Math. Log.*, 19(2):1950010:1–1950010:31, 2019. doi:10.1142/S0219061319500107.

3 Manuel Bodirsky. Cores of countably categorical structures. *Log. Methods Comput. Sci.*, 3(1), 2007. doi:10.2168/LMCS-3(1:2)2007.

4 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2008. doi:10.1007/978-3-540-70583-3_16.

5 Manuel Bodirsky and Santiago Guzmán-Pro. Forbidden tournaments and the orientation completion problem. *CoRR*, abs/2309.08327, 2023. doi:10.48550/arXiv.2309.08327.

6 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *Computer Science - Theory and Applications, First International Symposium on Computer Science in Russia, CSR 2006, St. Petersburg, Russia, June 8-12, 2006, Proceedings*, volume 3967 of *Lecture Notes in Computer Science*, pages 114–126. Springer, 2006. doi:10.1007/11753728_14.

7 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2):9:1–9:41, 2010. doi:10.1145/1667053.1667058.

8 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs. *SIAM J. Comput.*, 48(4):1224–1264, 2019. doi:10.1137/16M1082974.

9 Manuel Bodirsky and Antoine Mottet. Reducts of finitely bounded homogeneous structures, and lifting tractability from finite-domain constraint satisfaction. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 623–632. ACM, 2016. doi:10.1145/2933575.2934515.

10 Manuel Bodirsky and Antoine Mottet. A dichotomy for first-order reducts of unary structures. *Log. Methods Comput. Sci.*, 14(2), 2018. doi:10.23638/LMCS-14(2:13)2018.

11 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the American Mathematical Society*, 367(4):2527–2549, 2015. URL: http://www.jstor.org/stable/24513079.

12 Manuel Bodirsky and Michael Pinsker. Canonical functions: a new proofs via topological dynamics. *Contributions to Discrete Mathematics*, 16, 2021. doi:10.11575/cdm.v16i2.71724.

13 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. *J. Symb. Log.*, 78(4):1036–1054, 2013. doi:10.2178/JSL.7804020.

14 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.

15 Andrei A. Bulatov, Andrei A. Krokhin, and Benoît Larose. Dualities for constraint satisfaction problems. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints - An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*, volume 5250 of *Lecture Notes in Computer Science*, pages 93–124. Springer, 2008. doi:10.1007/978-3-540-92800-3_5.

16 Roland Fraïssé. Une hypothèse sur l'extension des relations finies et sa vérification dans certaines classes particulières (deuxième partie). *Synthese*, 16(1):34–46, 1966. URL: http://www.jstor.org/stable/20114493.

17 Pierre Gillibert, Julius Jonušas, Michael Kompatscher, Antoine Mottet, and Michael Pinsker. Hrushovski's encoding and ω-categorical CSP monsters. 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 131:1–131:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.131.

18 Pierre Gillibert, Julius Jonušas, Michael Kompatscher, Antoine Mottet, and Michael Pinsker. When symmetries are not enough: A hierarchy of hard constraint satisfaction problems. *SIAM J. Comput.*, 51(2):175–213, 2022. doi:10.1137/20M1383471.

19 Wilfrid Hodges. *Model theory*. Cambridge University Press, 1993.

20 Michael Kompatscher and Trung Van Pham. A complexity dichotomy for poset constraint satisfaction. *FLAP*, 5(8):1663–1696, 2018. URL: https://www.collegepublications.co.uk/downloads/ifcolog00028.pdf.

21 Benoît Larose and László Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Int. J. Algebra Comput.*, 16(3):563–582, 2006. doi:10.1142/S0218196706003116.

22 Antoine Mottet, Tomáš Nagy, Michael Pinsker, and Michal Wrona. Smooth approximations and relational width collapses. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 138:1–138:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.138.

23 Antoine Mottet, Tomáš Nagy, Michael Pinsker, and Michal Wrona. When symmetries are enough: collapsing the bounded width hierarchy for infinite-domain CSPs. *SIAM Journal on Computing*, 2024. To appear.

24 Antoine Mottet and Michael Pinsker. Cores over Ramsey structures. *J. Symb. Log.*, 86(1):352–361, 2021. doi:10.1017/JSL.2021.6.

25 Antoine Mottet and Michael Pinsker. Smooth approximations and CSPs over finitely bounded homogeneous structures. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 36:1–36:13. ACM, 2022. doi:10.1145/3531130.3533353.

26 Antoine Mottet, Michael Pinsker, and Tomáš Nagy. An order out of nowhere: a new algorithm for infinite-domain CSPs. In *Proceedings of ICALP*, 2024. To appear.

27 Jaroslav Nešetřil and Vojtěch Rödl. The partite construction and ramsey set systems. *Discrete Mathematics*, 75(1):327–334, 1989. doi:10.1016/0012-365X(89)90097-6.

28 E. L. Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematics Studies*, 1941.

29 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *J. ACM*, 67(5):30:1–30:78, 2020. doi:10.1145/3402029.

Equitable Connected Partition and Structural Parameters Revisited: N-Fold Beats Lenstra

Václav Blažej
University of Warwick, UK

Dušan Knop
Czech Technical University in Prague, Czech Republic

Jan Pokorný
Czech Technical University in Prague, Czech Republic

Šimon Schierreich
Czech Technical University in Prague, Czech Republic

Abstract

In the Equitable Connected Partition (ECP for short) problem, we are given a graph $G = (V, E)$ together with an integer $p \in \mathbb{N}$, and our goal is to find a partition of V into p parts such that each part induces a connected sub-graph of G and the size of each two parts differs by at most 1. On the one hand, the problem is known to be NP-hard in general and W[1]-hard with respect to the path-width, the feedback-vertex set, and the number of parts p combined. On the other hand, fixed-parameter algorithms are known for parameters the vertex-integrity and the max leaf number.

In this work, we systematically study ECP with respect to various structural restrictions of the underlying graph and provide a clear dichotomy of its parameterised complexity. Specifically, we show that the problem is in FPT when parameterized by the modular-width and the distance to clique. Next, we prove W[1]-hardness with respect to the distance to cluster, the 4-path vertex cover number, the distance to disjoint paths, and the feedback-edge set, and NP-hardness for constant shrub-depth graphs. Our hardness results are complemented by matching algorithmic upper-bounds: we give an XP algorithm for parameterisation by the tree-width and the distance to cluster. We also give an improved FPT algorithm for parameterisation by the vertex integrity and the first explicit FPT algorithm for the 3-path vertex cover number. The main ingredient of these algorithms is a formulation of ECP as N-fold IP, which clearly indicates that such formulations may, in certain scenarios, significantly outperform existing algorithms based on the famous algorithm of Lenstra.

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

Keywords and phrases Equitable Connected Partition, structural parameters, fixed-parameter tractability, N-fold integer programming, tree-width, shrub-depth, modular-width

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.29

Related Version *Full Version*: `https://arxiv.org/abs/2404.18968`

Funding This work was co-funded by the European Union under the project Robotics and advanced industrial production (reg. no. CZ.02.01.01/00/22_008/0004590).
Václav Blažej: Supported by the Engineering and Physical Sciences Research Council [grant number EP/V044621/1].
Dušan Knop: Acknowledges the support of the Czech Science Foundation Grant No. 22-19557S.
Jan Pokorný: Acknowledges the support of the Grant Agency of the CTU in Prague funded grant No. SGS23/205/OHK3/3T/18.
Šimon Schierreich: Acknowledges the support of the Czech Science Foundation Grant No. 22-19557S and of the Grant Agency of the CTU in Prague funded grant No. SGS23/205/OHK3/3T/18.

Acknowledgements We are grateful to all anonymous referees for their valuable comments.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 29; pp. 29:1–29:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A partition of a set V into $p \in \mathbb{N}$ parts is a set $\pi = \{V_1, \dots, V_p\}$ of subsets of V such that for every $i, j \in [p]$: $V_i \cap V_j = \emptyset$ and $\bigcup_{i=1}^{p} V_i = V$. In the Equitable Connected Partition problem, we are given an undirected graph $G = (V, E)$ together with an integer p, and our goal is to partition the vertex set V into p *parts* such that the sizes of each two parts differ by at most 1 and each class induces a connected sub-graph. Formally, our problem is defined as follows.

Equitable Connected Partition (ECP)

Input: A simple undirected and connected n-vertex graph $G = (V, E)$ and a positive integer $p \in \mathbb{N}$.

Question: Is there a partition $\pi = \{V_1, \dots, V_p\}$ of V such that every part $G[V_i]$ is connected, and $||V_i| - |V_j|| \leq 1$ for every pair $i, j \in [p]$?

The Equitable Connected Partition problem naturally arises in many fields such as redistricting theory [2, 50, 55], which is a subfield of computational social choice theory, VLSI circuit design [7], parallel computing [5], or image processing [52], to name a few.

One of the most prominent problems in the graph partitioning direction is the Bisection problem, where our goal is to split the vertex set into two parts A and B, each part of size at most $\lceil \frac{n}{2} \rceil$, such that the number of edges between A and B is at most some given $k \in \mathbb{N}$. Bisection is NP-hard [36] even if we restrict the input to unit disc graphs [24] and is heavily studied from the parameterised complexity perspective; see, e.g., [6, 13, 19, 30, 54]. The natural generalisation of the Bisection problem is called Balanced Partitioning where we partition the vertices into $p \in \mathbb{N}$ parts, each of size at most $\lceil \frac{n}{p} \rceil$. Balanced Partitioning is NP-hard already on trees [27] and a disjoint union of cliques [4]. The parameterised study of this problem is due to Ganian and Obdržálek [35] and van Bevern et al. [6]. In all the aforementioned problems, we are given only the upper-bound on the size of each part; hence, the parts are not necessarily equitable. Moreover, there is no connectivity requirement for the parts. For a survey of graph partitioning problems, we refer the reader to the monograph of Buluç et al. [14].

On the equitability side, the most notable direction of research is the Equitable k-Colouring problem (EC for short). Here, we are given an undirected graph G and the goal is to decide whether there is a proper colouring of the vertices of G using k colours such that the sizes of each two colour classes differ by at most one. Note that the graph induced by each colour class is necessarily an independent set, and hence is disconnected. As the k-Colouring problem can be easily reduced to the Equitable k-Colouring, it follows that EC is NP-hard. Polynomial-time algorithms are known for many simple graph classes, such as graphs of bounded tree-width [9, 16], split graphs [15], and many others [31]. The parameterised study was, to the best of our knowledge, initiated by Fellows et al. [28] and continued in multiple subsequent works [26, 29, 40]. For a detailed survey of the results on EC, we refer the reader to the monograph by Lih [51].

The Equitable Connected Partition problem then naturally brings the concepts of equitability and connectivity of the vertex set together. It is known that ECP is NP-complete [2]. Moreover, the problem remains NP-complete even if G is a planar graph or for every fixed p at least 2 [23, 36]. Enciso et al. [26] were the first who studied ECP from the viewpoint of parameterised complexity. They showed that ECP is fixed-parameter tractable with respect to the vertex cover number and the maximum leaf number of G. On the negative side, they showed that it is W[1]-hard to decide the problem for the combined parameter the

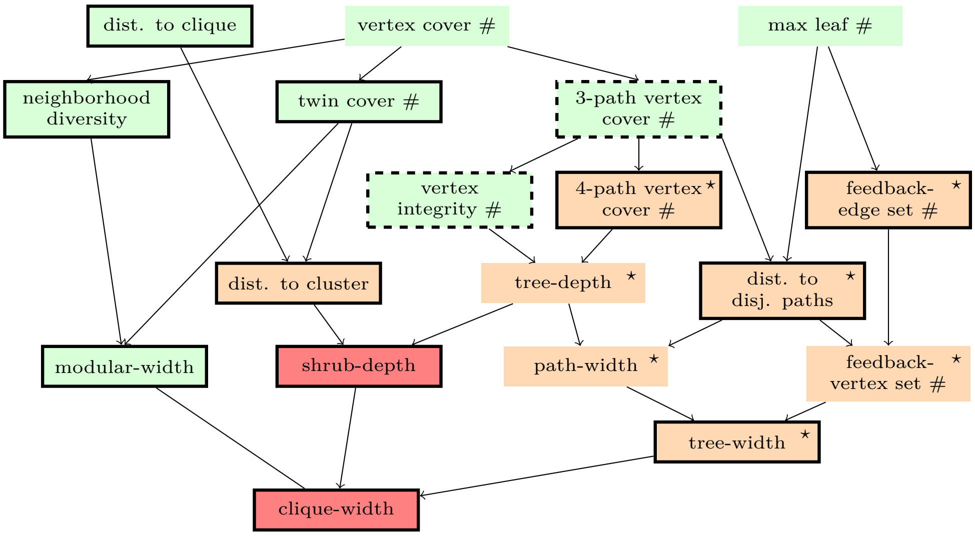


Figure 1 An overview of our results. The parameters for which the problem is in FPT are coloured green, the parameters for which ECP is W[1]-hard and in XP have an orange background, and para-NP-hard combinations are highlighted in red. Arrows indicate generalisations; e.g., modular width generalises both neighbourhood diversity and twin-cover number. The solid thick border represents completely new results, and the dashed border represents an improvement of previously known algorithms. All our W[1]-hardness results hold even when the problem is additionally parameterized by the number of parts p. Additionally, we show that the results marked with $\star$ becomes fixed-parameter tractable if the size of a larger part $\lceil n/p \rceil$ is an additional parameter.

path-width, the feedback-vertex set, and the number of parts p. Moreover, they gave an XP algorithm for ECP parameterised by tree-width. Later, Gima et al. [38] showed that the problem is fixed-parameter tractable when parameterised by the vertex-integrity of G. Very recently, Gima and Otachi [39] proved that ECP is W[1]-hard when parameterised by the tree-depth of G.

A more general variant of Equitable Connected Partition with parametric lower- and upper-bounds on the sizes of parts was studied by Ito et al. [43], and Blažej et al. [8] very recently introduced the requirement on the maximum diameter of each part.

It is worth pointing out that Equitable Connected Partition is also significant from a theoretical point of view. Specifically, this problem is a very common starting point for many W[1]-hardness reductions; see, e.,g., [6, 8, 20, 53]. Surprisingly, the graph in multiple of the before-mentioned reductions remains the same as in original instance, and therefore our study directly strengthens the results obtained in these works. Since the complexity picture with respect to structural parameters is rather incomplete, many natural questions arise. For example, what is the parameterised complexity of ECP when parameterised by the 4-path vertex cover number? Or, is ECP in FPT when parameterised by the feedback-edge set? Last but not least, is the problem easier to decide on graphs that are dense, such as cliques?

1.1 Our Contribution

In our work, we continue the line of study of the Equitable Connected Partition problem initiated by Enciso et al. [26] almost 15 years ago. For an overview of our results, we refer the reader to Figure 1; however, we believe that our contribution is much broader. We try to summarise it in the following four points.

First, directly following pioneering work on structural parameterisation of ECP, we provide a complete dichotomy between tractable and intractable cases for structural parameters that are bounded for sparse graphs. Namely, we provide W[1]-hardness proofs for ECP with respect to the 4-path vertex cover number and the feedback-edge set number, which encloses a gap between structural parameters that were known to be tractable – the vertex-cover number and the max-leaf number – and those that were known to be W[1]-hard– the path-width and the feedback-vertex set. It should also be mentioned that our constructions not only give much stronger intractability results but, at the same time, are much simpler compared to the original construction of Enciso et al. [26].

Second, we also turn our attention to dense graphs, which have, so far, been completely overlooked in the relevant literature. On our way to fixed-parameter tractable algorithms for various structural parameters, we prove polynomial-time solvability of some specific graph classes. Again, we provide a clear boundary between tractable and intractable cases. However, it turns out that for dense graphs, the problem is much easier.

Third, we clearly show where the limits of the parameterized complexity framework in the study of structural parameterisation of Equitable Connected Partition are. In particular, we show that the problem is NP-hard already on graphs of shrub-depth equal to 3, clique-width equal to 3, and twin-width equal to 2. Moreover, in some cases, our complexity results are tight. For example, we give a polynomial-time algorithm for graphs of clique-width 2.

Last but not least, in order to provide all the algorithms, we use multiple different techniques. Naturally, some algorithms are based on standard techniques such as dynamic-programming over decomposition or kernelisation; however, many of them still require deep insights into the structure of the solution and the instances. However, some of them use very careful branching together with formulation of the problem using N-fold integer linear programming, which is, informally speaking, an integer linear program with specific shape of the constraints. We are convinced that the technique of N-fold integer linear programming in the design and analysis of fixed-parameter tractable algorithms deserves more attention from the parameterised complexity community, as it in many scenarios significantly outperforms the classical FPT algorithms based on the famous Lenstra's algorithm; see, e.g., [3, 12, 45, 46, 47].

2 Preliminaries

We assume the reader to be familiar with the basic graph-theoretical notation as given by Diestel [21] and with the basics of parameterised complexity [18].

Structural Parameters. In this sub-section, we provide definitions of less widespread structural parameters we study in this work. Definitions of all the remaining parameters are provided in the full version.

▶ **Definition 1** (d-path vertex cover). *Let $G = (V, E)$ be an undirected graph and $d \in \mathbb{N}$ be an integer. A* d-path vertex cover *is a set $C \subseteq V$ such that the graph $G \setminus C$ contains no path with d vertices as a sub-graph. The* d-path vertex cover number $d\text{-pvcn}(G)$ *is the size of a minimum d-path vertex cover in G.*

Note that the 2-path vertex cover number is, in fact, the standard *vertex cover number* of a graph.

▶ **Definition 2** (Vertex integrity). *Let $G = (V, E)$ be an undirected graph.* Vertex integrity*, denoted* $\mathrm{vi}(G)$*, is the minimum $k \in \mathbb{N}$ such that there is a set $X \subseteq V$ of size at most k and every connected component C of $G \setminus X$ contains at most k vertices.*

▶ **Definition 3** (Distance to $\mathcal{G}$). *Let G be a graph, and $\mathcal{G}$ be a graph family. A set $M \subseteq V(G)$ is a* modulator *to $\mathcal{G}$, if $G \setminus M \in \mathcal{G}$. The* distance *to $\mathcal{G}$, denoted* $\mathrm{dist}_{\mathcal{G}}(G)$*, is the size of a minimum modulator to $\mathcal{G}$.*

In our paper, we will focus on the *distance to clique*, denoted by $\mathrm{dc}(G)$, the distance to disjoint union of cliques, which is usually referred to as the *distance to cluster graph* or the *cluster vertex deletion* [22] and denoted $\mathrm{dcg}(G)$, and the *distance to disjoint paths*, denoted by $\mathrm{ddp}(G)$.

▶ **Definition 4** (Neighbourhood-diversity [48]). *Let $G = (V, E)$ be a graph. We say that two vertices $u, v \in V$ have the same type iff $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The* neighbourhood diversity *of G is at most d, if there exists a partition of V into at most d sets such that all vertices in each set have the same type.*

Let $T_1, \ldots, T_d$ be a partition of V such that for each $u, v \in T_i$, $i \in [d]$, it holds that u and v are of the same type according to Definition 4. Observe that each type is either independent set or a clique. We define *type graph* to be an undirected graph with vertices being the types $T_1, \ldots, T_d$ and two vertices corresponding to some types T_i and T_j are connected by an edge iff there exists an edge $\{u, v\} \in E(G)$ such that $u \in T_i$ and $v \in T_j$.

▶ **Definition 5** (Modular-width [32]). *Consider graphs that can be obtained from an algebraic expression that uses only the following operations:*

1. *create an isolated vertex,*
2. *the disjoint union of two disjoint graphs G_1 and G_2 which is a graph $(V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$,*
3. *the complete join of two disjoint graphs G_1 and G_2 which produces a graph $(V(G_1) \cup V(G_2), E(G_1) \cup E(G_2) \cup \{\{u, v\} \mid u \in V(G_1) \text{ and } v \in V(G_2)\})$.*
4. *the substitution with respect to some pattern graph P – for a graph P with vertices $p_1, \ldots, p_\ell$ and disjoint graphs $G_1, \ldots, G_\ell$, the substitution of the vertices of P by the graphs $G_1, \ldots, G_\ell$ is the graph with vertex set $\bigcup_{i=1}^{\ell} V(G_i)$ and edge set $\bigcup_{i=1}^{\ell} E(G_i) \cup \{\{u, v\} \mid u \in V(G_i),\ v \in V(G_j),\ \text{and } \{p_i, p_j\} \in E(P)\}$.*

The width of such an algebraic expression is the maximum number of operands used by any occurrence of the substitution operation. The modular-width *of a graph G, denoted* $\mathrm{mw}(G)$*, is the least integer m such that G can be obtained from such algebraic expression of width m.*

N-fold Integer Programming. In recent years, integer linear programming (ILP) has become a very useful tool in the design and analysis of fixed-parameter tractable algorithms [37]. One of the best known results in this line of research is probably Lenstra's algorithm, roughly showing that ILP with bounded number of variables is solvable in FPT time [49].

In this work, we use the so-called *N-fold integer programming* formulation. Here, the problem is to minimise a linear objective over a set of linear constraints with a very restricted structure. In particular, the constraints are as follows. We use $x^{(i)}$ to denote a set of t_i variables (a so-called *brick*).

$$D_1 x^{(1)} + D_2 x^{(2)} + \cdots + D_N x^{(N)} = \mathbf{b}_0 \tag{1}$$

$$A_i x^{(i)} = \mathbf{b}_i \qquad \forall i \in [N] \tag{2}$$

$$\mathbf{0} \le x^{(i)} \le \mathbf{u}_i \qquad \forall i \in [N] \tag{3}$$

Where we have $D_i \in \mathbb{Z}^{r \times t_i}$ and $A_i \in \mathbb{Z}^{s_i \times t_i}$; let us denote $s = \max_{i \in [N]} s_i$, $t = \max_{i \in [N]} t_i$, and let the dimension be d, i.e., $d = \sum_{i \in [N]} t_i \le Nt$. Constraints (1) are the so-called *linking constraints* and the rest are the *local constraints*. In the analysis of our algorithms, we use the following result of Eisenbrand et al. [25].

▶ **Proposition 6** ([25, Corollary 91]). *N-fold IP can be solved in $a^{r^2 s + rs^2} \cdot d \cdot \log(d) \cdot L$ time, where*

- *L is the maximum feasible value of the objective and*
- *$a = r \cdot s \cdot \max_{i \in [N]} (\max(\|D_i\|_\infty, \|A_i\|_\infty))$.*

3 Algorithmic Results

In this section, we provide our algorithmic results. The first algorithm is for ECP parameterised by the vertex-integrity and combines careful branching with N-fold integer programming. Specifically, Gima et al. [38] showed that ECP is in FPT with respect to this parameter by giving an algorithm running in $k^{k^{k^{\mathcal{O}(k)}}} \cdot n^{\mathcal{O}(1)}$ time, where $k = \mathrm{vi}(G)$. We show that using an N-fold IP formulation, we can give a simpler algorithm with a doubly exponential improvement in the running time.

▶ **Theorem 7.** *The* Equitable Connected Partition *problem is fixed-parameter tractable parameterised by the vertex-integrity* vi *and can be solved in $k^{\mathcal{O}(k^4)} \cdot n \log n$ time, where $k = \mathrm{vi}(G)$.*

Proof. First, if it holds that $p > k$, we use the algorithm of Gima et al. [38] which, in this special case, runs in time $k^{\mathcal{O}(k^2)} \cdot n$. Therefore, the bottleneck of their approach is clearly the case when $p \le k$. In what follows, we introduce our own procedure for this case, which is based on the N-fold integer programming. Note that the algorithm of Gima et al. [38] for the case $p \le k$ is based on the algorithm of Lenstra [49].

First, we guess (by guessing we mean exhaustively trying all possibilities) a partition of the modulator vertices X in the solution. Let this solution partition be $X_1, \ldots, X_p$. Furthermore, we guess which (missing) connections between the vertices in the modulator will be realised through the components of $G - X$. Let $E(X)$ be the set of these guessed connections.

Now, we check the validity of our guess using the (configuration) N-fold ILP. Each component of $G - X$ (call them *pieces*) has at most k vertices; therefore, it can be split in at most k chunks (not necessarily connected) that will be attached to some modulator vertices already assigned to the parts of the solution. Let $\mathcal{P}(G, X)$ be the set of all pieces of $G - X$. Now, we want to verify if there exists a selection of chunks for every piece so that when we collect these together the solution is indeed connected and contains the right number of vertices. Thus, there are altogether at most k^k configurations of chunks in a piece. Let $\mathcal{C}(Z)$ be the set of all configurations of a piece Z. Let $s^Z_{C,i}$ be the number of vertices in the chunk attached to the i-th part from a piece Z in the configuration C.Let Z be a piece and $C \in \mathcal{C}(Z)$, we set $e^Z_C(u, v) = 1$ if the chunk assigned by C to the part containing both $u, v \in X$ connects u and v.

Now, we have to ensure (local constraint) that each piece is in exactly one configuration

$$\sum_{C \in \mathcal{C}(Z)} x^Z_C = 1 \qquad \forall Z \in \mathcal{P}(G, X)\,. \tag{4}$$

Observe that these constraints have no variables in common for two distinct elements of $\mathcal{P}(G, X)$. The rest of the necessary computation uses global constraints. We ensure that the total contribution of chunks assigned to the parts is the correct number (x_i is a binary slack such that $\sum_i x_i = n \bmod p$):

$$x_i + \sum_{Z \in \mathcal{P}(G,X)} \sum_{C \in \mathcal{C}(Z)} s^Z_{C,i} \cdot x^Z_C = \lceil n/p \rceil - |X_i| \qquad \forall i \in [p] \tag{5}$$

Next, we have to verify the connectivity of parts in X

$$\sum_{Z \in \mathcal{P}(G,X)} \sum_{C \in \mathcal{C}(Z)} e^Z_C(u,v) \cdot x^Z_C \geq 1 \qquad \forall \{u,v\} \in E(X) \tag{6}$$

It is not hard to verify, that the parameters of N-fold IP are as follows:

- the number s of local constraints in a brick is exactly 1 as there is a single local constraint (4) for each piece,
- the number r of global constraints is in $\mathcal{O}\left(k^2\right)$: there are $p \leq k$ constraints (5) and $\binom{k}{2}$ constraints (6),
- the number t of variables in a brick is $|\mathcal{C}(Z)|$ which is $k^{\mathcal{O}(k)}$, and
- $a \in \mathcal{O}\left(k^3\right)$, since all coefficients in the constraints are bounded by k in absolute value.

Thus, using Proposition 6, the Equitable Connected Partition problem can be solved in $k^{\mathcal{O}\left(k^4\right)} \cdot n \log n$ time. ◀

Using techniques from the proof of Theorem 7, we may give a specialised algorithm for Equitable Connected Partition parameterised by the 3-path vertex cover. The core idea is essentially the same, but the components that remain after removing the modulator are much simpler: they are either isolated vertices or isolated edges. This fact allows us to additionally speed the algorithm up.

▶ **Theorem 8.** *The Equitable Connected Partition problem is fixed-parameter tractable parameterised by the 3-path vertex cover number and can be solved in $k^{\mathcal{O}\left(k^2\right)} \cdot n \log n$ time, where $k = 3$-pvcn(G).*

The next result is an XP algorithm with respect to the tree-width of G. It should be noted that a similar result was reported already by Enciso et al. [26]; however, their proof was never published[1] and the only clue for the algorithm the authors give in [26] is that this algorithm “can be proved using standard techniques for problems on graphs of bounded treewidth”. Therefore, to fill this gap in the literature, we give our own algorithm.

▶ **Theorem 9.** *The Equitable Connected Partition problem is in XP when parameterized by the tree-width tw of G.*

Proof sketch. The algorithm is, as is usual, a leaf-to-root dynamic programming algorithm along a nice tree decomposition. The crucial observation we need for the algorithm is that at every moment of the computation, there are at most $\mathcal{O}(\text{tw})$ opened parts. This holds because each bag forms a separator in G and therefore no edge can “circumvent” currently processed bag.

The algorithm then proceeds as follows. In each node x of the tree-decomposition, we remember all possible partitions of vertices into open parts and the sizes of each open cluster including already forgotten vertices. We require that each open part is connected. Once a new vertex v is introduced, we have three possibilities: create a new part consisting of only

[1] In particular, in their conference version, Enciso et al. [26] promised to include the proof in an extended version, which, however, has never been published. There is also a version containing the appendix of the conference paper available from `https://www.researchgate.net/publication/220992885_What_Makes_Equitable_Connected_Partition_Easy`; however, even in this version, the proof is not provided.

v, put v into an existing opened part, or merge (via v) multiple already existing parts into a new one. When a vertex v is forgotten we need to check whether v is the last vertex of its part and, if yes, whether the part v is member of is of size $\lfloor n/p \rfloor$ or $\lceil n/p \rceil$. In join nodes, we just merge two records from child nodes with the same partition of bag vertices, or we merge different partitions whose connectivity is secured by the past vertices.

Once the dynamic table is correctly filled, we ask whether the dynamic table for the root of the tree-decomposition stores `true` in its single cell. The size of each dynamic programming table is $\mathrm{tw}^{\mathcal{O}(\mathrm{tw})} \cdot n^{\mathcal{O}(\mathrm{tw})} = n^{\mathcal{O}(\mathrm{tw})}$, and each table can be computed in the same time. Therefore, the algorithm runs in $\mathcal{O}\left(n \cdot \mathrm{tw}\right) \cdot n^{\mathcal{O}(\mathrm{tw})}$ time, which is in XP. ◀

Observe that if the size of every part is bounded by a parameter ς, the size of each dynamic programming table is $\mathrm{tw}^{\mathcal{O}(\mathrm{tw})} \cdot \varsigma^{\mathcal{O}(\mathrm{tw})}$ and we need the same time to compute each cell. Therefore, the algorithm also shows the following tractability result.

▶ **Corollary 10.** *The* EQUITABLE CONNECTED PARTITION *problem is fixed-parameter tractable parameterised by the tree-width* tw *and the size of a large part* $\varsigma = \lceil n/p \rceil$ *combined.*

In other words, the EQUITABLE CONNECTED PARTITION problem becomes tractable if the tree-width is bounded and the number of parts is large.

So far, we investigated the complexity of the problem mostly with respect to structural parameters that are bounded for sparse graphs. Now, we turn our attention to parameters that our bounded for dense graphs. Note that such parameters are indeed interesting for the problem, as the problem becomes polynomial-time solvable on cliques. We were not able to find this result in the literature, and, therefore, we present it in its entirety.

▶ **Observation 11.** *The* EQUITABLE CONNECTED PARTITION *problem can be solved in linear time if the graph G is a clique.*

Proof. First, we determine the number of parts of size $\lceil n/p \rceil$ as $\ell = (n \bmod p)$, and the number of smaller parts of size $\lfloor n/p \rfloor$ as $s = p - \ell$. Now, we arbitrarily assign vertices to p parts such that the first ℓ parts contain $\lceil n/p \rceil$ vertices and the remaining s parts contain exactly $\lfloor n/p \rfloor$ vertices. This, in fact, creates an equitable partition. Moreover, every partition is connected, since each pair of vertices is connected by an edge in G. ◀

Following the usual approach of distance from triviality [1, 41], we study the problem of our interest with respect to the distance to clique. We obtain the following tractability result.

▶ **Theorem 12.** *The* EQUITABLE CONNECTED PARTITION *problem is fixed-parameter tractable when parameterised by the distance to clique k.*

Next, we prove polynomial-time solvability for a more general class of graphs than cliques. Namely, we provide a tractable algorithm for co-graphs.

▶ **Theorem 13.** *The* EQUITABLE CONNECTED PARTITION *problem can be solved in polynomial time if the graph G is a co-graph.*

Next structural parameter we study is the neighbourhood diversity, which is generalisation of the famous vertex cover number that, in contrast, allows for large cliques to be present in G. Later on, we will also provide a fixed-parameter tractable algorithm for a more general parameter called modular-width; however, the algorithm for neighbourhood diversity will serve as a building block for the later algorithm, and therefore we find it useful to present the algorithm in its entirety.

▶ **Theorem 14.** *The* Equitable Connected Partition *problem is fixed-parameter tractable parameterised by the neighbourhood diversity* $\operatorname{nd}(G)$*.*

Proof. We first observe that each connected sub-graph of G "induces" a connected graph of the type-graph of G. More precisely, each connected sub-graph of G is composed of vertices that belong to types of G that induce a connected sub-graph of the type-graph. Therefore, the solution is composed of various realisations of connected sub-graphs of the type-graph of G. Note that there are at most $2^{\operatorname{nd}(G)}$ connected sub-graphs of the type-graph of G. We will resolve this task using an ILP using integral variables x_H^t for a type t and a connected sub-graph H of the type-graph of G. Furthermore, we have additional variables x_H. That is, the total number of variables is (upper-bounded by) $\operatorname{nd}(G) \cdot 2^{\operatorname{nd}(G)} + 2^{\operatorname{nd}(G)}$. The meaning of a variable x_H^t is "how many vertices of type t we use in a realisation of H". The meaning of a variable x_H is "how many realisations of H there are in the solution we find". We write $t \in H$ for a type that belongs to H (a connected sub-graph of the type-graph). Let σ denote the lower-bound on the size of parts of a solution, that is, $\sigma = \lfloor n/k \rfloor$. In order for this to hold we add the following set of constraints (here, $\xi_G = 0$ if $n = k \cdot \sigma$ and $\xi_G = 1$, otherwise):

$$\sigma x_H \le \sum_{t \in H} x_H^t \le (\sigma + \xi_G) x_H \qquad \forall H \tag{7}$$

$$x_H \le x_H^t \qquad \forall H\, \forall t \in H \tag{8}$$

$$\sum_H x_H^t = n_t \qquad \forall t \in T(G) \tag{9}$$

$$0 \le x_H^t \le n_t,\ x_H^t \in \mathbb{Z} \qquad \forall H\, \forall t \in H \tag{10}$$

$$0 \le x_H,\ x_H^t \in \mathbb{Z} \qquad \forall H \tag{11}$$

That is, (9) ensures that we place each vertex to some sub-graph H. The set of conditions (7) ensures that the total number of vertices assigned to the pattern H is divisible into parts of sizes σ or $\sigma + 1$. The set of conditions (8) ensures that each type that participates in a realisation of H contains at least x_H vertices, that is, we can assume that each realisation contains at least one vertex of each of its types. It is not difficult to verify that any solution to the Equitable Connected Partition problem fulfils (7)–(11).

In the opposite direction, suppose that we have an integral solution x satisfying (7)–(11). Let $\mathcal{H}$ be the collection of graphs H with multiplicities corresponding to x, that is, a graph H belongs to $\mathcal{H}$ exactly x_H-times. First, we observe that $|\mathcal{H}| = k$. To see this note that

$$|G| = \sum_{t \in H} n_t = \sum_{t \in H} \sum_{H \in \mathcal{H}} x_H^t = \sum_{H \in \mathcal{H}} \sum_{t \in H} x_H^t \ge \sum_{H \in \mathcal{H}} \sigma \cdot 1 \ge \sigma \sum_{H} x_H = \sigma |\mathcal{H}| \,.$$

Similarly, we have $|G| \le (\sigma + \xi_G)|\mathcal{H}|$ and the claim follows. Now, we find a realisation for every $H \in \mathcal{H}$. We know that there are $x_H^t \ge \sigma x_H$ vertices allocated to H. We assign them to the copies of H in $\mathcal{H}$ as follows. First, from each type $t \in H$ we assign one vertex to each copy of H (note that this is possible due to (8)). We assign the rest of the vertices greedily, so that there are σ vertices assigned to each copy of H; then, we assign the leftover vertices (note that there are at most x_H of them in total) to the different copies of H. In this way, we have assigned all vertices and gave a realisation of $\mathcal{H}$.

As was stated before, the integer linear program has only parameter-many variables. Hence, we can use the algorithm of Lenstra [49] to solve it in FPT time. ◀

With the algorithm from the proof of Theorem 14 in hand, we are ready to derive the result also for the modular-width.

▶ **Theorem 15.** *The* Equitable Connected Partition *problem is fixed-parameter tractable parameterised by the modular-width* $\mathrm{mw}(G)$*.*

Proof sketch. Clearly, the leaf-nodes of the modular decomposition of G have bounded neighbourhood diversity. For the graph of a leaf-node, we employ the following ILP:

$$\sigma x_H \le \sum_{t \in H} x_H^t \le (\sigma + \xi_G) x_H \qquad \forall H \tag{12}$$

$$x_H \le x_H^t \qquad \forall H\, \forall t \in H \tag{13}$$

$$\sum_{H} x_H^t \le n_t \qquad \forall t \in T(G) \tag{14}$$

$$0 \le x_H^t \le n_t,\; x_H^t \in \mathbb{Z} \qquad \forall H\, \forall t \in H \tag{15}$$

$$0 \le x_H,\; x_H^t \in \mathbb{Z} \qquad \forall H \tag{16}$$

Note the difference between (14) and (9); that is, this time we do not insist on assigning all vertices and can have some leftover vertices. We add an objective function

$$\max \sum_{H} \sum_{t \in H} x_H^t\,,$$

that is, we want to cover as many vertices as possible already in the corresponding leaf-node. Next, we observe that based on the solution of the above ILP, we can replace the leaf-node with a graph of neighbourhood diversity 2. In order to do so, we claim that if we replace the graph represented by the leaf-node by a disjoint union of a clique of size $\sum_H \sum_{t\in H} x_H^t$ and an independent set of size $\sum_H (n_t - \sum_{t\in H} x_H^t)$, then we do not change the answer to the Equitable Connected Partition problem. That is, the answer to the original graph was *yes* if and only if the answer is *yes* after we alter the leaf-node. The algorithm for modular-width then follows by a repeated application of the above ILP. ◀

As the last result of this section, we give an XP algorithm for another structural parameter called distance to cluster graph. The algorithm, in its core, is based on the same ideas as our FPT algorithm for distance to clique. Nevertheless, the number of types of vertices is no longer bounded only by a function of a parameter, and to partition the vertices that are not in the neighbourhood of the modulator vertices, we need to employ dynamic programming.

▶ **Theorem 16.** *The* Equitable Connected Partition *problem is* XP *parameterised by the distance to cluster graph* $\mathrm{dcg}(G)$*.*

4 Hardness Results

In this section, we complement our algorithmic upper-bounds from the previous section with matching hardness lower-bounds. The results from this section clearly show that no XP algorithm introduced in this paper can be improved to a fixed-parameter tractable one, or pushed to a more general parameter.

First, we observe that ECP is W[1]-hard with respect to the feedback-edge set number fes of G. This negatively resolves the question from the introduction of our paper. In fact, the actual statement shows an even stronger intractability result.

▶ **Theorem 17.** *The* Equitable Connected Partition *problem is* W[1]*-hard with respect to the path-width* $\mathrm{pw}(G)$*, the feedback-edge set number* $\mathrm{fes}(G)$*, and the number of parts* p *combined.*

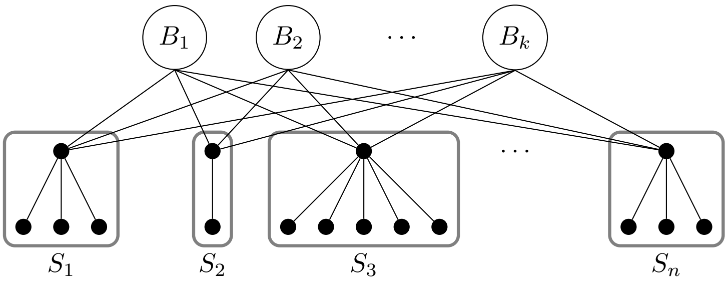


Figure 2 An illustration of the construction used to prove Theorem 19.

Note that the result from Theorem 17 can be, following the same arguments as used by [26], strengthened to show the same result for planar graphs.

▶ **Corollary 18.** *The* Equitable Connected Partition *problem is* W[1]*-hard when parameterised by the path-width, the feedback-edge set, and the number of parts combined, even if G is a planar graph.*

Next, we show that the d-pvcn parameter from Theorem 8 cannot be relaxed any more while the problem is kept tractable. Our reduction[2] is even more general and comes from the Unary Bin Packing problem, which is defined as follows.

Unary Bin Packing

Input: A number of bins k, a capacity of a single bin b, and a multi-set of integers $A = \{a_1, \ldots, a_n\}$ such that $\sum_{a\in A} a = bk$.

Question: Is there a surjective mapping $\alpha\colon A \to [k]$ such that for every $i \in [k]$ we have $\sum_{a\in\alpha^{-1}(i)} a = b$?

The Unary Bin Packing problem is well-known to be W[1]-hard when parameterised by the number of bins k and not solvable in $f(k) \cdot n^{o(k/\log k)}$ time for any computable function f, even if all numbers are given in unary [44].

▶ **Theorem 19.** *For every graph family $\mathcal{G}$ such that it contains at least one connected graph G with s vertices for every $s \in \mathbb{N}$, the* Equitable Connected Partition *problem is* W[1]*-hard parameterised by the distance to $\mathcal{G}$ referred to as $\mathrm{dist}_{\mathcal{G}}(G)$ and the number of parts p combined and, unless ETH fails, there is no algorithm running in $f(\ell) \cdot n^{o(\ell/\log \ell)}$ time for any computable function f, where $\ell = p + \mathrm{dist}_{\mathcal{G}}(G)$.*

Proof sketch. Let $\mathcal{I} = (A, k, b)$ be an instance of the Unary Bin Packing problem. We construct an equivalent instance $\mathcal{J} = (G, p)$ of the Equitable Connected Partition problem as follows (see Figure 2 for an overview of the construction). For the sake of exposition, we assume that $\mathcal{G}$ is a family containing all disjoint unions of stars; later we show how to tweak the construction to work with any $\mathcal{G}$ satisfying the conditions from the theorem statement.

For every number $a_i \in A$, we create a single *item-gadget* S_i which is a star with a_i vertices. Every S_i will be connected with the rest of the graph G only via the star centre c_i; we call this special vertex a *hub*. Next, we create k *bin-gadgets* $B_1, \ldots, B_k$. Each of these gadgets

[2] We would like to mention here that the construction used in the proof of Theorem 19 starts with the same problem and share similarities with the independent hardness construction of Gima and Otachi [39]; however, our construction is arguably easier and prove much more general hardness results.

consists of a single vertex. Slightly abusing the notation, we call this vertex also B_i, $i \in [k]$. As the last step of the construction, we add an edge connecting every bin-gadget with every item-gadget and set $p = k$. ◀

It is not hard to see that every graph G which is a disjoint union of stars has constant 4-path vertex cover number – 0, to be precise. Therefore, by Theorem 19 we obtain the desired hardness result.

▶ **Corollary 20.** *The* Equitable Connected Partition *problem is* W[1]*-hard parameterised by the* 4*-path vertex cover number 4-*pvcn *and the number of parts p combined.*

Now, a previous blind spot of our understanding of the Equitable Connected Partition problem's complexity with respect to the structural parameters that are bounded mostly for sparse graphs is the distance to disjoint paths. We again obtain hardness as a direct corollary of Theorem 19.

▶ **Corollary 21.** *The* Equitable Connected Partition *problem is* W[1]*-hard parameterised by the distance to disjoint paths* $\mathrm{ddp}(G)$ *and the number of parts p combined.*

Enciso et al. [26] stated (and we formalized in Theorem 9) that there is an XP algorithm for the Equitable Connected Partition problem parameterised by the tree-width of G. A natural question is then whether this algorithm can be improved to solve the problem in the same running-time also with respect to the more general parameter called clique-width. We give a strong evidence that such an algorithm is unlikely in the following theorem.

▶ **Theorem 22.** *The* Equitable Connected Partition *problem is* NP*-hard even if the graph G has clique-width* 3*, and is solvable in polynomial-time on graphs of clique-width at most* 2*.*

Proof. To show the hardness, we reuse the reduction used to prove Theorem 19. Recall that the construction can be done in polynomial time, thus, the reduction is also a polynomial reduction. What we need to show is that the clique-width of the constructed graph G is constant. We show this by providing an algebraic expression that uses 3 labels. First, we create a graph G_1 containing all bin-gadgets. This can be done by introducing a single vertex and by repeating disjoint union operation. We additionally assume that all vertices in G_1 have label 3. Next, we create a graph G_2 containing all item-gadgets. Every item-gadget is a star which can be constructed using two labels 1 and 2. Without loss of generality, we assume that all item-gadgets' centres have label 1 and all leaves have label 2. To complete the construction, we create disjoint union of G_1 and G_2 and, then, we perform a full join of vertices labelled 1 and 3. It is easy to see that the expression indeed leads to a desired graph. Polynomial-time solvability follows from Theorem 13 and the fact that co-graphs are exactly the graphs with clique-width at most 2 [17]. ◀

Using similar arguments, we can show para-NP-hardness also for a more restrictive parameter called shrub-depth [32, 34, 33].

▶ **Theorem 23.** *The* Equitable Connected Partition *problem is* NP*-hard even if the graph G has shrub-depth* 3*.*

As the last piece of the complexity picture of the Equitable Connected Partition problem, we show that ECP is W[1]-hard with respect to the distance to disjoint cliques. Recall that we give an XP algorithm for this parameter in Theorem 16.

▶ **Corollary 24.** *The* Equitable Connected Partition *problem is* W[1]*-hard with respect to the distance to cluster graph* $\mathrm{dcg}(G)$ *and the number of parts p combined.*

5 Conclusions

We revisit the complexity picture of the Equitable Connected Partition problem with respect to various structural restrictions of the graph. We complement the existing results with algorithmic upper-bounds and corresponding complexity lower-bounds that clearly show that no existing parameterised algorithm can be significantly improved.

Despite that the provided complexity study gives us a clear dichotomy between tractable and intractable cases, there still remain a few blind spots. One of the most interesting is the complexity classification of the Equitable Connected Partition problem with respect to the band-width parameter, which lies between the maximum leaf number and the path-width; however, is incomparable with the feedback-edge set number.

An interesting line of research can target the tightness of our results. For example, we show a clear dichotomy between the tractable and intractable cases of ECP when parameterised by the clique-width. Using similar arguments, we can show the following result for the recently introduced parameter twin-width [11].

▶ **Theorem 25.** *The Equitable Connected Partition problem is* NP*-hard even if the graph G has twin-width 2, and is solvable in polynomial-time on graphs of twin-width 0.*

We conjecture that the problem is polynomial-time solvable also on graphs of twin-width 1, which, unlike the twin-width 2 graphs, are additionally known to be recognisable efficiently [10]. Similarly, we can ask whether the provided FPT algorithms are optimal under some standard theoretical assumptions, such as the well-known Exponential-Time Hypothesis [42].

Last but not least, the parameterised complexity framework not only gives us formal tools for finer-grained complexity analysis of algorithms for NP-hard problems, but, at the same time, equips us also with the necessary formalism for analysis of effective preprocessing, which is widely known as *kernelisation*. A natural follow-up question is then whether the Equitable Connected Partition problem admits a polynomial kernel with respect to any of the studied structural parameters. We conjecture that there is a polynomial kernel with respect to the distance to clique and 3-path vertex cover number.

References

1 Akanksha Agrawal and M. S. Ramanujan. Distance from triviality 2.0: Hybrid parameterizations. In Cristina Bazgan and Henning Fernau, editors, *Proceedings of the 33rd International Workshop on Combinatorial Algorithms, IWOCA '22*, volume 13270 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2022. doi:10.1007/978-3-031-06678-8_1.

2 Micah Altman. The computational complexity of automated redistricting: Is automation the answer? *Rutgers Computer and Law Technology Journal*, 23(1):81–142, March 1997.

3 Kateřina Altmanová, Dušan Knop, and Martin Koutecký. Evaluating and tuning n-fold integer programming. *ACM Journal of Experimental Algorithmics*, 24(2), July 2019. doi:10.1145/3330137.

4 Konstantin Andreev and Harald Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, November 2006. doi:10.1007/s00224-006-1350-7.

5 Peter Arbenz, G. Harry van Lenthe, Uche Mennel, Ralph Müller, and Marzio Sala. Multi-level μ-finite element analysis for human bone structures. In Bo Kågström, Erik Elmroth, Jack Dongarra, and Jerzy Waśniewski, editors, *Proceedings of the 8th International Workshop on Applied Parallel Computing, PARA '06*, volume 4699 of *Lecture Notes in Computer Science*, pages 240–250. Springer, 2007. doi:10.1007/978-3-540-75755-9_30.

6 René van Bevern, Andreas Emil Feldmann, Manuel Sorge, and Ondřej Suchý. On the parameterized complexity of computing balanced partitions in graphs. *Theory of Computing Systems*, 57:1–35, July 2015. doi:10.1007/s00224-014-9557-5.

7 Sandeep N. Bhatt and Frank Thomson Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, April 1984. doi:10.1016/0022-0000(84)90071-0.

8 Václav Blažej, Robert Ganian, Dušan Knop, Jan Pokorný, Šimon Schierreich, and Kirill Simonov. The parameterized complexity of network microaggregation. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI '23*, pages 6262–6270. AAAI Press, 2023. doi:10.1609/aaai.v37i5.25771.

9 Hans L. Bodlaender and Fedor V. Fomin. Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, 349(1):22–30, December 2005. doi:10.1016/j.tcs.2005.09.027.

10 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022. doi:10.1007/S00453-022-00965-5.

11 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO model checking. *Journal of the ACM*, 69(1), 2022. doi:10.1145/3486655.

12 Robert Bredereck, Andrzej Kaczmarczyk, Dušan Knop, and Rolf Niedermeier. High-multiplicity fair allocation: Lenstra empowered by N-fold integer programming. In Anna Karlin, Nicole Immorlica, and Ramesh Johari, editors, *Proceedings of the 20th ACM Conference on Economics and Computation, EC '19*, pages 505–523. ACM, 2019. doi:10.1145/3328526.3329649.

13 Thang Nguyen Bui and Andrew Peck. Partitioning planar graphs. *SIAM Journal on Computing*, 21(2):203–215, 1992. doi:10.1137/0221016.

14 Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, pages 117–158. Springer, 2016. doi:10.1007/978-3-319-49487-6_4.

15 Bor-Liang Chen, Ming-Tat Ko, and Ko-Wei Lih. Equitable and m-bounded coloring of split graphs. In Michel Deza, Reinhardt Euler, and Ioannis Manoussakis, editors, *Combinatorics and Computer Science, CCS '95*, volume 1120 of *Lecture Notes in Computer Science*, pages 1–5. Springer, 1996.

16 Bor-Liang Chen and Ko-Wei Lih. Equitable coloring of trees. *Journal of Combinatorial Theory, Series B*, 61(1):83–87, May 1994.

17 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

18 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, Cham, 2015. doi:10.1007/978-3-319-21275-3.

19 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019. doi:10.1137/140988553.

20 Argyrios Deligkas, Eduard Eiben, Robert Ganian, Thekla Hamm, and Sebastian Ordyniak. The parameterized complexity of connected fair division. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI '21*, pages 139–145. International Joint Conferences on Artificial Intelligence Organization, August 2021. Main Track. doi:10.24963/ijcai.2021/20.

21 Reinhard Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, Berlin, Heidelberg, 5th edition, 2017. doi:10.1007/978-3-662-53622-3.

22 Martin Doucha and Jan Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science, MFCS '12*, volume 7464 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2012. doi:10.1007/978-3-642-32589-2_32.

23 M. E. Dyer and A. M. Frieze. A partitioning algorithm for minimum weighted Euclidean matching. *Information Processing Letters*, 18(2):59–62, 1984. doi:10.1016/0020-0190(84)90024-3.

24 Josep Díaz and George B. Mertzios. Minimum bisection is NP-hard on unit disk graphs. *Information and Computation*, 256:83–92, 2017. doi:10.1016/j.ic.2017.04.010.

25 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *CoRR*, abs/1904.01361, 2019. arXiv:1904.01361.

26 Rosa Enciso, Michael R. Fellows, Jiong Guo, Iyad Kanj, Frances Rosamond, and Ondřej Suchý. What makes equitable connected partition easy. In Jianer Chen and Fedor V. Fomin, editors, *Proceedings of the 4th International Workshop on Parameterized and Exact Computation, IWPEC '09*, volume 5917 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2009.

27 Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, February 2015. doi:10.1007/s00453-013-9802-3.

28 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. *Information and Computation*, 209(2):143–153, 2011. doi:10.1016/j.ic.2010.11.026.

29 Jiří Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theoretical Computer Science*, 412(23):2513–2523, 2011. doi:10.1016/j.tcs.2010.10.043.

30 Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Almost optimal lower bounds for problems parameterized by clique-width. *SIAM Journal on Computing*, 43(5):1541–1563, 2014. doi:10.1137/130910932.

31 H. Furmańczyk and M. Kubale. The complexity of equitable vertex coloring of graphs. *Journal of Applied Computer Science*, 13(2):95–106, 2005.

32 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In Gregory Gutin and Stefan Szeider, editors, *Proceedings of the 8th International Symposium on Parameterized and Exact Computation, IPEC '13*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013.

33 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona de Mendez. Shrub-depth: Capturing height of dense graphs. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:7)2019.

34 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science, MFCS '12*, volume 7464 of *Lecture Notes in Computer Science*, pages 419–430. Springer, 2012. doi:10.1007/978-3-642-32589-2_38.

35 Robert Ganian and Jan Obdržálek. Expanding the expressive power of monadic second-order logic on restricted graph classes. In Thierry Lecroq and Laurent Mouchard, editors, *Proceedings of the 24th International Workshop on Combinatorial Algorithms, IWOCA '13*, volume 8288 of *Lecture Notes in Computer Science*, pages 164–177. Springer, 2013. doi:10.1007/978-3-642-45278-9_15.

36 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

37 Tomáš Gavenčiak, Martin Koutecký, and Dušan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44:100596, 2022. Optimization and Discrete Geometry. doi:10.1016/j.disopt.2020.100596.

38 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theoretical Computer Science*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.

39 Tatsuya Gima and Yota Otachi. Extended MSO model checking via small vertex integrity. *Algorithmica*, 86(1):147–170, 2024. doi:10.1007/S00453-023-01161-9.

40 Guilherme C. M. Gomes, Matheus R. Guedes, and Vinicius F. dos Santos. Structural parameterizations for equitable coloring: Complexity, FPT algorithms, and kernelization. *Algorithmica*, 85:1912–1947, July 2023. doi:10.1007/s00453-022-01085-w.

41 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In Rod Downey, Michael Fellows, and Frank Dehne, editors, *Proceedings of the 1st International Workshop on Parameterized and Exact Computation, IWPEC '04*, pages 162–173. Springer, 2004.

42 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.

43 Takehiro Ito, Xiao Zhou, and Takao Nishizeki. Partitioning a graph of bounded tree-width to connected subgraphs of almost uniform size. *Journal of Discrete Algorithms*, 4(1):142–154, 2006. doi:10.1016/j.jda.2005.01.005.

44 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *Journal of Computer and System Sciences*, 79(1):39–49, 2013. doi:10.1016/j.jcss.2012.04.004.

45 Dušan Knop, Martin Koutecký, Asaf Levin, Matthias Mnich, and Shmuel Onn. High-multiplicity N-fold IP via configuration LP. *Mathematical Programming*, 200(1):199–227, 2023. doi:10.1007/s10107-022-01882-9.

46 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial N-fold integer programming and applications. *Mathematical Programming*, 184(1):1–34, 2020. doi:10.1007/s10107-019-01402-2.

47 Dušan Knop, Šimon Schierreich, and Ondřej Suchý. Balancing the spread of two opinions in sparse social networks (student abstract). In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, AAAI '22*, pages 12987–12988. AAAI Press, 2022. doi:10.1609/aaai.v36i11.21630.

48 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, September 2012. doi:10.1007/s00453-011-9554-x.

49 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4), 1983. doi:10.1287/moor.8.4.538.

50 Harry A. Levin and Sorelle A. Friedler. Automated congressional redistricting. *ACM Journal of Experimental Algorithmics*, 24, April 2019. doi:10.1145/3316513.

51 Ko-Wei Lih. Equitable coloring of graphs. In Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 1199–1248. Springer, 2013. doi:10.1007/978-1-4419-7997-1_25.

52 Mario Lucertini, Yehoshua Perl, and Bruno Simeone. Most uniform path partitioning and its use in image processing. *Discrete Applied Mathematics*, 42(2):227–256, 1993. doi:10.1016/0166-218X(93)90048-S.

53 Kitty Meeks and Fiona Skerman. The parameterised complexity of computing the maximum modularity of a graph. *Algorithmica*, 82(8):2174–2199, August 2020. doi:10.1007/s00453-019-00649-7.

54 Manfred Wiegers. The k-section of treewidth restricted graphs. In Branislav Rovan, editor, *Proceedings of the 15th International Symposium on Mathematical Foundations of Computer Science, MFCS '90*, volume 452 of *Lecture Notes in Computer Science*, pages 530–537. Springer, 1990.

55 Justin C. Williams Jr. Political redistricting: A review. *Papers in Regional Science*, 74(1), 1995. doi:10.1111/j.1435-5597.1995.tb00626.x.

When Lawvere Meets Peirce: An Equational Presentation of Boolean Hyperdoctrines

Filippo Bonchi
University of Pisa, Italy

Alessandro Di Giorgio
University College London, UK

Davide Trotta
University of Padova, Italy

Abstract

Fo-bicategories are a categorification of Peirce's calculus of relations. Notably, their laws provide a proof system for first-order logic that is both purely equational and complete. This paper illustrates a correspondence between fo-bicategories and Lawvere's hyperdoctrines. To streamline our proof, we introduce peircean bicategories, which offer a more succinct characterization of fo-bicategories.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Categorical semantics

Keywords and phrases relational algebra, hyperdoctrines, cartesian bicategories, string diagrams

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.30

Related Version *Full Version*: https://arxiv.org/abs/2404.18795 [5]

Funding *Filippo Bonchi*: Supported by MUR project PRIN 2022 PNRR No. P2022HXNSC – RAP (Resource Awareness in Programming) and University of Pisa project PRA_2022_99 – FM4HD.
Alessandro Di Giorgio: Supported by EPSRC project EP/V002376/1 – Nominal String Diagrams.

1 Introduction

The first appearances of the characteristic features of first-order logic can be traced back to the works of Peirce [54] and Frege [21]. Frege was mainly motivated by the pursuit of a rigorous foundation for mathematics: his work was inspired by real analysis, bringing the concept of functions and variables into the logical realm [18]. On the other hand Peirce, inspired by the work of De Morgan [16] on relational reasoning, introduced a calculus in which operations allow the combination of relations and adhere to a set of algebraic laws. Like Boole's algebra of classes [9], Peirce's calculus of relations does not feature variables nor quantifiers and its sole deduction rule is substituting equals by equals.

Despite several negative results [51, 28, 63, 22, 2, 60] regarding axiomatizations for the calculus, its lack of binder-related complexities, coupled with purely equational proofs, has rendered the calculus of relations highly influential in computer science, e.g., in the context of database theory [13], programming languages [61, 27, 38, 1, 37] and proof assistants [58, 59, 36]. In logic, the calculus played a secondary role for many years, likely because it is strictly less expressive than first-order logic [43]. This was until Tarski in [67] recognized its algebraic flavour and initiated a program of algebraizing first-order logic, including works such as [17, 26, 62]. Quoting Quine [62]:

> "Logic in his adolescent phase was algebraic. There was Boole's algebra of classes and Peirce's algebra of relations. But in 1879 logic come of age, with Frege's quantification theory. Here the bound variables, so characteristic of analysis rather than of algebra, became central to logic."

Such a perspective, which regarded algebraic aspects and those concerning quantifiers as separate entities, changed with the work of Lawvere.

Thanks to the recent development of a new branch of mathematics, namely category theory, Lawvere introduced in [40, 41, 42] *hyperdoctrines* which enabled the study of logic from a pure algebraic perspective. The crucial insights of Lawvere was to show that quantifiers, as well as many logical constructs, can be algebraically captured through the crucial notion of adjointness. Hyperdoctrines, along with many categorical structures related to logics, such as regular, Heyting, and boolean categories [32, 33], align with Frege's functional perspective: arrows represent functions (terms), and relations are derived through specific constructions.

In the last decade, the paradigm shift towards treating data as a physical resource has motivated many computer scientists to move from traditional term-based (cartesian) syntax toward a string diagrammatic (monoidal) syntax [34, 65] (see e.g., [66, 4, 6, 8, 14, 19, 20, 24, 52, 56]). This shift in syntax enables an extension of Peirce's calculus of relations that is as expressive as first-order logic, accompanied by an axiomatization that is purely equational and complete. The axioms are those of *first-order bicategories* [3]: see Figures 1, 3 and 4. In essence, a first-order bicategory, or fo-bicategory, encompasses a cartesian and a cocartesian bicategory [11], interacting as a linear bicategory [12], while additionally satisfying linear versions of Frobenius equations and adjointness conditions.

In this paper, we reconcile Lawvere's understanding of logic with Peirce's calculus of relations by illustrating a formal correspondence between boolean hyperdoctrines and first-order bicategories.

To reach such a correspondence, we found convenient to introduce *peircean bicategories*: these are cartesian bicategories with each homset carrying a boolean algebra where the negation behaves appropriately with *maps* – special arrows that intuitively generalize functions. Our first result (Theorem 27) states that peircean and fo-bicategories are equivalent.

While the definition of peircean bicategories is not purely equational, as in the case of fo-bicategories, it is notably more concise. Moreover, it allows us to reuse from [7] an adjunction between cartesian bicategories and *elementary and existential doctrines* [46, 45, 47], which are a generalisation of hyperdoctrines, corresponding to the $(\exists, =, \top, \wedge)$-fragment of first-order logic. Our main result (Theorem 32) reveals an adjunction between the category of first-order bicategories and the category of boolean hyperdoctrines.

It is essential to note that our theorem establishes an adjunction rather than an equivalence. The discrepancy can be intuitively explained by noting that, akin to first-order logic, terms and formulas are distinct entities in hyperdoctrines. Thus for two terms t_1 and t_2, the hyperdoctrine where the formula $t_1 = t_2$ is true differs from the hyperdoctrine where t_1 and t_2 are equated as terms, a distinction not present in fo-bicategories. These issues, related to the extensionality of equality, are thoroughly analyzed in the literature (see e.g. [45, 31]).

Leveraging another result from [7], we demonstrate (Theorem 37) that the adjunction in Theorem 32 becomes an equivalence when restricted to well-behaved hyperdoctrines (i.e., those whose equality is extensional and satisfying the rule of unique choice [44]).

Synopsis. In § 2, we provide a review of (co)cartesian, linear and fo-bicategories. § 3 covers a recap of elementary and existential doctrines and boolean hyperdoctrines. The key adjunction from [7] is recalled in §4. Our original contributions commence in § 5, where we introduce peircean bicategories and establish their equivalence with fo-bicategories. This result is used in § 6 to show the adjunction and in § 7 to establish the equivalence. Missing proofs can be found in [5].

Terminology and Notation. All bicategories considered in this paper are just poset-enriched symmetric monoidal categories. For a bicategory $\mathbf{C}$, we will write $\mathbf{C}^{\mathrm{op}}$ for the bicategory having the same objects as $\mathbf{C}$ but homsets $\mathbf{C}^{\mathrm{op}}[X,Y] \stackrel{\mathrm{def}}{=} \mathbf{C}[Y,X]$. Similarly, we will write $\mathbf{C}^{\mathrm{co}}$ to denote the bicategory having the same objects and arrows of $\mathbf{C}$ but equipped with the reversed ordering $\geq$. The cartesian bicategories in this paper are called in [11] cartesian bicategories of relations. We refer the reader to [3, Rem. 2] for a comparison with the presentation of linear bicategories in [12]. In a category with finite products, we write $\langle f, g\rangle$ for the pairing of f and g and Δ_X for $\langle id^{\circ}_X, id^{\circ}_X\rangle$.

2 From (Co)Cartesian to First-Order Bicategories

In this section we recall the notion of *first-order bicategory* from [3]. To provide a preliminary intuition, it is convenient to consider $\mathbf{Rel}$, the first-order bicategory of sets and relations.

It is well known that sets and relations form a symmetric monoidal category, hereafter denoted as $\mathbf{Rel}^{\circ}$, with composition, identities, monoidal product and symmetries defined as

$$\begin{aligned}
a \overset{\circ}{,} b &\stackrel{\mathrm{def}}{=} \{(x,z) \mid \exists y \in Y \,.\, (x,y) \in a \wedge (y,z) \in b\} \subseteq X \times Z \qquad id^{\circ}_X \stackrel{\mathrm{def}}{=} \{(x,y) \mid x = y\} \subseteq X \times X \\
a \otimes c &\stackrel{\mathrm{def}}{=} \{(\,(x,z),(y,v)\,) \mid (x,y) \in a \wedge (z,v) \in c\} \subseteq (X \times Z) \times (Y \times V) \\
\sigma^{\circ}_{X,Y} &\stackrel{\mathrm{def}}{=} \{(\,(x,y),(y',x')\,) \mid x = x' \wedge y = y'\} \subseteq (X \times Y) \times (Y \times X)
\end{aligned} \tag{1}$$

for all sets X, Y, Z, V and relations $a \subseteq X \times Y$, $b \subseteq Y \times Z$ and $c \subseteq Z \times V$. As originally observed by Peirce in [55], beyond $\overset{\circ}{,}$ there exists another form of relational composition that enjoys noteworthy algebraic properties. This different composition gives rise to another symmetric monoidal category of sets and relations, hereafter denoted by $\mathbf{Rel}^{\bullet}$ and defined as follows.

$$\begin{aligned}
a \overset{\bullet}{,} b &\stackrel{\mathrm{def}}{=} \{(x,z) \mid \forall y \in Y \,.\, (x,y) \in a \vee (y,z) \in b\} \subseteq X \times Z \qquad id^{\bullet}_X \stackrel{\mathrm{def}}{=} \{(x,y) \mid x \neq y\} \subseteq X \times X \\
a \boldsymbol{\otimes} c &\stackrel{\mathrm{def}}{=} \{(\,(x,z),(y,v)\,) \mid (x,y) \in a \vee (z,v) \in c\} \subseteq (X \times Z) \times (Y \times V) \\
\sigma^{\bullet}_{X,Y} &\stackrel{\mathrm{def}}{=} \{(\,(x,y),(y',x')\,) \mid x \neq x' \vee y \neq y'\} \subseteq (X \times Y) \times (Y \times X)
\end{aligned} \tag{2}$$

Note that $\otimes$ and $\boldsymbol{\otimes}$ are both defined on objects as the cartesian product of sets and have as unit the singleton set $I \stackrel{\mathrm{def}}{=} \{\star\}$. Both $\mathbf{Rel}^{\circ}$ and $\mathbf{Rel}^{\bullet}$ are poset-enriched symmetric monoidal categories when taking as ordering the inclusion $\subseteq$ and the complement $\neg\colon (\mathbf{Rel}^{\circ})^{\mathrm{co}} \to \mathbf{Rel}^{\bullet}$ is an isomorphism. As we will explain in § 2.1, the relations defined for all sets X as

$$\begin{aligned}
\blacktriangleleft^{\circ}_X &\stackrel{\mathrm{def}}{=} \{(x,\ (y,z)) \mid x = y \wedge x = z\} \subseteq X \times (X \times X) &\quad \blacktriangleleft^{\bullet}_X &\stackrel{\mathrm{def}}{=} \{(x,\ (y,z)) \mid x \neq y \vee x \neq z\} \subseteq X \times (X \times X) \\
\blacktriangleright^{\circ}_X &\stackrel{\mathrm{def}}{=} \{((y,z),\ x) \mid x = y \wedge x = z\} \subseteq (X \times X) \times X &\quad \blacktriangleright^{\bullet}_X &\stackrel{\mathrm{def}}{=} \{((y,z),\ x) \mid x \neq y \vee x \neq z\} \subseteq (X \times X) \times X \\
!^{\circ}_X &\stackrel{\mathrm{def}}{=} \{(x,\star) \mid x \in X\} \subseteq X \times I &\quad !^{\bullet}_X &\stackrel{\mathrm{def}}{=} \varnothing \subseteq X \times I \\
¡^{\circ}_X &\stackrel{\mathrm{def}}{=} \{(\star,x) \mid x \in X\} \subseteq I \times X &\quad ¡^{\bullet}_X &\stackrel{\mathrm{def}}{=} \varnothing \subseteq I \times X
\end{aligned} \tag{3}$$

make $\mathbf{Rel}^{\circ}$ a cartesian bicategory, while $\mathbf{Rel}^{\bullet}$ a cocartesian one.

Intuitively, a first-order bicategory $\mathbf{C}$ consists of a cartesian bicategory $\mathbf{C}^{\circ}$, called the "white structure", and a cocartesian bicategory $\mathbf{C}^{\bullet}$, called the "black structure", that interact by obeying the same laws of $\mathbf{Rel}^{\circ}$ and $\mathbf{Rel}^{\bullet}$. The name "first-order" is due to the fact that such laws provide a complete system of axioms for first-order logic.

The axioms can be conveniently given by means of a graphical representation inspired by string diagrams [34, 65]: composition is depicted as horizontal composition while the monoidal product by vertically "stacking" diagrams. However, since there are two compositions $\overset{\circ}{,}$ and

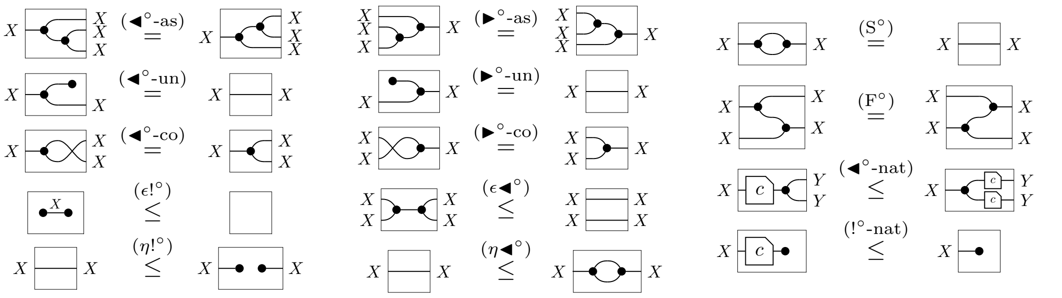
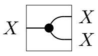
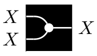


Figure 1 Axioms of cartesian bicategories.

$⨾$ and two monoidal products $\otimes$ and $\boldsymbol{\otimes}$, to distinguish them we use different colors. All white constants have white background, mutatis mutandis for the black ones: for instance $\blacktriangleleft^{\circ}_{X}$ and $\blacktriangleright^{\bullet}_{X}$ are drawn  and  , while for some arrows a, b, c, d of the appropriate type, $(a \otimes c) ⨾ (b \boldsymbol{\otimes} d)$ is drawn as on the right of (ν°_{l}) in Figure 3.

2.1 (Co)Cartesian Bicategories

We commence with the notion of cartesian bicategories by Carboni and Walters [11].

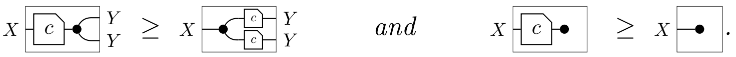
▶ **Definition 1.** *A* cartesian bicategory $(\mathbf{C}, \otimes, I, \blacktriangleleft^{\circ}, !^{\circ}, \blacktriangleright^{\circ}, ¡^{\circ})$, *shorthand* $(\mathbf{C}, \blacktriangleleft^{\circ}, \blacktriangleright^{\circ})$, *is a poset-enriched symmetric monoidal category* $(\mathbf{C}, \otimes, I)$ *and, for every object* X *in* $\mathbf{C}$, *arrows* $\blacktriangleleft^{\circ}_{X} : X \to X \otimes X$, $!^{\circ}_{X} : X \to I$, $\blacktriangleright^{\circ}_{X} : X \otimes X \to X$, $¡^{\circ}_{X} : I \to X$ *such that*

1. $(\blacktriangleleft^{\circ}_{X}, !^{\circ}_{X})$ *is a comonoid and* $(\blacktriangleright^{\circ}_{X}, ¡^{\circ}_{X})$ *a monoid, i.e., the equalities* (◀°-as), (◀°-un), (◀°-co) *and* (▶°-as), (▶°-un), (▶°-co) *in Figure 1 hold;*
2. *every arrow* $c \colon X \to Y$ *is a lax comonoid homomorphism, i.e.,* (◀°-nat) *and* (!°-nat) *hold;*
3. *comonoids are left adjoints to the monoids, i.e.,* $(\eta \blacktriangleleft^{\circ})$, $(\epsilon \blacktriangleleft^{\circ})$, $(\eta !^{\circ})$ *and* $(\epsilon !^{\circ})$ *hold;*
4. *monoids and comonoids form special Frobenius bimonoids, i.e.,* (F°) *and* (S°) *hold;*
5. *monoids and comonoids satisfy the expected coherence conditions (see e.g. [7]).*

$\mathbf{C}$ *is a* cocartesian bicategory *if* $\mathbf{C}^{\mathrm{co}}$ *is a cartesian bicategory. A* morphism of (co)cartesian bicategories *is a poset-enriched strong symmetric monoidal functor preserving monoids and comonoids. We denote by* $\mathbb{CB}$ *the category of cartesian bicategories and their morphisms.*

As already mentioned, $\mathbf{Rel}^{\circ}$ with $\blacktriangleleft^{\circ}_{X}$, $!^{\circ}_{X}$, $\blacktriangleright^{\circ}_{X}$ and $¡^{\circ}_{X}$ defined in (3) form a cartesian bicategory: the reader can easily check, using the definitions in (1) and (3), that all the laws in Figure 1 are satisfied. Similarly, one can observe that the opposite inequality of (◀°-nat) holds iff the relation $c \subseteq X \times Y$ is single-valued (i.e., deterministic), while the opposite of (!°-nat) iff c is total. In other words, c is a function iff both (◀°-nat) and (!°-nat) hold as equalities.

▶ **Definition 2.** *Let* $c \colon X \to Y$ *be an arrow of a cartesian bicategory* $\mathbf{C}$. *It is a* map *if*



(4)

Maps form a monoidal subcategory of $\mathbf{C}$, denoted by $\mathsf{Map}(\mathbf{C})$, that has finite products [11].

In a cartesian bicategory $\mathbf{C}$, each homset $\mathbf{C}[X, Y]$ carries the structure of inf-semilattice, defined for all $c, d \colon X \to Y$ as in (5) below. Furthermore, the equation (6) defines an identity-on-objects isomorphism of cartesian bicategories $(\cdot)^{\dagger} \colon \mathbf{C} \to \mathbf{C}^{\mathrm{op}}$.

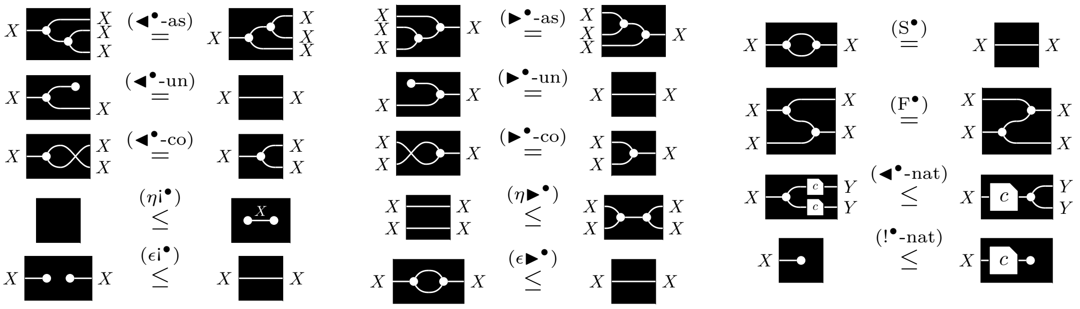
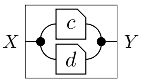
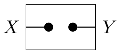
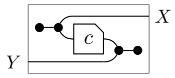


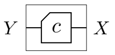
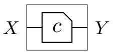
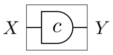
Figure 2 Axioms of cocartesian bicategories.

$c \wedge d \overset{\text{def}}{=}$  $\top \overset{\text{def}}{=}$  (5) $c^\dagger \overset{\text{def}}{=}$  (6)

The reader can check, using (1) and (3) that in $\mathbf{Rel}^\circ$, $c^\dagger\colon Y \to X$ is the opposite of the relation c, namely $\{(y,x) \mid (x,y) \in c\}$. It is well known that a relation c is a function iff it is left adjoint to $c^\dagger$. More generally in a cartesian bicategory c is a map iff it is left adjoint to $c^\dagger$. Summarising:

▶ **Proposition 3.** *Let* $\mathbf{C}$ *be a cartesian bicategory and* $c\colon X \to Y$ *an arrow of* $\mathbf{C}$. *The following hold:*

1. *every homset carries the inf-semilattice structure, defined as in* (5)*;*
2. *there is an isomorphism of cartesian bicategories* $(\cdot)^\dagger\colon \mathbf{C} \to \mathbf{C}^{\mathrm{op}}$*, defined as in* (6)*;*
3. c *is a map iff* c *is left adjoint to* $c^\dagger$*;*
4. $\mathsf{Map}(\mathbf{C})$ *is a category with finite products; moreover, a morphism of cartesian bicategories* $F\colon \mathbf{C} \to \mathbf{D}$ *restricts to a functor* $\tilde{F}\colon \mathsf{Map}(\mathbf{C}) \to \mathsf{Map}(\mathbf{D})$ *preserving finite products.*

Hereafter, we draw  for ( $)^\dagger$ and  for a map $c\colon X \to Y$.

We mentioned that $\mathbf{Rel}^\bullet$ with $\blacktriangleleft^\bullet_X$, $!^\bullet_X$, $\blacktriangleright^\bullet_X$ and $¡^\bullet_X$ defined in (3) forms a cocartesian bicategory. To prove this, it is enough to observe that the complement $\neg$ is a poset-enriched symmetric monoidal isomorphism $\neg\colon (\mathbf{Rel}^\circ)^{\mathrm{co}} \to \mathbf{Rel}^\bullet$ preserving (co)monoids.

2.2 Linear Bicategories

We have seen that $\mathbf{Rel}^\circ$ forms a cartesian bicategory, and $\mathbf{Rel}^\bullet$ a cocartesian bicategory. The next step consists of merging them into one entity and studying their algebraic interactions. However, the coexistence of two different compositions $\mathbin{⨾}$ and $\mathbin{\bullet\!\!⨾}$ on the same class of objects and arrows brings us out of the realm of ordinary categories. The appropriate setting is provided by *linear bicategories* [12] by Cockett, Koslowski and Seely.

▶ **Definition 4.** *A* linear bicategory $(\mathbf{C}, ⨾, id^\circ, \bullet⨾, id^\bullet)$ *consists of two poset-enriched categories* $(\mathbf{C}, ⨾, id^\circ)$ *and* $(\mathbf{C}, \bullet⨾, id^\bullet)$ *with the same class of objects, arrows and orderings (but possibly different identities and compositions) such that* $⨾$ *linearly distributes over* $\bullet⨾$*, i.e.,* (δ_l) *and* (δ_r) *in Figure 3 hold.*

A symmetric monoidal linear bicategory $(\mathbf{C}, ⨾, id^\circ, \bullet⨾, id^\bullet, \otimes, \sigma^\circ, \blacksquare\!\!\!\otimes, \sigma^\bullet, I)$*, shortly* $(\mathbf{C}, \otimes, \blacksquare\!\!\!\otimes, I)$*, consists of a linear bicategory* $(\mathbf{C}, ⨾, id^\bullet, \bullet⨾, id^\bullet)$ *and two poset-enriched symmetric monoidal categories* $(\mathbf{C}, \otimes, I)$ *and* $(\mathbf{C}, \blacksquare\!\!\!\otimes, I)$ *s.t.* $\otimes$ *and* $\blacksquare\!\!\!\otimes$ *agree on objects, i.e.,* $X \otimes Y = X \blacksquare\!\!\!\otimes Y$*, share the same unit* I *and*

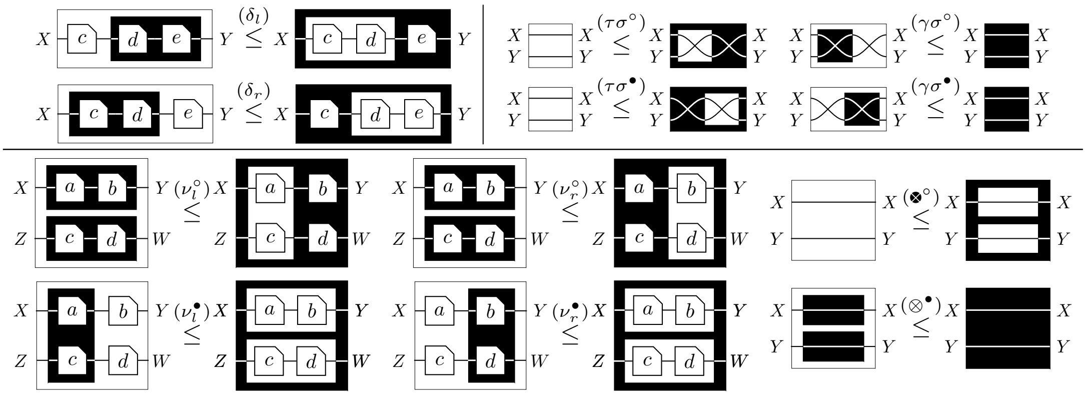


Figure 3 Axioms of closed symmetric monoidal linear bicategories.

2. *there are linear strengths for $(\otimes, \boldsymbol{\otimes})$, i.e., the inequalities (ν_l°), (ν_r°), $(\nu_l^{\bullet})$ and $(\nu_r^{\bullet})$ hold;*
3. *$\boldsymbol{\otimes}$ preserves id° colaxly and $\otimes$ preserves $id^{\bullet}$ laxly, i.e., $(\otimes^{\bullet})$ and $(\boldsymbol{\otimes}^{\circ})$ hold.*

A morphism of symmetric monoidal linear bicategories *$F\colon (\mathbf{C_1}, \otimes, \boldsymbol{\otimes}, I) \to (\mathbf{C_2}, \otimes, \boldsymbol{\otimes}, I)$ consists of two poset-enriched symmetric monoidal functors $F^{\circ}\colon (\mathbf{C_1}, \otimes, I) \to (\mathbf{C_2}, \otimes, I)$ and $F^{\bullet}\colon (\mathbf{C_1}, \boldsymbol{\otimes}, I) \to (\mathbf{C_2}, \boldsymbol{\otimes}, I)$ that agree on objects and arrows: $F^{\circ}(X) = F^{\bullet}(X)$ and $F^{\circ}(c) = F^{\bullet}(c)$.*

We omit the adjective *symmetric monoidal*, since all linear bicategories in this paper are such. In linear bicategories one can define *linear* adjoints: for $a\colon X \to Y$ and $b\colon Y \to X$, a is *left linear adjoint* to b, or b is *right linear adjoint* to a, written $b \Vdash a$, if $id_X^{\circ} \leq a \mathbin{;^{\bullet}} b$ and $b \mathbin{;^{\circ}} a \leq id_Y^{\bullet}$.

▶ **Definition 5.** *A linear bicategory $(\mathbf{C}, \otimes, \boldsymbol{\otimes}, I)$ is said to be* closed *if every $a\colon X \to Y$ has both a left and a right linear adjoint and, in particular, the white symmetry σ° is both left and right linear adjoint to the black symmetry $\sigma^{\bullet}$ ($\sigma^{\bullet} \Vdash \sigma^{\circ} \Vdash \sigma^{\bullet}$), i.e. $(\tau\sigma^{\circ})$, $(\gamma\sigma^{\circ})$, $(\tau\sigma^{\bullet})$ and $(\gamma\sigma^{\bullet})$ in Figure 3 hold.*

Our main example is the closed linear bicategory **Rel** of sets and relations. The white structure is the symmetric monoidal category $\mathbf{Rel}^{\circ}$ and the black structure is $\mathbf{Rel}^{\bullet}$. Observe that the two have the same objects, arrows and ordering. The white and black monoidal products $\otimes$ and $\boldsymbol{\otimes}$ agree on objects (they are the cartesian product of sets) and have common unit object (the singleton set I). By (1) and (2), one can easily check all the inequalities in Figure 3. Both left and right linear adjoints of a relation $c \subseteq X \times Y$ are given by $\neg c^{\dagger}$.

2.3 First-Order Bicategories

After (co)cartesian and linear bicategories, we can recall first-order bicategories from [3].

▶ **Definition 6.** *A* first-order bicategory $\mathbf{C}$ *consists of a closed linear bicategory $(\mathbf{C}, \otimes, \boldsymbol{\otimes}, I)$, a cartesian bicategory $(\mathbf{C}, \blacktriangleleft^{\circ}, \blacktriangleright^{\circ})$ and a cocartesian bicategory $(\mathbf{C}, \blacktriangleleft^{\bullet}, \blacktriangleright^{\bullet})$, such that*

1. *the white comonoid $(\blacktriangleleft^{\circ}, !^{\circ})$ is left and right linear adjoint to black monoid $(\blacktriangleright^{\bullet}, ¡^{\bullet})$ and $(\blacktriangleright^{\circ}, ¡^{\circ})$ is left and right linear adjoint to $(\blacktriangleleft^{\bullet}, !^{\bullet})$ i.e., the 16 inequalities in the top of Figure 4 hold;*
2. *white and black (co)monoids satisfy the linear Frobenius laws, i.e. $(\mathrm{F}^{\bullet}_{\circ})$, $(\mathrm{F}^{\circ}_{\bullet})$, $(\mathrm{F}_{\bullet}^{\circ})$, $(\mathrm{F}_{\circ}^{\bullet})$ hold.*

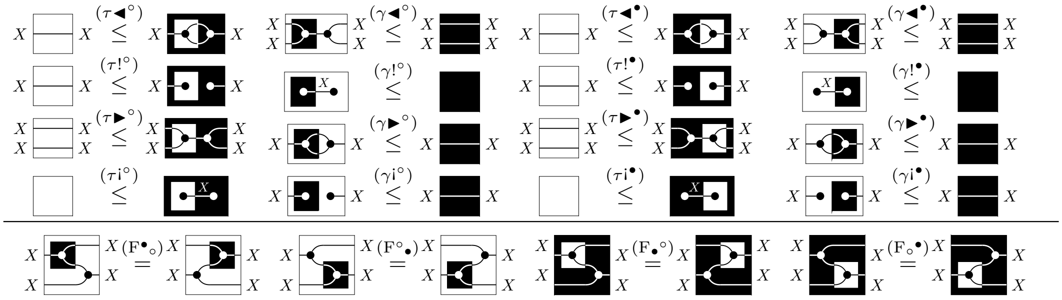


Figure 4 Additional axioms for fo-bicategories.

A morphism of fo-bicategories *is a morphism of linear bicategories* and *of (co)cartesian bicategories. We denote by* $\mathbb{FOB}$ *the category of fo-bicategories and their morphisms.*

We have seen that **Rel** is a closed linear bicategory, $\mathbf{Rel}^\circ$ a cartesian bicategory and $\mathbf{Rel}^\bullet$ a cocartesian bicategory. Given (3), it is easy to check the inequalities in Figure 4.

If $\mathbf{C}$ is a fo-bicategory, then $\mathbf{C}^{\text{co}}$ is a fo-bicategory when swapping white and black structures. Similarly, $\mathbf{C}^{\text{op}}$ is a fo-bicategory when swapping monoids and comonoids.

In a fo-bicategory $\mathbf{C}$, left and right linear adjoints of an arrow c coincide and are denoted by $c^\perp$. The assignment $c \mapsto c^\perp$ gives rise to an identity-on-objects isomorphism of fo-bicategories $(\cdot)^\perp \colon \mathbf{C} \to (\mathbf{C}^{\text{co}})^{\text{op}}$. Similarly, $(\cdot)^\dagger \colon \mathbf{C} \to \mathbf{C}^{\text{op}}$ in (6) is also an isomorphism of fo-bicategories.

Since the following diagram commutes, one can define the complement as the diagonal of the square, namely $\neg(\cdot) \stackrel{\text{def}}{=} ((\cdot)^\perp)^\dagger$.

$$\begin{tikzcd}
\mathbf{C} \arrow[r, "(\cdot)^\dagger"] \arrow[d, "(\cdot)^\perp"'] & \mathbf{C}^{\text{op}} \arrow[d, "(\cdot)^\perp"] \\
(\mathbf{C}^{\text{co}})^{\text{op}} \arrow[r, "(\cdot)^\dagger"] & \mathbf{C}^{\text{co}}
\end{tikzcd}$$

Clearly $\neg \colon \mathbf{C} \to \mathbf{C}^{\text{co}}$ is an isomorphism of fo-bicategories. Moreover, it induces a boolean algebra on each homset of $\mathbf{C}$.

▶ **Proposition 7.** *Let* $\mathbf{C}$ *be a fo-bicategory. Then, every homset of* $\mathbf{C}$ *is a boolean algebra.*

▶ **Proposition 8.** *Let* $F \colon \mathbf{C} \to \mathbf{D}$ *be a morphism of fo-bicategories. For all arrows* c*,* $\neg F(c) = F(\neg c)$*.*

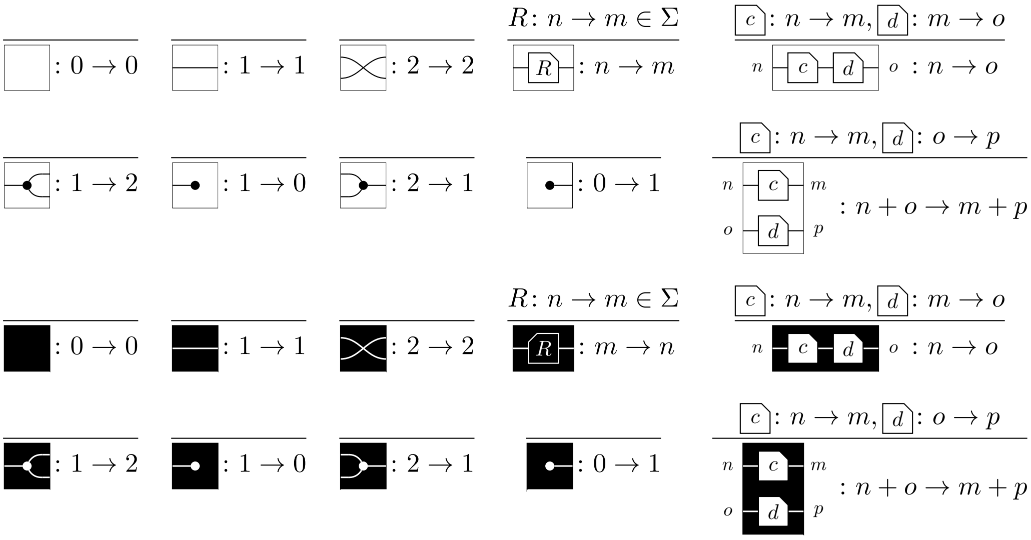
The next property of maps (Definition 2) plays a key role in our work.

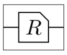
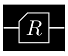
▶ **Proposition 9.** *For all maps* $f \colon X \to Y$ *and arrows* $c \colon Y \to Z$*, it holds that* $f \mathbin{;} \neg c = \neg(f \mathbin{;} c)$*.*

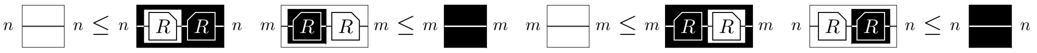
2.4 Freely Generated First-Order Bicategories

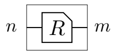
We conclude this section by giving to the reader a taste of how fo-bicategories relate to first-order theories. First, we recall from [3] the freely generated fo-bicategory $\mathbf{FOB}_\Sigma$.

Given a monoidal signature Σ, namely a set of symbols $R \colon n \to m$ with arity n and coarity m, $\mathbf{FOB}_\Sigma$ is the fo-bicategory whose objects are natural numbers and arrows $c \colon n \to m$ are string diagrams generated by the following rules:



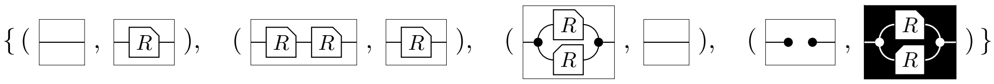
More precisely, arrows are equivalence classes of string diagrams w.r.t $\lesssim \cap \gtrsim$, where $\lesssim$ is the precongruence (w.r.t. $;$, $\otimes$, $\blacktriangledown$ and $\circledast$) generated by the axioms in Figures 1,2,3,4 (with X, Y, Z, W replaced by natural numbers, and a, b, c, d by diagrams of the appropriate type) and the axioms forcing  and  to be linear adjoints:



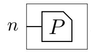
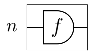
To give semantics to these diagrams we need *interpretations*, i.e. pairs $\mathcal{I} = (X, \rho)$, where X is a set and ρ is a function assigning to each $R\colon n \to m \in \Sigma$ a relation $\rho(R)\colon X^n \to X^m$. Since $\mathbf{FOB}_\Sigma$ is the free fo-bicategory, for any interpretation $\mathcal{I}$ there exists a unique morphism of fo-bicategories $\mathcal{I}^\sharp\colon \mathbf{FOB}_\Sigma \to \mathbf{Rel}$ such that $\mathcal{I}^\sharp(1) = X$ and $\mathcal{I}^\sharp($  $) = \rho(R) \subseteq X^n \times X^m$. Intuitively, $\mathcal{I}^\sharp$ is defined inductively by (1), (2) and (3) with the free cases provided by $\mathcal{I}$.

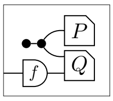
A *diagrammatic first-order theory* is a pair $\mathbb{T} = (\Sigma, \mathbb{I})$ where Σ is a monoidal signature and $\mathbb{I}$ is a set of *axioms*: pairs (c, d) for $c, d\colon n \to m$ in $\mathbf{FOB}_\Sigma$, standing for $c \leq d$. An interpretation $\mathcal{I}$ is a *model* of $\mathbb{T}$ if and only if, for all $(c, d) \in \mathbb{I}$, $\mathcal{I}^\sharp(c) \subseteq \mathcal{I}^\sharp(d)$. As illustrated in [3], one can generate the fo-bicategory $\mathbf{FOB}_\mathbb{T}$ and, in the spirit of Lawvere's functorial semantics [39], models of $\mathbb{T}$ are in one-to-one correspondence with morphisms $F\colon \mathbf{FOB}_\mathbb{T} \to \mathbf{Rel}$.

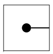
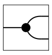
▶ **Example 10.** Consider the theory $\mathbb{T} = (\Sigma, \mathbb{I})$, where $\Sigma = \{R\colon 1 \to 1\}$ and $\mathbb{I}$ be as follows:



An interpretation is a set X and a relation $R \subseteq X \times X$. It is a model iff R is an order, i.e., reflexive ($id^\circ_X \subseteq R$), transitive ($R ; R \subseteq R$), antisymmetric ($R \cap R^\dagger \subseteq id^\circ$) and total ($\top \subseteq R \cup R^\dagger$).

▶ Remark 11. A direct encoding of traditional first-order theories into diagrammatic ones is illustrated in [3]. Shortly, a predicate symbol P of arity n becomes a symbol $P\colon n \to 0 \in \Sigma$, drawn as , and a n-ary function symbol f becomes $f\colon n \to 1 \in \Sigma$, drawn as . For instance, the formula $\exists x.P(x) \wedge Q(x, f(y))$ is rendered as follows



where  plays the role of $\exists$ and  that of $\wedge$. Note that both predicate and function symbols of traditional first-order theories are regarded as symbols of the monoidal signature Σ. Function symbols are constrained to represent functions by adding to $\mathbb{I}$ the axioms of maps, i.e., the inequalities in (4).

3 From Elementary-Existential Doctrines to Boolean Hyperdoctrines

The notion of hyperdoctrine was introduced by Lawvere in a series of seminal papers [40, 42]. Over the years, various generalizations and specializations of this concept have been formulated and applied across multiple domains in the fields of logic and computer science. In this work, we employ a generalization of the notion of hyperdoctrine introduced by Maietti and Rosolini in [46, 45, 47], namely that of an *elementary and existential doctrine.*

▶ **Definition 12.** *An* elementary and existential doctrine *is a functor* $P\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ *from the opposite of a category* $\mathbf{C}$ *with finite products to the category of inf-semilattices such that:*

- *for every* Y *in* $\mathbf{C}$ *there exists an element* δ_Y *in* $P(Y \times Y)$*, called* equality predicate*, such that for a morphism* $id^\circ_X \times \Delta_Y\colon X \times Y \to X \times Y \times Y$ *in* $\mathbf{C}$ *and every element* α *in* $P(X \times Y)$*, the assignment*

$$\exists_{id^\circ_X \times \Delta_Y}(\alpha) \overset{def}{=} P_{\langle \pi_1, \pi_2 \rangle}(\alpha) \wedge P_{\langle \pi_2, \pi_3 \rangle}(\delta_Y)$$

  determines a left adjoint to the functor $P_{id^\circ_X \times \Delta_Y}\colon P(X \times Y \times Y) \to P(X \times Y)$*;*
- *for any projection* $\pi_X\colon X \times Y \to X$*, the functor* $P_{\pi_X}\colon P(X) \to P(X \times Y)$ *has a left adjoint* $\exists_{\pi_X}$*, and these satisfy the* Beck-Chevalley condition *and* Frobenius reciprocity*, see [46, Sec. 2].*

▶ Remark 13. In an elementary and existential doctrine, for every $f\colon X \to Y$ of $\mathbf{C}$ the functor P_f has a left adjoint $\exists_f$ that can be computed as $\exists_{\pi_Y}(P_{f \times id^\circ_{X\,Y}}(\delta_Y) \wedge P_{\pi_X}(\alpha))$ for α in $P(X)$, where π_X and π_Y are the projections from $X \times Y$. These left ajoints satisfy the Frobenius reciprocity but not necessarily the Beck-Chevalley condition. See [48, Rem. 6.4].

▶ **Definition 14.** *Let* $P\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ *and* $R\colon \mathbf{D}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ *be two elementary and existential doctrines. A* morphism of elementary and existential doctrines *is given by a pair* $(F, \mathfrak{b})$ *where*

- $F\colon \mathbf{C} \to \mathbf{D}$ *is a finite product preserving functor;*
- $\mathfrak{b}\colon P \to F^{\mathrm{op}} \mathbin{;} R$ *is a natural transformation;*

satisfying the following conditions:

$$\begin{tikzcd}
\mathbf{C}^{\mathrm{op}} \arrow[d, "F^{\mathrm{op}}"'] \arrow[rd, "P"] \arrow[d, phantom, shift left=12, "\mathfrak{b}\downarrow"] & \\
\mathbf{D}^{\mathrm{op}} \arrow[r, "R"'] & \mathsf{InfSl}
\end{tikzcd}$$

1. *for every object* X *of* $\mathbf{C}$*,* $\mathfrak{b}_{X \times X}(\delta_X) = \delta_{FX \times FX}$*;*
2. *for every* $\pi_X\colon X \times Y \to X$ *of* $\mathbf{C}$ *and for every* α *in* $P(X \times Y)$*,* $\exists_{F(\pi_X)} \mathfrak{b}_{X \times Y}(\alpha) = \mathfrak{b}_X(\exists_{\pi_X}(\alpha))$*.*

We write $\mathbb{EED}$ for the category of elementary and existential doctrines and morphisms.

▶ **Example 15.** The powerset functor $\mathcal{P}\colon \mathsf{Set}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ is the archetypal example of an elementary and existential doctrine. More generally, for any regular category $\mathbf{C}$, the subobjects functor $\mathsf{Sub}_{\mathbf{C}}\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ is an elementary and existential doctrine, see [45, 46]. This assignment extends to an inclusion of the category $\mathbb{REG}$ of regular categories into $\mathbb{EED}$.

▶ **Example 16.** For a cartesian bicategory $\mathbf{C}$, the functor $\mathbf{C}[-, I]\colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ is an elementary and existential doctrine, where the actions of left adjoints is given $\exists_g(f) := f \mathbin{;} g^\dagger$ [7, Thm. 20]. As we will see in §4, this assignment extends to an inclusion of $\mathbb{CB}$ into $\mathbb{EED}$.

Similarly to cartesian bicategories, elementary and existential doctrines have enough structure to deal with the notion of *functional* (or single-valued) and *entire* (total) predicates.

▶ **Definition 17** (From [44]). *Let $P\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ be an elementary and existential doctrine. An element $\alpha \in P(X \times Y)$ is said to be* functional from X *to* Y *if* $P_{\langle\pi_1,\pi_2\rangle}(\alpha) \wedge P_{\langle\pi_1,\pi_3\rangle}(\alpha) \leq P_{\langle\pi_2,\pi_3\rangle}(\delta_Y)$ *in* $P(X \times Y \times Y)$. *Also, α is said to be* entire from X to Y *if* $\top_X \leq \exists_{\pi_X}(\alpha)$ *in* $P(X)$.

▶ Remark 18. By definition, a morphism of elementary and existential doctrines preserves both $\exists_{\pi_X}$ and δ_Y. Therefore it preserves functional and entire elements.

▶ **Example 19.** In $\mathcal{P}\colon \mathsf{Set}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ from Example 15, an $\alpha \in \mathcal{P}(X \times Y)$ is functional iff it defines a partial function from X to Y, while it is entire iff it is a total relation from X to Y.

▶ **Example 20.** In the doctrine $\mathbf{C}[-,I]\colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ from Example 16, functional and entire elements are precisely maps of $\mathbf{C}$. A detailed proof is in [5, Appendix E].

We can now recall the definition of *boolean hyperdoctrine.*

▶ **Definition 21** (boolean hyperdoctrine). *Let $\mathbf{C}$ be a category with finite products. A functor $P\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{Bool}$ is a* boolean hyperdoctrine *if it is an elementary and existential doctrine.*

A morphism $(F, \mathfrak{b}) : P \to R$ of boolean hyperdoctrines is a morphism of elementary and existential doctrines such that $\mathfrak{b}_X$ is a morphism of boolean algebras for all objects X of $\mathbf{C}$. We denote by $\mathbb{BHD}$ the category of boolean hyperdoctrines and their morphisms.

It is well-known that in first-order logic the universal quantifier can be derived by the existential quantifier and the negation. The same happens in boolean hyperdoctrines: for all arrows $f\colon X \to Y$, the functor $\forall_f(-) \overset{\mathrm{def}}{=} \neg\exists_f\neg(-)$ is a right adjoint to P_f (see [5, Appendix B.1]).

▶ **Example 22.** The powerset functor $\mathcal{P}\colon \mathsf{Set}^{\mathrm{op}} \longrightarrow \mathsf{Bool}$ provides an example of a boolean hyperdoctrine. This can be generalized to an arbitrary *boolean category* $\mathbf{B}$, namely a coherent category such that every subobject has a complement, see [33, Sec. A1.4, p. 38]. The subobjects functor on $\mathbf{B}$ is a boolean hyperdoctrine $\mathsf{Sub}_{\mathbf{B}}\colon \mathbf{B}^{\mathrm{op}} \longrightarrow \mathsf{Bool}$.

▶ **Example 23.** Given a standard first-order theory Th in a first-order language $\mathcal{L}$ (for simplicity single sorted), one can consider the functor $\mathcal{L}^{\mathrm{Th}}\colon \mathcal{V}^{\mathrm{op}} \longrightarrow \mathsf{Bool}$. The base category $\mathcal{V}$ is the *syntactic* category of $\mathcal{L}$, i.e. the category where objects are natural numbers and morphisms are lists of terms, while the predicates of $\mathcal{L}^{\mathrm{Th}}(n)$ are given by equivalence classes (with respect to provable reciprocal consequence $\dashv\vdash$) of well-formed formulae with free variables in $\{x_1, \dots, x_n\}$, and the partial order is given by the provable consequences, according to the fixed theory Th. In this case, the left adjoint to the weakening functor $\mathcal{L}^{\mathrm{Th}}_{\pi}$ is computed by existentially quantifying the variables that are not involved in the substitution induced by the projection π. Dually, the right adjoint is computed by quantifying universally. The equality predicate is give by the formula $x_1 = x_2$.

▶ **Example 24.** Let A be a boolean algebra. The representable functor $\mathsf{A}^{(-)}\colon \mathsf{Set}^{\mathrm{op}} \longrightarrow \mathsf{Bool}$ assigning to a set X the poset A^X of functions from X to A with the point-wise order is a boolean hyperdoctrine.

We conclude this section with a result that, intuitively, is the analogous of Proposition 9.

▶ **Lemma 25.** *Let $P\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{Bool}$ be a boolean hyperdoctrine and $\phi \in P(X \times Y)$ a functional and entire element from X toY. For all $\psi \in P(Y \times Z)$, it holds that*

$$\exists_{\pi_{X\times Z}}(P_{\pi_{X\times Y}}(\phi) \wedge P_{\pi_{Y\times Z}}(\neg\psi)) = \neg(\,\exists_{\pi_{X\times Z}}(P_{\pi_{X\times Y}}(\phi) \wedge P_{\pi_{Y\times Z}}(\psi))\,).$$

4 An Adjunction and an Equivalence

In [7], cartesian bicategories and elementary existential doctrines are compared. The main results of [7, Thm. 28] states that there exists the following adjunction.

$$\begin{tikzcd}
\mathbb{CB} \arrow[r, hook, bend right, "\mathsf{Hml}"'] \arrow[r, phantom, "\perp"] & \mathbb{EED} \arrow[l, bend right, "\mathsf{Rel}"']
\end{tikzcd} \tag{7}$$

The embedding $\mathsf{Hml}\colon \mathbb{CB} \to \mathbb{EED}$ maps a cartesian bicategory $\mathbf{C}$ into the hom-functor $\mathbf{C}[-,I]\colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ that, as explained in Example 16, is an elementary existential doctrine. The functor $\mathsf{Rel}\colon \mathbb{EED} \to \mathbb{CB}$ is a generalisation to elementary and existential doctrines of the construction of bicategory relations associated with a regular category (see [11, Ex. 1.4]). For $P\colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$, the cartesian bicategory $\mathsf{Rel}(P)$ is defined as follows:

- objects are those of $\mathbf{C}$; for objects X, Y, the homsets $\mathsf{Rel}(P)[X,Y]$ are the posets $P(X \times Y)$;
- the identity for an object X is the equality predicate δ_X in $P(X \times X)$;
- composition of $\phi\colon X \to Y$ and $\psi\colon Y \to Z$ is given by $\exists_{\pi_{X \times Z}}(P_{\pi_{X\times Y}}(\phi) \wedge P_{\pi_{Y \times Z}}(\psi))$.

The reader is referred to [7] or to [5, Appendix C] for further details on the adjunction in (7).

Another result in [7, Thm. 35] shows that the adjunction in (7) restricts to an equivalence

$$\mathbb{CB} \equiv \overline{\mathbb{EED}} \tag{8}$$

where $\overline{\mathbb{EED}}$ is a full subcategory of $\mathbb{EED}$ whose objects are particularly well-behaved doctrines. For the sake of readability, we will make clear in §7 what these doctrines are.

5 Peircean Bicategories

We now introduce *peircean bicategories*, an alternative presentation of fo-bicategories. The name peircean is due to the fact that, like in Peirce's algebra of relations [55], and differently from fo-bicategories, the structure of boolean algebra is taken as a primitive.

▶ **Definition 26.** *A* peircean bicategory *consists of a cartesian bicategory* $(\mathbf{C}, \blacktriangleleft^{\circ}, \blacktriangleright^{\circ})$ *such that*

1. *every homset* $\mathbf{C}[X,Y]$ *carries a Boolean algebra* $(\mathbf{C}[X,Y], \vee, \bot, \wedge, \top, \neg)$*;*
2. *for all maps* $f\colon X \to Y$ *and arrows* $c\colon Y \to Z$*,*

$$f \fatsemi \neg c = \neg(f \fatsemi c). \tag{$\neg\mathcal{M}$}$$

A morphism of peircean bicategories *is a morphism of cartesian bicategories* $F\colon \mathbf{C} \to \mathbf{D}$ *such that* $F(\neg c) = \neg F(c)$*. We write* $\mathbb{PB}$ *for the category of peircean bicategories and their morphisms.*

By Propositions 7 and 9 every fo-bicategory is a peircean bicategory. By Proposition 8 every morphism of fo-bicategories is a morphism of peircean bicategories.

Vice versa, every peircean bicategory $(\mathbf{C}, \blacktriangleleft^{\circ}, \blacktriangleright^{\circ})$ gives rise to a fo-bicategory. The black structure $(\mathbf{C}, \blacktriangleleft^{\bullet}, \blacktriangleright^{\bullet})$ is defined as expected from the white one and $\neg$. Namely:

$$\begin{array}{cccc}
c \mathbin{\overset{\bullet}{;}} d \overset{\text{def}}{=} \neg(\neg c \fatsemi \neg d) & id^{\bullet}_X \overset{\text{def}}{=} \neg id^{\circ}_X & c \mathbin{\boldsymbol{\otimes}} d \overset{\text{def}}{=} \neg(\neg c \otimes \neg d) & \sigma^{\bullet}_{X,Y} \overset{\text{def}}{=} \neg\sigma^{\circ}_{X,Y} \\
\blacktriangleleft^{\bullet}_X \overset{\text{def}}{=} \neg\blacktriangleleft^{\circ}_X & !^{\bullet}_X \overset{\text{def}}{=} \neg !^{\circ}_X & \blacktriangleright^{\bullet}_X \overset{\text{def}}{=} \neg\blacktriangleright^{\circ}_X & ¡^{\bullet}_X \overset{\text{def}}{=} \neg ¡^{\circ}_X
\end{array} \tag{9}$$

With this definition, it is immediate to see that $\neg\colon (\mathbf{C}^{\mathrm{co}}, \blacktriangleleft^{\circ}, \blacktriangleright^{\circ}) \to (\mathbf{C}, \blacktriangleleft^{\bullet}, \blacktriangleright^{\bullet})$ is an isomorphism and thus to conclude that $(\mathbf{C}, \blacktriangleleft^{\bullet}, \blacktriangleright^{\bullet})$ is a cocartesian bicategory. Proving that $(\mathbf{C}, \blacktriangleleft^{\circ}, \blacktriangleright^{\circ})$ and $(\mathbf{C}, \blacktriangleleft^{\bullet}, \blacktriangleright^{\bullet})$ give rise to a fo-bicategory is the main technical effort of this paper.

▶ **Theorem 27.** *There is an isomorphism of categories* $\mathbb{FOB} \cong \mathbb{PB}$.

Proof (sketch). As mentioned above, most of the proof is devoted to showing that every peircean bicategory is a fo-bicategory.

This is achieved by proving first that the relational operations on the homsets, like $(\cdot)^\dagger$ defined in (6), are preserved by negation, e.g. $\neg(c^\dagger) = (\neg c)^\dagger$. This is also where the property on maps ($\neg\mathcal{M}$) mostly comes into play.

Then, to prove that the axioms of fo-bicategories are satisfied, one crucially exploits the laws of boolean algebras.

Finally, to show that every morphism F of peircean bicategories is a morphism of fo-bicategories, it suffices to observe that F preserves the strucure in (9), as it preserves negation by definition.

The full diagrammatic proof can be found in [5, Appendix D]. ◀

Note that, differently from Definition 6, Definition 26 is not purely axiomatic, since ($\neg\mathcal{M}$) requires f to be a map. However, the notion of a peircean bicategory is notably more succinct than that of a fo-bicategory, making it more convenient for our purposes.

6 An Equational Presentation of Boolean Hyperdoctrines

The main purpose of this section is to establish a formal link between fo-bicategories and boolean hyperdoctrines. In particular, we are going to show that the adjunction presented in (7) restricts to an adjunction between $\mathbb{FOB}$ and $\mathbb{BHD}$. Theorem 27 allows us to conveniently work with peircean bicategories. We commence with the following result.

▶ **Proposition 28.** *Let* $\mathbf{C}$ *be a peircean bicategory. Then* $\mathsf{HmI}(\mathbf{C})$ *is a boolean hyperdoctrine.*

Proof. By (7), $\mathbf{C}[-, I] \colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ is an elementary and existential doctrine and, by definition of peircean bicategories, $\mathbf{C}[X, I]$ is a boolean algebra for all objects X. To conclude that $\mathbf{C}[-, I] \colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{Bool}$, one has only to show that, for all maps $f \colon X \to Y$, $\mathbf{C}[f, I] \colon \mathbf{C}[Y, I] \to \mathbf{C}[X, I]$ is a morphism of boolean algebras. Since, by (7), $\mathbf{C}[f, I]$ is a morphism of inf-semilattices, it is enough to show that it preserves negation: for all $c \in \mathbf{C}[Y, I]$

$$\begin{aligned}
\mathbf{C}[f, I](\neg c) &= f \mathbin{;} \neg c && \text{(Definition of } \mathbf{C}[-, I]) \\
&= \neg(f \mathbin{;} c) && (\neg\mathcal{M}) \\
&= \neg\mathbf{C}[f, I](c) && \text{(Definition of } \mathbf{C}[-, I])
\end{aligned}$$

◀

The above proposition allows us to characterize peircean bicategories as follows:

▶ **Corollary 29.** *Let* $\mathbf{C}$ *be a cartesian bicategory. Then it is a peircean bicategory if and only if* $\mathsf{HmI}(\mathbf{C})$ *is a boolean hyperdoctrine.*

To prove that, for any boolean hyperdoctrine P, $\mathsf{Rel}(P)$ is a peircean bicategory, we need to establish a formal correspondence between Definition 2 and Definition 17.

▶ **Proposition 30.** *Let* $P \colon \mathbf{C}^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ *be an elementary and existential doctrine. Then the maps of* $\mathsf{Rel}(P)$ *are precisely the functional and entire elements of* P.

▶ **Proposition 31.** *Let* P *be a boolean hyperdoctrine. Then* $\mathsf{Rel}(P)$ *is a peircean bicategory.*

Proof. By (7), $\mathsf{Rel}(P)$ is a cartesian bicategory. Since $P(X)$ is a boolean algebra for all objects X, then each hom-set $\mathsf{Rel}(P)[X,Y]$ – by definition $P(X \times Y)$ – is a boolean algebra. To conclude that $\mathsf{Rel}(P)$ is a peircean bicategory, it is enough to show that $(\neg\mathcal{M})$ holds, that is

$$\phi \fatsemi \neg\psi = \neg(\phi \fatsemi \psi)$$

for all maps $\phi \in \mathsf{Rel}(P)[X,Y]$ and arrows $\psi \in \mathsf{Rel}(P)[Y,Z]$. By Proposition 30, ϕ is a functional and entire element of P. Thus, one can rely on Lemma 25 to conclude that

$$\begin{aligned} \phi \fatsemi \neg\psi &= \exists_{\pi_{X\times Z}}(P_{\pi_{X\times Y}}(\phi) \wedge P_{\pi_{Y\times Z}}(\neg\psi)) && \text{(Defintion of } \mathsf{Rel}(P)) \\ &= \neg(\, \exists_{\pi_{X\times Z}}(P_{\pi_{X\times Y}}(\phi) \wedge P_{\pi_{Y\times Z}}(\psi))\,) && \text{(Lemma 25)} \\ &= \neg(\phi \fatsemi \psi) && \text{(Defintion of } \mathsf{Rel}(P)) \end{aligned}$$

◀

By Propositions 28 and 31 proving the following result amounts to a few routine checks.

▶ **Theorem 32.** *The adjunction in* (7)*, restricts to the adjunction below on the left.*

$$\begin{tikzcd} \mathbb{PB} \arrow[r, bend right, "\mathsf{Hml}"'] \arrow[r, phantom, "\perp"] & \mathbb{BHD} \arrow[l, bend right, "\mathsf{Rel}"'] \end{tikzcd}$$

Thus, by Theorem 27, there is an adjunction $\begin{tikzcd} \mathbb{FOB} \arrow[r, bend right] \arrow[r, phantom, "\perp"] & \mathbb{BHD} \arrow[l, bend right] \end{tikzcd}$ *.*

Proof. First, we want to prove that the inclusion $\mathsf{Hml}\colon \mathbb{CB} \hookrightarrow \mathbb{EED}$ in (7) restricts to an inclusion of categories $\mathbb{PB} \hookrightarrow \mathbb{BHD}$. By Proposition 28, one only needs to check for morphisms in $\mathbb{PB}$. Given a morphism of peircean bicategories $F\colon \mathbf{C} \to \mathbf{D}$, $\mathsf{Hml}(F)$ is the morphism of elementary and existential doctrines $(\tilde{F}, \mathfrak{b}^F)$ defined in Section 4. In order to conclude that it is a morphism of boolean doctrines, it is enough to show that $\mathfrak{b}^F_X$ is a morphism of boolean algebras for all objects X. Since $(\tilde{F}, \mathfrak{b}^F)$ is a morphism of doctrines, $\mathfrak{b}^F_X$ is a morphism of inf-semilattices. Thus it is enough to show that $\mathfrak{b}^F_X$ preserve negation. But this is trivial since, for all $c \in \mathbf{C}[X, I]$,

$$\begin{aligned} \mathfrak{b}^F_X(\neg c) &= F(\neg c) && \text{(Def. } \mathfrak{b}^F) \\ &= \neg F(c) && \text{(morphism of Peircean, Definition 26)} \\ &= \neg \mathfrak{b}^F_X(c) && \text{(Def. } \mathfrak{b}^F) \end{aligned}$$

Now, to prove that Rel restrict to a functor $\mathsf{Rel}\colon \mathbb{BHD} \to \mathbb{PB}$, by Proposition 31, one only needs to check that for all morphisms of boolean hyperdoctrines $(F, \mathfrak{b})\colon P \to Q$, $\mathsf{Rel}(F, \mathfrak{b})\colon \mathsf{Rel}(P) \to \mathsf{Rel}(Q)$ is a morphism of peicean bicategories. Since by (7), $\mathsf{Rel}(F, \mathfrak{b})$ is a morphism of cartesian bicategories, one only needs to check that it preserves the negation. But this is obvious since for all arrows $\phi \in \mathsf{Rel}(P)[X,Y]$, $\mathsf{Rel}(F, \mathfrak{b})(\phi)$ is – by definition – $\mathfrak{b}_{X\times Y}(\phi)$ and $\mathfrak{b}_{X\times Y}$ is a morphism of boolean algebras.

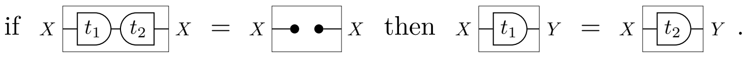
To conclude, one only needs to check the unit and the counit of the adjunction in (7). The counit is an isomorphism of cartesian bicategories (see Equation (9) in [7]), and then it provides an isomorphism of peircean bicategories $\mathbf{C} \cong \mathsf{Rel}(\mathbf{C}[-, I])$ whenever $\mathbf{C}$ is a peircean bicategory. The unit of the adjunction $\eta_P\colon P \to \mathsf{Rel}(P)[-, I]$ is the morphism of elementary and existential doctrines (Γ_P, ρ) illustrated in [7, Section 7] or [5, Appendix C]. To conclude that η_P is a morphism of boolean hyperdoctrine whenever P is a boolean hyperdoctrine, one has only to prove that ρ is a morphism of boolean algebras, but this is trivial since ρ is always an isomorphism of inf-semilattices. ◀

7 Boolean Hyperdoctrines Representing First-Order Bicategories

As anticipated in §4, the adjunction in (7) becomes an equivalence for certain well-behaved doctrines. Definitions 33 and 34 state the conditions that such doctrines must satisfy.

▶ **Definition 33.** *An elementary and existential doctrine $P\colon \mathbf{C}^{\text{op}} \longrightarrow \mathsf{InfSl}$ has* comprehensive diagonals *if for the equality predicate $\delta_X \in P(X)$ it holds that $P_{\Delta_X}(\delta_X) = \top_X$ and every arrow $f\colon Y \to X \times X$ such that $P_f(\delta_X) = \top_Y$ factors (uniquely) through Δ_X .*

Intuitively, a doctrine has comprehensive diagonals if its equality is *extensional*, namely if a formula $t_1 = t_2$ is true, then the terms t_1 and t_2 are syntactically equal. In the language of cartesian bicategories, for two maps t_1, t_2, this can be stated by means of diagrams as



(10)

While it is sometimes meaningful to consider syntactic doctrines (e.g. Example 23) in which the equality is not extensional, in several semantical doctrines this condition is satisfied.

▶ **Definition 34.** *Let $P\colon \mathbf{C}^{\text{op}} \longrightarrow \mathsf{InfSl}$ be an elementary existential doctrine. We say that P satisfies the* Rule of Unique Choice *(RUC) if for every entire functional element ϕ in $P(X \times Y)$ there exists an arrow $f\colon X \to Y$ such that $\top_X \leq P_{\langle id_X^\circ, f\rangle}(\phi)$.*

The reader can think that a doctrine has (RUC) if for every element (intuitively formula) that is entire and functional, there exists an arrow in $\mathbf{C}$ (intuitively a term) that represents it.

▶ **Example 35.** The doctrine $\mathcal{P}\colon \mathsf{Set}^{\text{op}} \longrightarrow \mathsf{InfSl}$ has comprehensive diagonals, and it satisfies the (RUC) (since every functional and total relation can be represented by a function). More generally, every subobject doctrine $\mathsf{Sub}_{\mathbf{C}}\colon \mathbf{C}^{\text{op}} \longrightarrow \mathsf{InfSl}$ on a regular category, as presented in Example 15 satisfies the (RUC) and it has comprehensive diagonals, as observed in [44].

▶ **Example 36.** The doctrine $\mathbf{C}[-, I]\colon \mathsf{Map}(\mathbf{C})^{\text{op}} \longrightarrow \mathsf{InfSl}$ presented in Example 16 satisfies the (RUC) and it has comprehensive diagonals, as proved in [7]. The reader can find a diagrammatic proof of (10) in [5, Appendix A].

Hereafter – and in the equivalence in (8) – $\overline{\mathbb{EED}}$ is the full subcategory of $\mathbb{EED}$ whose objects are doctrines satisfying (RUC) and with comprehensive diagonals. Similarly $\overline{\mathbb{BHD}}$ is the full subcategory of $\mathbb{BHD}$ whose objects are boolean hyperdoctrines satisfying (RUC) and with comprehensive diagonals.

By means of Theorem 32, it is easy to prove that the equivalence in (8) restricts as follows.

▶ **Theorem 37.** $\mathbb{PB} \equiv \overline{\mathbb{BHD}}$ *and thus, by Theorem 27,* $\mathbb{FOB} \equiv \overline{\mathbb{BHD}}$*.*

Proof. By Equation (8) we have that the Hml and Rel functors provide an equivalence between the categories $\mathbb{CB}$ and $\overline{\mathbb{EED}}$. Now, since every peircean category is in particular a cartesian bicategory, we have that every boolean hyperdoctrine arising from a peircean bicategory satisfies (RUC) and it has comprehensive diagonals. Then, we have that the functor $\mathsf{Hml}\colon \mathbb{PB} \hookrightarrow \mathbb{BHD}$ factors through the canonical inclusion $\overline{\mathbb{BHD}} \hookrightarrow \mathbb{BHD}$:

$$\begin{tikzcd}
\mathbb{PB} \arrow[rr, hook, "\mathsf{Hml}"] \arrow[dr, hook, "\mathsf{Hml}"'] & & \mathbb{BHD} \\
& \overline{\mathbb{BHD}} \arrow[ur, hook] &
\end{tikzcd}$$

By Theorem 32, we have that $\mathsf{Hml}\colon \mathbb{PB} \hookrightarrow \mathbb{BHD}$ is fully and faithful (since the counit of the adjunction is an iso), so it remains to prove that it is essentially surjective (with respect to the objects of $\overline{\mathbb{BHD}}$). By the equivalence presented in Equation (8), we know that every boolean hyperdoctrine (that is in particular an elementary and existential doctrine) satisfying (RUC) and having comprehensive diagonals, is isomorphic to an elementary and existential doctrine $\mathbf{C}[-, I]\colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ for some cartesian bicategory $\mathbf{C}$. Thus, we can conclude that $\mathbf{C}[-, I]\colon \mathsf{Map}(\mathbf{C})^{\mathrm{op}} \longrightarrow \mathsf{InfSl}$ is a boolean hyperdoctrine and, by Corollary 29, that $\mathbf{C}$ is a peircean bicategory. This concludes the proof that $\mathbb{PB} \equiv \overline{\mathbb{BHD}}$. ◀

8 Related Work

There exists many structures that are closely related to fo-bicategories, as discussed in [3]. The introduction of peircean bicategory in § 5 provide clearer correspondences with such structures. Here we discuss some of them.

Boolean hyperdoctrines are used in [10] as a categorical treatment of another work of Peirce: *existential graphs* [64]. While the latter share some similarities with the graphical language of fo-bicategories there is one notable difference: negation is a primitive operator rather than a derived one, as it happens for instance also in [25] and Definition 26. In [3] and in §5, it is emphasised how this choice makes the resulting calculus less algebraic in flavour, having to deal with convoluted rules such as the one for (de)iteration or properties which are not purely equational, such as $(\neg\mathcal{M})$.

Inspired by [10], another graphical language [50] akin to Peirce's graphs is based on a decomposition of a hyperdoctrine into a bifibration. In this work, the categorical treatment revolves around the notion of monoidal *chiralities* [49], which are much more closer in spirit to fo-bicategories. We believe that our results might set an initial step towards a connection between fo-bicategories and chiralities.

A recent work [15] proposes a relational understanding of doctrines. However, these corresponds to the regular fragment of first-order logic, and thus it might by intriguing to understand the role of the additional black structure of first-order bicategories in this setting. Another route, suggested by the equivalence in Theorem 37, might be to understand the role of $(\neg\mathcal{M})$ in relational doctrines with boolean fibres.

Finally, it is also worth remarking that peircean bicategories, as well as fo-bicategories, are poset-enriched categories. Such categorical treatements of first-order logic are also found in works such as [30, 23], along with the references therein. Their primary focus, though, is on the categorical approach to classical proof theory instead of semantics.

9 Conclusions and Future Work

Theorems 32 and 37 provide a solid bridge between functional and relational approaches to classical logic. The former rely on categorical structures that are usually defined by means of exactness properties; the latter on fo-bicategories which enjoy a purely equational presentation, much in the spirit of Boole's algebra and Peirce's calulus.

To achieve our result, we found it extremely convenient to introduce the notion of peircean bicategories that, by Theorem 27, provide a far handier characterisation of fo-bicategories.

Theorem 27 might also be useful to establish a correspondence with *allegories* [22]: since cartesian bicategories are equivalent to unitary pretabular allegories [35], we expect that such allegories where, additionally, homsets carry boolean algebras satisfying $(\neg\mathcal{M})$ are equivalent to fo-bicategories. Despite searching the literature, we did not find analogous structures. Interestingly, the property $(\neg\mathcal{M})$ can be proven in any Peirce allegories, as shown in Proposition 4.6.1 in [53].

Finally, as future work we aim to investigate how our characterizations can be extended to higher-order classical logic, which is categorically represented through the notion of *tripos* [29, 57]. Indeed, we believe that the constructions and results presented in this work, together with the notion of tripos, can serve as a guide for defining a variant of fo-bicategories – hopefully, purely equational – capable of representing higher-order classical logic.

References

1 Richard Bird and Oege De Moor. The algebra of programming. *NATO ASI DPD*, 152:167–203, 1996.

2 Benedikt Bollig, Alain Finkel, and Amrita Suresh. Bounded Reachability Problems Are Decidable in FIFO Machines. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2020.49.

3 Filippo Bonchi, Alessandro Di Giorgio, Nathan Haydon, and Pawel Sobocinski. Diagrammatic algebra of first order logic. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '24, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3661814.3662078.

4 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory I: rewriting with frobenius structure. *J. ACM*, 69(2):14:1–14:58, 2022. doi:10.1145/3502719.

5 Filippo Bonchi, Alessandro Di Giorgio, and Davide Trotta. When lawvere meets peirce: an equational presentation of boolean hyperdoctrines, 2024. arXiv:2404.18795.

6 Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: From linear to concurrent systems. *Proceedings of the ACM on Programming Languages*, 3(POPL):25:1–25:28, January 2019. doi:10.1145/3290338.

7 Filippo Bonchi, Alessio Santamaria, Jens Seeber, and Paweł Sobociński. On doctrines and cartesian bicategories. In *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

8 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full Abstraction for Signal Flow Graphs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 515–526, New York, NY, USA, January 2015. Association for Computing Machinery. doi:10.1145/2676726.2676993.

9 George Boole. *The mathematical analysis of logic*. Philosophical Library, 1847.

10 Geraldine Brady and Todd Trimble. A categorical interpretation of c.s. peirce's propositional logic alpha. *Journal of Pure and Applied Algebra - J PURE APPL ALG*, 149:213–239, June 2000. doi:10.1016/S0022-4049(98)00179-0.

11 A. Carboni and R. F. C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49(1):11–32, November 1987. doi:10.1016/0022-4049(87)90121-6.

12 J. Robin B. Cockett, Jürgen Koslowski, and Robert AG Seely. Introduction to linear bicategories. *Mathematical Structures in Computer Science*, 10(2):165–203, 2000.

13 Edgar Frank Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 26(1):64–69, 1983.

14 Bob Coecke and Ross Duncan. Interacting quantum observables: Categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, April 2011. doi:10.1088/1367-2630/13/4/043016.

15 Francesco Dagnino and Fabio Pasquali. Quotients and Extensionality in Relational Doctrines. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023)*, volume 260 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:23, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2023.25.

16 Augustus De Morgan. On the syllogism, no. iv. and on the logic of relations. *Printed by C.J. Clay at the University Press*, 1860.

17 Charles J Everett and Stanislaw Ulam. Projective algebra i. *American Journal of Mathematics*, 68(1):77–88, 1946.

18 William Ewald. The emergence of first-order logic. *Stanford Encyclopedia of Philosophy*, 2018.

19 Brendan Fong, Paweł Sobociński, and Paolo Rapisarda. A categorical approach to open and interconnected dynamical systems. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 495–504, New York, NY, USA, July 2016. Association for Computing Machinery. doi:10.1145/2933575.2934556.

20 Brendan Fong and David Spivak. String diagrams for regular logic (extended abstract). In John Baez and Bob Coecke, editors, *Applied Category Theory 2019*, volume 323 of *Electronic Proceedings in Theoretical Computer Science*, pages 196–229. Open Publishing Association, September 2020. doi:10.4204/eptcs.323.14.

21 GOTTLOB FREGE. *Begriffsschrift und andere Aufsätze*. Georg Olms Verlag, 1977.

22 Peter Freyd and Andre Scedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. Elsevier B.V, 1990.

23 CARSTEN FÜHRMANN and DAVID PYM. On categorical models of classical logic and the geometry of interaction. *Mathematical Structures in Computer Science*, 17(5):957–1027, 2007. doi:10.1017/S0960129507006287.

24 Dan R. Ghica and Achim Jung. Categorical semantics of digital circuits. In *2016 Formal Methods in Computer-Aided Design (FMCAD)*, pages 41–48, 2016. doi:10.1109/FMCAD.2016.7886659.

25 Nathan Haydon and Paweł Sobociński. Compositional diagrammatic first-order logic. In *11th International Conference on the Theory and Application of Diagrams (DIAGRAMS 2020)*, 2020.

26 Leon Henkin. Cylindric algebras, 1971.

27 CAR Hoare and He Jifeng. The weakest prespecification, part i. *Fundamenta Informaticae*, 9(1):51–84, 1986.

28 Ian Hodkinson and Szabolcs Mikulás. Axiomatizability of reducts of algebras of relations. *Algebra Universalis*, 43(2):127–156, August 2000. doi:10.1007/s000120050150.

29 J.M.E. Hyland, P.T. Johnstone, and A.M. Pitts. Tripos theory. *Math. Proc. Camb. Phil. Soc.*, 88:205–232, 1980.

30 Martin Hyland. Abstract interpretation of proofs: Classical propositional calculus. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic*, pages 6–21, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

31 B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and the foundations of mathematics*. North Holland Publishing Company, 1999.

32 P.T. Johnstone. *Topos Theory*. cademic Press, 1977.

33 P.T. Johnstone. *Sketches of an elephant: a topos theory compendium*, volume 2 of *Studies in Logic and the foundations of mathematics*. Oxford Univ. Press, 2002.

34 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, July 1991. doi:10.1016/0001-8708(91)90003-P.

35 Petrus Marinus Waltherus Knijnenburg and Frank Nordemann. *Two Categories of Relations*. Citeseer, 1994.

36 Alexander Krauss and Tobias Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Automated Reasoning*, 49(1):95–106, 2012. doi:10.1007/s10817-011-9223-4.

37 Ugo Dal Lago and Francesco Gavazzo. A relational theory of effects and coeffects. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022. doi:10.1145/3498692.

38 Søren B Lassen. Relational reasoning about contexts. *Higher order operational techniques in semantics*, 91, 1998.

39 F. W. Lawvere. *Functorial Semantics of Algebraic Theories.* PhD thesis, Columbia University, New York, NY, USA, 1963.

40 F.W. Lawvere. Adjointness in foundations. *Dialectica*, 23:281–296, 1969.

41 F.W. Lawvere. Diagonal arguments and cartesian closed categories. In *Category Theory, Homology Theory and their Applications*, volume 2, pages 134–145. Springer, 1969.

42 F.W. Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In A. Heller, editor, *New York Symposium on Application of Categorical Algebra*, volume 2, pages 1–14. American Mathematical Society, 1970.

43 Leopold Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76(4):447–470, 1915.

44 M.E. Maietti, F. Pasquali, and G. Rosolini. Triposes, exact completions, and hilbert's ε-operator. *Tbilisi Mathematica journal*, 10(3):141–166, 2017.

45 M.E. Maietti and G. Rosolini. Elementary quotient completion. *Theory App. Categ.*, 27(17):445–463, 2013.

46 M.E. Maietti and G. Rosolini. Quotient completion for the foundation of constructive mathematics. *Log. Univers.*, 7(3):371–402, 2013.

47 M.E. Maietti and G. Rosolini. Unifying exact completions. *Appl. Categ. Structures*, 23:43–52, 2013.

48 M.E. Maietti and D. Trotta. A characterization of generalized existential completions. *Annals of Pure and Applied Logic*, 174(4):103234, 2023.

49 Paul-André Melliès. Dialogue categories and chiralities. *Publications of the Research Institute for Mathematical Sciences*, 52(4):359–412, 2016.

50 Paul-André Melliès and Noam Zeilberger. A bifibrational reconstruction of lawvere's presheaf hyperdoctrine. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 555–564, 2016.

51 Donald Monk. On representable relation algebras. *Michigan Mathematical Journal*, 11(3):207–210, 1964. doi:10.1307/mmj/1028999131.

52 Koko Muroya, Steven W. T. Cheung, and Dan R. Ghica. The geometry of computation-graph abstraction. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 749–758. ACM, 2018. doi:10.1145/3209108.3209127.

53 Jean-Pierre Olivier and Dany Serrato. Peirce allegories. identities involving transitive elements and symmetrical ones. *Journal of Pure and Applied Algebra*, 116(1-3):249–271, 1997.

54 Charles S. Peirce. The logic of relatives. *The Monist*, 7(2):161–217, 1897. URL: http://www.jstor.org/stable/27897407.

55 Charles Sanders Peirce. *Studies in logic. By members of the Johns Hopkins university.* Little, Brown, and Company, 1883.

56 Robin Piedeleu and Fabio Zanasi. A String Diagrammatic Axiomatisation of Finite-State Automata. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 469–489, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-71995-1_24.

57 A.M. Pitts. Tripos theory in retrospect. *Math. Struct. in Comp. Science*, 12:265–279, 2002.

58 Damien Pous. Kleene algebra with tests and coq tools for while programs. In *Interactive Theorem Proving: 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings 4*, pages 180–196. Springer, 2013.

59 Damien Pous. *Automata for relation algebra and formal proofs.* PhD thesis, ENS Lyon, 2016.

60 Damien Pous. On the positive calculus of relations with transitive closure. 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 3:1–3:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.3.

61 Vaughan R Pratt. Semantical considerations on floyd-hoare logic. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 109–121. IEEE, 1976.

62 W.V. Quine. Predicate-functor logics. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 309–315. Elsevier, 1971. doi:10.1016/S0049-237X(08)70850-4.

63 Valentin N Redko. On defining relations for the algebra of regular events. *Ukrainskii Matematicheskii Zhurnal*, 16:120–126, 1964.

64 Don D. Roberts. *The Existential Graphs of Charles S. Peirce.* De Gruyter Mouton, 1973.

65 P. Selinger. A Survey of Graphical Languages for Monoidal Categories. In B. Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, pages 289–355. Springer, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-12821-9_4.

66 Dario Stein and Sam Staton. Probabilistic programming with exact conditions. *Journal of the ACM*, 2023.

67 Alfred Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, September 1941. doi:10.2307/2268577.

Unveiling the Connection Between the Lyndon Factorization and the Canonical Inverse Lyndon Factorization via a Border Property

Paola Bonizzoni[1]

Dip. di Informatica, Sistemistica e Comunicazione, University of Milano-Bicocca, Milan, Italy

Clelia De Felice[2]

Dip. di Informatica, University of Salerno, Fisciano, Italy

Brian Riccardi

Dip. di Informatica, Sistemistica e Comunicazione, University of Milano-Bicocca, Milan, Italy

Rocco Zaccagnino

Dip. di Informatica, University of Salerno, Fisciano, Italy

Rosalba Zizza

Dip. di Informatica, University of Salerno, Fisciano, Italy

Abstract

The notion of Lyndon word and Lyndon factorization has shown to have unexpected applications in theory as well as in developing novel algorithms on words. A counterpart to these notions are those of inverse Lyndon word and inverse Lyndon factorization. Differently from the Lyndon words, the inverse Lyndon words may be bordered. The relationship between the two factorizations is related to the inverse lexicographic ordering, and has only been recently explored. More precisely, a main open question is how to get an inverse Lyndon factorization from a classical Lyndon factorization under the inverse lexicographic ordering, named CFL_{in}. In this paper we reveal a strong connection between these two factorizations where the border plays a relevant role. More precisely, we show two main results. We say that a factorization has the border property if a nonempty border of a factor cannot be a prefix of the next factor. First we show that there exists a unique inverse Lyndon factorization having the border property. Then we show that this unique factorization with the border property is the so-called canonical inverse Lyndon factorization, named ICFL. By showing that ICFL is obtained by compacting factors of the Lyndon factorization over the inverse lexicographic ordering, we provide a linear time algorithm for computing ICFL from CFL_{in}.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Lyndon words, Lyndon factorization, Combinatorial algorithms on words

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.31

Funding *Paola Bonizzoni*: from the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement PANGAIA No. 872539, and from MUR 2022YRB97K, PINC, Pangenome INformatiCs: from Theory to Applications.

Clelia De Felice: from MUR 2022YRB97K, PINC, Pangenome INformatiCs: from Theory to Applications.

Brian Riccardi: from the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement PANGAIA No. 872539, and from MUR 2022YRB97K, PINC, Pangenome INformatiCs: from Theory to Applications.

Rocco Zaccagnino: from INdAM-GNCS Project 2023.

Rosalba Zizza: from INdAM-GNCS Project 2023.

[1] Corresponding author

[2] Corresponding author

1 Introduction

The theoretical investigation of combinatorial properties of well-known word factorizations is a research topic that recently have witnessed special interest especially for improving the efficiency of algorithms [5]. Among these, the *Lyndon Factorization* introduced by Chen, Fox, Lyndon in [12], named CFL, undoubtedly stands. Any word w admits a unique factorization CFL(w), that is a lexicographically non-increasing sequence of factors which are *Lyndon words*. A Lyndon word w is strictly lexicographically smaller than each of its proper cyclic shifts, or, equivalently, than each of its nonempty proper suffixes [24]. Interesting applications of the use of the Lyndon factorization and Lyndon words are the development of the bijective Burrows-Wheeler Transforms [2, 6, 21] and a novel algorithm for sorting suffixes [5]. In particular, the notion of a Lyndon word has been re-discovered various times as a theoretical tool to locate short motifs [15] and relevant k-mers in bioinformatics applications [26]. In this line of research, Lyndon-based word factorizations have been explored to define a novel feature representation for biological sequences based on theoretical combinatorial properties proved to capture sequence similarities [7].

The notion of a *Lyndon word* has a counterpart that is the notion of an *inverse Lyndon word*, i.e., a word lexicographically greater than its suffixes. Inverting the relation between a word and its suffixes, as between Lyndon words and inverse Lyndon words, leads to different properties. Indeed, although a word could admit more than one *inverse Lyndon factorization*, that is a factorization into a nonincreasing product of *inverse Lyndon words*, in [8] the *Canonical Inverse Lyndon Factorization*, named ICFL, was introduced. ICFL maintains the main properties of CFL: it is unique and can be computed in linear time. In addition, it maintains a similar *Compatibility Property*, used for obtaining the sorting of the suffixes of w ("global suffixes") by using the sorting of the suffixes of each factor of CFL(w) ("local suffixes") [25]. Most notably, ICFL(w) has another interesting property [8, 9, 10]: we can provide an upper bound on the length of the longest common prefix of two substrings of a word w starting from different positions.

A relationship between ICFL(w) and CFL(w) has been proved by using the notion of *grouping* [8]. First, let $\mathrm{CFL}_{in}(w)$ be the Lyndon factorization of w with respect to the inverse lexicographic order, it is proved that ICFL(w) is obtained by concatenating the factors of a non-increasing maximal chain with respect to the prefix order, denoted by $\mathcal{PMC}_w$, in $\mathrm{CFL}_{in}(w)$ (see Section 6). Despite this result, the connection between $\mathrm{CFL}_{in}(w)$ and the inverse Lyndon factorization still remained obscure, mainly by the fact that a word may have multiple inverse Lyndon factorizations.

In this paper, we explore this connection between CFL_{in} and the inverse Lyndon factorizations. Our first main contribution consists in showing that there is a unique inverse Lyndon factorization of a word that has *border property*. The border property states that any nonempty border of a factor cannot be a prefix of the next factor. We further highlight the aforementioned connection by proving that the inverse Lyndon factorization with the border property is a *compact* factorization (Definition 6.7), i.e., each inverse Lyndon factor is the concatenation of *compact factors*. In turn, a compact factor is the concatenation of the longest sequence of identical words in a $\mathcal{PMC}$. We then show the second contribution of this paper: this unique factorization is ICFL itself and then provide a simpler linear time algorithm for computing ICFL. Our algorithm is based on a new property that characterizes ICFL(w): the last factor in ICFL(w) is the longest suffix of w that is an inverse Lyndon word. Recall that the Lyndon factorization of w has a similar property: the last factor is the longest suffix of w that is a Lyndon word.

2 Words

Throughout this paper we follow [4, 13, 22, 23, 27] for the notations. We denote by Σ^* the *free monoid* generated by a finite alphabet Σ and we set $\Sigma^+ = \Sigma^* \setminus \{1\}$, where 1 is the empty word. For a word $w \in \Sigma^*$, we denote by $|w|$ its *length*. A word $x \in \Sigma^*$ is a *factor* of $w \in \Sigma^*$ if there are $u_1, u_2 \in \Sigma^*$ such that $w = u_1 x u_2$. If $u_1 = 1$ (resp. $u_2 = 1$), then x is a *prefix* (resp. *suffix*) of w. A factor (resp. prefix, suffix) x of w is *proper* if $x \neq w$. Two words x, y are *incomparable* for the prefix order, denoted as $x \bowtie y$, if neither x is a prefix of y nor y is a prefix of x. Otherwise, x, y are *comparable* for the prefix order. We write $x \leq_p y$ if x is a prefix of y and $x \geq_p y$ if y is a prefix of x. The notion of a pair of words comparable (or incomparable) for the suffix order is defined symmetrically.

We recall that, given a nonempty word w, a *border* of w is a word which is both a proper prefix and a suffix of w [14]. The longest proper prefix of w which is a suffix of w is also called *the border* of w [14, 23]. A word $w \in \Sigma^+$ is *bordered* if it has a nonempty border. Otherwise, w is *unbordered*. A nonempty word w is *primitive* if $w = x^k$ implies $k = 1$. An unbordered word is primitive. A *sesquipower* of a word x is a word $w = x^n p$ where p is a proper prefix of x and $n \geq 1$. Two words x, y are called *conjugate* if there exist words u, v such that $x = uv, y = vu$. The conjugacy relation is an equivalence relation. A conjugacy class is a class of this equivalence relation.

▶ **Definition 2.1.** *Let $(\Sigma, <)$ be a totally ordered alphabet. The lexicographic (or alphabetic order) $\prec$ on $(\Sigma^*, <)$ is defined by setting $x \prec y$ if*

- *x is a proper prefix of y, or*
- *$x = ras$, $y = rbt$, $a < b$, for $a, b \in \Sigma$ and $r, s, t \in \Sigma^*$.*

In the next part of the paper we will implicitly refer to totally ordered alphabets. For two nonempty words x, y, we write $x \ll y$ if $x \prec y$ and x is not a proper prefix of y [3]. We also write $y \succ x$ if $x \prec y$. Basic properties of the lexicographic order are recalled below.

▶ **Lemma 2.2.** *For $x, y \in \Sigma^+$, the following properties hold.*

(1) *$x \prec y$ if and only if $zx \prec zy$, for every word z.*
(2) *If $x \ll y$, then $xu \ll yv$ for all words u, v.*
(3) *If $x \prec y \prec xz$ for a word z, then $y = xy'$ for some word y' such that $y' \prec z$.*
(4) *If $x \ll y$ and $y \ll z$, then $x \ll z$.*

Let t, j, r_j be positive integers, with $1 \leq j \leq t$. Let $\mathcal{S}_1, \ldots, \mathcal{S}_t$ be sequences, with $\mathcal{S}_j = (s_{j,1}, \ldots, s_{j,r_j})$. We let $(\mathcal{S}_1, \ldots, \mathcal{S}_t)$ stand for the sequence $(s_{1,1}, \ldots, s_{1,r_1}, \ldots, s_{t,1}, \ldots, s_{t,r_t})$.

3 Lyndon words

▶ **Definition 3.1.** *A Lyndon word $w \in \Sigma^+$ is a word which is primitive and the smallest one in its conjugacy class for the lexicographic order.*

▶ **Example 3.2.** Let $\Sigma = \{a, b\}$ with $a < b$. The words a, b, $aaab$, $abbb$, $aabab$ and $aababaabb$ are Lyndon words. On the contrary, $abab$, aba and $abaab$ are not Lyndon words.

▶ **Proposition 3.3.** *Each Lyndon word w is unbordered.*

A class of conjugacy is also called a *necklace* and often identified with the minimal word for the lexicographic order in it. We will adopt this terminology. Then a word is a necklace if and only if it is a power of a Lyndon word. A *prenecklace* is a prefix of a necklace. Then clearly any nonempty prenecklace w has the form $w = (uv)^k u$, where uv is a Lyndon word, $u \in \Sigma^*$, $v \in \Sigma^+$, $k \geq 1$, that is, w is a sesquipower of a Lyndon word uv. The following result has been proved in [16]. It shows that the nonempty prefixes of Lyndon words are exactly the nonempty prefixes of the powers of Lyndon words with the exclusion of c^k, where c is the maximal letter and $k \geq 2$.

▶ **Proposition 3.4.** *A word is a nonempty prefix of a Lyndon word if and only if it is a sesquipower of a Lyndon word distinct of c^k, where c is the maximal letter and $k \geq 2$.*

In the following $L = L_{(\Sigma^*,<)}$ will be the set of Lyndon words, totally ordered by the relation $\prec$ on $(\Sigma^*,<)$.

▶ **Theorem 3.5.** *Any word $w \in \Sigma^+$ can be written in a unique way as a nonincreasing product $w = \ell_1 \ell_2 \cdots \ell_h$ of Lyndon words, i.e., in the form*

$$w \quad = \quad \ell_1 \ell_2 \cdots \ell_h, \ \text{with } \ell_j \in L \text{ and } \ell_1 \succeq \ell_2 \succeq \ldots \succeq \ell_h \tag{3.1}$$

The sequence $\mathrm{CFL}(w) = (\ell_1, \ldots, \ell_h)$ in Eq. (3.1) is called the *Lyndon decomposition* (or *Lyndon factorization*) of w. It is denoted by $\mathrm{CFL}(w)$ because Theorem 3.5 is usually credited to Chen, Fox and Lyndon [12]. The following result, proved in [16], is necessary for our aims.

▶ **Corollary 3.6.** *Let $w \in \Sigma^+$, let ℓ_1 be its longest prefix which is a Lyndon word and let w' be such that $w = \ell_1 w'$. If $w' \neq 1$, then $\mathrm{CFL}(w) = (\ell_1, \mathrm{CFL}(w'))$.*

Sometimes we need to emphasize consecutive equal factors in CFL. We write $\mathrm{CFL}(w) = (\ell_1^{n_1}, \ldots, \ell_r^{n_r})$ to denote a tuple of $n_1 + \ldots + n_r$ Lyndon words, where $r > 0$, $n_1, \ldots, n_r \geq 1$. Precisely $\ell_1 \succ \ldots \succ \ell_r$ are Lyndon words, also named *Lyndon factors* of w. There is a linear time algorithm to compute the pair (ℓ_1, n_1) and thus, by iteration, the Lyndon factorization of w [17, 23]. Linear time algorithms may also be found in [16] and in the more recent paper [19].

4 Inverse Lyndon words

For the material in this section see [8, 9, 10]. Inverse Lyndon words are related to the inverse alphabetic order. Their definition is recalled below.

▶ **Definition 4.1.** *Let $(\Sigma, <)$ be a totally ordered alphabet. The* inverse $<_{in}$ *of $<$ is defined by*

$$\forall a, b \in \Sigma \quad b <_{in} a \Leftrightarrow a < b$$

The inverse lexicographic *or* inverse alphabetic order *on $(\Sigma^*, <)$, denoted $\prec_{in}$, is the lexicographic order on $(\Sigma^*, <_{in})$.*

▶ **Example 4.2.** Let $\Sigma = \{a, b, c, d\}$ with $a < b < c < d$. Then $dab \prec dabd$ and $dabda \prec dac$. We have $d <_{in} c <_{in} b <_{in} a$. Therefore $dab \prec_{in} dabd$ and $dac \prec_{in} dabda$.

Of course for all $x, y \in \Sigma^*$ such that $x \bowtie y$,

$$y \prec_{in} x \Leftrightarrow x \prec y.$$

Moreover, in this case $x \ll y$. This justifies the adopted terminology.

From now on, $L_{in} = L_{(\Sigma^*,<_{in})}$ denotes the set of the Lyndon words on Σ^* with respect to the inverse lexicographic order. Following [18], a word $w \in L_{in}$ will be named an *anti-Lyndon word*. Correspondingly, an *anti-prenecklace* will be a prefix of an *anti-necklace*, which in turn will be a necklace with respect to the inverse lexicographic order.

In the following, we denote by $\mathrm{CFL}_{in}(w)$ the Lyndon factorization of w with respect to the inverse order $<_{in}$.

▶ **Definition 4.3.** *A word $w \in \Sigma^+$ is an inverse Lyndon word if $s \prec w$, for each nonempty proper suffix s of w.*

▶ **Example 4.4.** The words a, b, $aaaaa$, $bbba$, $baaab$, $bbaba$ and $bbababbaa$ are inverse Lyndon words on $\{a, b\}$, with $a < b$. On the contrary, $aaba$ is not an inverse Lyndon word since $aaba \prec ba$. Analogously, $aabba \prec ba$ and thus $aabba$ is not an inverse Lyndon word.

The following result, proved in [8, 10], and also in [11], summarizes some properties of the inverse Lyndon words.

▶ **Proposition 4.5.** *Let $w \in \Sigma^+$. Then we have*

1. *The word w is an anti-Lyndon word if and only if it is an unbordered inverse Lyndon word.*
2. *The word w is an inverse Lyndon word if and only if w is a nonempty anti-prenecklace.*
3. *If w is an inverse Lyndon word, then any nonempty prefix of w is an inverse Lyndon word.*

▶ **Definition 4.6.** *An inverse Lyndon factorization of a word $w \in \Sigma^+$ is a sequence $(m_1, \ldots, m_k)$ of inverse Lyndon words such that $m_1 \cdots m_k = w$ and $m_i \ll m_{i+1}$, $1 \leq i \leq k-1$.*

As the following example in [8] shows, a word may have different inverse Lyndon factorizations.

▶ **Example 4.7.** Let $\Sigma = \{a, b, c, d\}$ with $a < b < c < d$, $z = dabdadacddbdc$. It is easy to see that $(dab, dadacd, db, dc)$, $(dabda, dac, ddbdc)$, $(dab, dadac, ddbdc)$ are all inverse Lyndon factorizations of z.

5 The border property

In this section we prove the main result of this paper, namely, for any nonempty word w, there exists a unique inverse Lyndon factorization of w which has a special property, named the border property.

▶ **Definition 5.1** (Border property)**.** *Let $w \in \Sigma^+$. A factorization $(m_1, \ldots, m_k)$ of w* has the border property *if each nonempty border z of m_i is not a prefix of m_{i+1}, $1 \leq i \leq k-1$.*

We first prove a fundamental property of the inverse Lyndon factorizations of w which have the border property.

▶ **Lemma 5.2.** *Let $w \in \Sigma^+$, let $(m_1, \ldots, m_k)$ be an inverse Lyndon factorization of w having the border property. If α is a nonempty border of m_j, $1 \leq j \leq k-1$, then there exists a nonempty prefix β of m_{j+1} such that $|\beta| \leq |\alpha|$ and $\alpha \ll \beta$.*

Proof. Let $w \in \Sigma^+$, let $(m_1, \ldots, m_k)$ be an inverse Lyndon factorization of w having the border property, let α be a nonempty border of m_j, $1 \leq j \leq k-1$. We distinguish two cases: either $|m_{j+1}| < |\alpha|$ or $|m_{j+1}| \geq |\alpha|$.

Assume $|m_{j+1}| < |\alpha|$. By hypothesis $(m_1, \ldots, m_k)$ is an inverse Lyndon factorization, hence $m_j \ll m_{j+1}$, that is, there are $r, s, t \in \Sigma^*$, $a, b \in \Sigma$, such that $a < b$ and $m_j = ras$, $m_{j+1} = rbt$. Obviously $|ra| \leq |m_{j+1}| < |\alpha|$, thus there is $s' \in \Sigma^*$ such that $\alpha = ras'$. Consequently, $\alpha = ras' \ll rbt = m_{j+1}$ and our claim holds with $\beta = m_{j+1}$.

Assume $|m_{j+1}| \geq |\alpha|$. Let β be the nonempty prefix of m_{j+1} such that $|\beta| = |\alpha|$. Clearly $\beta \neq \alpha$ because $(m_1, \ldots, m_k)$ has the border property. Since α and β are two different nonempty words of the same length, either $\beta \ll \alpha$ or $\alpha \ll \beta$. The first case leads to a contradiction because if $\beta \ll \alpha$, then $m_{j+1} \ll m_j$ by Lemma 2.2 and this contradicts the fact that $(m_1, \ldots, m_k)$ is an inverse Lyndon factorization. Thus, $\alpha \ll \beta$ and the proof is complete. ◀

▶ **Proposition 5.3.** *For each $w \in \Sigma^+$, there exists a unique inverse Lyndon factorization of w having the border property.*

Proof. The proof is by induction on $|w|$. If $|w| = 1$, then the statement clearly holds. Thus assume $|w| > 1$. Let $F_1(w) = (f_1, \ldots, f_k)$ and $F_2(w) = (f'_1, \ldots, f'_v)$ be two inverse Lyndon factorizations of w having the border property. Thus

$$f_1 \cdots f_k = f'_1 \cdots f'_v = w \tag{5.1}$$

If $|f_k| = |f'_v|$ and $v = 1$ or $k = 1$, clearly $f_k = f'_v$ and $F_1(w) = F_2(w)$. Analogously, if $|f_k| = |f'_v|$, $v > 1$ and $k > 1$, then $f_k = f'_v$ and $F'_1(w') = (f_1, \ldots, f_{k-1})$, $F'_2(w') = (f'_1, \ldots, f'_{v-1})$ would be two inverse Lyndon factorizations of w' having the border property, where w' is such that $w = w' f_k$. Of course, $|w'| < |w|$. By induction hypothesis, $F'_1(w') = F'_2(w')$, hence $F_1(w) = F_2(w)$.

By contradiction, let $|f_k| \neq |f'_v|$. Assume $|f_k| < |f'_v|$ (similar arguments apply if $|f_k| > |f'_v|$). The word f_k is a proper suffix of f'_v. Clearly $k > 1$. Let g be the smallest integer such that $f_{g+1} \cdots f_k$ is a proper suffix of f'_v, $1 \leq g \leq k-1$, that is,

$$f'_v = \alpha f_{g+1} \cdots f_k \tag{5.2}$$

where $\alpha \in \Sigma^+$ is a suffix of f_g.

Notice that

$$\alpha \not\ll f_{g+1}. \tag{5.3}$$

Indeed, if $\alpha \ll f_{g+1}$, then, by Eq. (5.2), we would have $f'_v = \alpha f_{g+1} \cdots f_k \ll f_{g+1} \cdots f_k$, which is impossible because f'_v is an inverse Lyndon word.

The word α is a nonempty proper suffix of f_g since otherwise we would have $\alpha = f_g \ll f_{g+1}$, contrary to Eq. (5.3). Since f_g is an inverse Lyndon word and α is a nonempty proper suffix of f_g, either $\alpha \leq_p f_g$ or $\alpha \ll f_g$.

If $\alpha \leq_p f_g$, then α is a nonempty border of f_g, then, by Lemma 5.2, there exists a nonempty prefix β of f_{g+1} such that $|\beta| \leq |\alpha|$ and $\alpha \ll \beta$. Thus, $\alpha \ll f_{g+1}$ which contradicts Eq. (5.3). Assume $\alpha \ll f_g$. Since $f_g \ll f_{g+1}$, by Lemma 2.2 we have $\alpha \ll f_{g+1}$ which contradicts once again Eq. (5.3). This finishes the proof. ◀

▶ **Example 5.4.** Let $\Sigma = \{a, b, c, d\}$ with $a < b < c < d$, let $z = dabdadacddbdc$. Notice that only the last one of the inverse Lyndon factorizations of z from Example 4.7 fulfils the border property, and the others do not.

6 Groupings and compact factorizations

In this section we prove a structural property of an inverse Lyndon factorization having the border property, namely it is a *compact factorization*. This result is crucial to characterize the relationship between $\mathrm{CFL}_{in}(w)$ and the factorization into inverse Lyndon words of w. First we report the notion of *grouping* given in [8]. We refer to [8, 10] for a detailed and complete discussion on this topic.

Let $\mathrm{CFL}_{in}(w) = (\ell_1, \ldots, \ell_h)$, where $\ell_1 \succeq_{in} \ell_2 \succeq_{in} \ldots \succeq_{in} \ell_h$. Consider the partial order $\geq_p$, where $x \geq_p y$ if y is a prefix of x. Recall that a *chain* is a set of a pairwise comparable elements. We say that a chain is maximal if it is not strictly contained in any other chain. A non-increasing *(maximal) chain* in $\mathrm{CFL}_{in}(w)$ is the sequence corresponding to a (maximal) chain in the multiset $\{\ell_1, \dots, \ell_h\}$ with respect to $\geq_p$. We denote by $\mathcal{PMC}_w$, or simply $\mathcal{PMC}$ when it is understood, a non-increasing maximal chain in $\mathrm{CFL}_{in}(w)$. Looking at the definition of the (inverse) lexicographic order, it is easy to see that a $\mathcal{PMC}$ is a sequence of consecutive factors in $\mathrm{CFL}_{in}(w)$. Moreover $\mathrm{CFL}_{in}(w)$ is the concatenation of its $\mathcal{PMC}$. The formal definitions are given below.

▶ **Definition 6.1.** *Let $w \in \Sigma^+$, let $\mathrm{CFL}_{in}(w) = (\ell_1, \dots, \ell_h)$ and let $1 \leq r < s \leq h$. We say that $\ell_r, \ell_{r+1}, \dots, \ell_s$ is a non-increasing* maximal chain for the prefix order *in $\mathrm{CFL}_{in}(w)$, abbreviated $\mathcal{PMC}$, if $\ell_r \geq_p \ell_{r+1} \geq_p \dots \geq_p \ell_s$. Moreover, if $r > 1$, then $\ell_{r-1} \not\geq_p \ell_r$, if $s < h$, then $\ell_s \not\geq_p \ell_{s+1}$. Two $\mathcal{PMC}$ $\mathcal{C}_1 = \ell_r, \ell_{r+1}, \dots, \ell_s$, $\mathcal{C}_2 = \ell_{r'}, \ell_{r'+1}, \dots, \ell_{s'}$ are* consecutive *if $r' = s + 1$ (or $r = s' + 1$).*

▶ **Definition 6.2.** *Let $w \in \Sigma^+$, let $\mathrm{CFL}_{in}(w) = (\ell_1, \dots, \ell_h)$. We say that $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_s)$ is* the decomposition *of $\mathrm{CFL}_{in}(w)$ into its non-increasing maximal chains for the prefix order if the following holds*

(1) *Each $\mathcal{C}_j$ is a non-increasing maximal chain in $\mathrm{CFL}_{in}(w)$.*

(2) *$\mathcal{C}_j$ and $\mathcal{C}_{j+1}$ are consecutive, $1 \leq j \leq s - 1$.*

(3) *$\mathrm{CFL}_{in}(w)$ is the concatenation of the sequences $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_s$.*

▶ **Definition 6.3.** *Let $w \in \Sigma^+$. We say that $(m_1, \dots, m_k)$ is a* grouping *of $\mathrm{CFL}_{in}(w)$ if the following holds*

(1) *$(m_1, \dots, m_k)$ is an inverse Lyndon factorization of w*

(2) *Each factor m_j, is the product of consecutive factors in a $\mathcal{PMC}$ in $\mathrm{CFL}_{in}(w)$.*

▶ **Example 6.4.** Let $\Sigma = \{a, b, c, d\}$, $a < b < c < d$, and $w = dabadabdabdadac$. We have $\mathrm{CFL}_{in}(w) = (daba, dab, dab, dadac)$. The decomposition of $\mathrm{CFL}_{in}(w)$ into its $\mathcal{PMC}$ is $((daba, dab, dab), (dadac))$. Moreover, $(daba, dabdab, dadac)$ is a grouping of $\mathrm{CFL}_{in}(w)$ but for the inverse Lyndon factorization $(dabadab, dabda, dac)$ this is no longer true.

Next, let $y = dabadabdabdabdadac$. We have $\mathrm{CFL}_{in}(y) = (daba, dab, dab, dab, dadac)$. The decomposition of $\mathrm{CFL}_{in}(w)$ into its $\mathcal{PMC}$ is $((daba, dab, dab, dab), (dadac))$. Moreover, $(daba, (dab)^3, dadac)$ and $(dabadab, (dab)^2, dadac)$ are two groupings of $\mathrm{CFL}_{in}(y)$.

For our aims, we need to consider the words that are concatenations of equal factors in CFL_{in}. This approach leads to a refinement of the partition of CFL_{in} into non-increasing maximal chains for the prefix order, as defined below.

▶ **Definition 6.5** (Compact sequences). *Let $\mathcal{C} = (\ell_1, \dots, \ell_h)$ be a non-increasing maximal chain for the prefix order in $\mathrm{CFL}_{in}(w)$. The decomposition of $\mathcal{C}$ into* maximal compact sequences *is the sequence $(\mathcal{G}_1, \dots, \mathcal{G}_n)$ such that*

(1) *$\mathcal{C} = (\mathcal{G}_1, \dots, \mathcal{G}_n)$*

(2) *For every i, $1 \leq i \leq n$, $\mathcal{G}_i$ consists of the longest sequence of consecutive identical elements in $\mathcal{C}$*

Let $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_s)$ be the decomposition of $\mathrm{CFL}_{in}(w)$ into its non-increasing maximal chains for the prefix order. The decomposition of $\mathrm{CFL}_{in}(w)$ into its maximal compact sequences is obtained by replacing each $\mathcal{C}_j$ in $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_s)$ with its decomposition into maximal compact sequences.

▶ **Definition 6.6** (Compact factor). *Let $(\mathcal{G}_1, \dots, \mathcal{G}_n)$ be the decomposition of $\mathrm{CFL}_{in}(w)$ into its maximal compact sequences. For every i, $1 \leq i \leq n$, the concatenation g_i of the elements in $\mathcal{G}_i$ is a* compact factor *in $\mathrm{CFL}_{in}(w)$.*

▶ **Definition 6.7** (Compact factorization). *Let $w \in \Sigma^+$. We say that $(m_1, \ldots, m_k)$ is a* compact factorization *of w if $(m_1, \ldots, m_k)$ is an inverse Lyndon factorization of w and each m_j, $1 \leq j \leq k$, is a concatenation of compact factors in $\mathrm{CFL}_{in}(w)$.*

▶ **Example 6.8.** Consider again $y = dabadabdabdabdadac$ over $\Sigma = \{a, b, c, d\}$, $a < b < c < d$, as in Example 6.4. The decomposition of $\mathrm{CFL}_{in}(y) = (daba, dab, dab, dab, dadac)$ into its maximal compact sequences is $((daba), (dab, dab, dab), (dadac))$. The compact factors in $\mathrm{CFL}_{in}(w)$ are $daba, (dab)^3, dadac$. Moreover, $(daba, (dab)^3, dadac)$ is a compact factorization whereas $(dabadab, (dab)^2, dadac)$ is a grouping of $\mathrm{CFL}_{in}(y)$ which is not a compact factorization.

▶ **Proposition 6.9.** *Let $w \in \Sigma^+$. If $(m_1, \ldots, m_k)$ is an inverse Lyndon factorization of w having the border property, then $(m_1, \ldots, m_k)$ is a compact factorization of w.*

Proof. Let $w \in \Sigma^+$, let $(m_1, \ldots, m_k)$ be an inverse Lyndon factorization of w having the border property. Let $\mathrm{CFL}_{in}(w) = (\ell_1, \ldots, \ell_h)$, where $\ell_1 \succeq_{in} \ell_2 \succeq_{in} \ldots \succeq_{in} \ell_h$ and $\ell_1, \ldots, \ell_h$ are anti-Lyndon words. First we prove that $(m_1, \ldots, m_k)$ is a grouping of $\mathrm{CFL}_{in}(w)$ by induction on $|w|$. If $|w| = 1$ the statement clearly holds, thus assume $|w| > 1$.

The words m_1 and ℓ_1 are comparable for the prefix order, hence either m_1 is a proper prefix of ℓ_1 or ℓ_1 is a prefix of m_1. Suppose that m_1 is a proper prefix of ℓ_1. Thus, there are j, $1 < j \leq k$, and $x, y \in \Sigma^*$, $x \neq 1$, such that $m_j = xy$ and $\ell_1 = m_1 \cdots m_{j-1}x$. Necessarily it turns out $j = 2$ because otherwise $m_1 \ll m_{j-1}$, hence, by Lemma 2.2, $\ell_1 \ll m_{j-1}x$ and this contradicts the fact that ℓ_1 is an anti-Lyndon word. In conclusion $\ell_1 = m_1x$ and $m_2 = xy$. We know that $m_1 \ll m_2$, that is, there are $r, s, t \in \Sigma^*$, $a, b \in \Sigma$, such that $a < b$ and $m_1 = ras$, $m_2 = rbt = xy$. If $|x| \leq |r|$, then x is a nonempty border of ℓ_1 and if $|x| > |r|$, then there is a word t' such that $x = rbt'$ which implies $\ell_1 \ll x$. Both cases again contradict the fact that ℓ_1 is an anti-Lyndon word.

Therefore, ℓ_1 is a prefix of m_1. If $m_1 = \ell_1 \cdots \ell_h = w$, then $k = 1$ and the statement is proved. Otherwise, let i be the largest integer such that $m_1 = \ell_1 \cdots \ell_{i-1}x$, $x, y \in \Sigma^*$, $\ell_i = xy$, $1 < i \leq h$, $y \neq 1$. Let $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s)$ be the decomposition of $\mathrm{CFL}_{in}(w)$ into its non-increasing maximal chains for the prefix order. We claim that $\ell_1 \cdots \ell_{i-1}$ is a prefix of the concatenation of the elements of $\mathcal{C}_1$, thus $(\ell_1, \ldots, \ell_{i-1})$ is a chain for the prefix order. If $i = 1$ we are done. Let $i > 1$. By contradiction, assume that there is j, $1 < j < i$, such that $\ell_j \notin \mathcal{C}_1$. Therefore, $\ell_1 \ll \ell_j$ which implies $m_1 \ll \ell_j \cdots \ell_{i-1}x$ and this contradicts the fact that m_1 is an inverse Lyndon word.

We now prove that $x = 1$. Assume $x \neq 1$. As a preliminary step, we prove that there is no nonempty prefix β of m_2 such that $|\beta| \leq |x|$ and $x \ll \beta$. In fact, if such a prefix existed, there would be $r, s, t \in \Sigma^*$, $a, b \in \Sigma$, such that $a < b$ and $x = ras$, $\beta = rbt$. Notice that y is a nonempty prefix of $m_2 \cdots m_k$, thus y and $\beta = rbt$ are comparable for the prefix order. If $|\ell_i| = |xy| \leq |xr|$, then $0 < |y| \leq |r|$ and y would be a nonempty prefix of r. Thus y would be a nonempty border of ℓ_i. If $|\ell_i| = |xy| > |xr|$, then there would be a word t' such that $\ell_i = rasrbt'$ which would imply $\ell_i \ll rbt'$. Both cases contradict the fact that ℓ_i is an anti-Lyndon word.

Now either ℓ_i is a prefix of ℓ_1 or $\ell_1 \ll \ell_i$. If ℓ_i were a prefix of ℓ_1, then x would be a nonempty border of m_1. By Lemma 5.2 there would exist a nonempty prefix β of m_2 such that $|\beta| \leq |x|$ and $x \ll \beta$ which contradicts our preliminary step.

If it were true that $\ell_1 \ll \ell_i$ then there would be $r, s, t \in \Sigma^*$, $a, b \in \Sigma$, such that $a < b$ and $\ell_1 = ras$, $\ell_i = rbt = xy$. If $|x| > |r|$, then there would be a word t' such that $x = rbt'$ which would imply $m_1 \ll x$ and this contradicts the fact that m_1 is an inverse Lyndon word. If $|x| \leq |r|$, then x would be a prefix of r and x would be a nonempty border of m_1. By Lemma 5.2 again, there would exist a nonempty prefix β of m_2 such that $|\beta| \leq |x|$ and $x \ll \beta$ which contradicts again our preliminary step.

Let $w' \in \Sigma^*$ be such that $w = m_1 w'$. We know that $w' \neq 1$ and clearly $|w'| < |w|$. Of course $(m_2, \ldots, m_k)$ is an inverse Lyndon factorization of w having the border property. Moreover, by Corollary 3.6, $\mathrm{CFL}_{in}(w') = (\ell_i, \ldots, \ell_h)$ and $(\mathcal{C}'_1, \mathcal{C}_2, \ldots, \mathcal{C}_s)$ is the decomposition of $\mathrm{CFL}_{in}(w')$ into its non-increasing maximal chains for the prefix order, where $\mathcal{C}'_1$ is defined by $\mathcal{C}_1 = (\ell_1, \ldots, \ell_{i-1}, \mathcal{C}'_1)$. By induction hypothesis, $(m_2, \ldots, m_k)$ is a grouping of $\mathrm{CFL}_{in}(w')$ and consequently $(m_1, \ldots, m_k)$ is a grouping of $\mathrm{CFL}_{in}(w)$.

Finally, to obtain a contradiction, suppose that $(m_1, \ldots, m_k)$ is a grouping of $\mathrm{CFL}_{in}(w)$ having the border property such that $(m_1, \ldots, m_k)$ is not a compact factorization of w. To adapt the notation to the proof, set $\mathrm{CFL}_{in}(w) = (\ell_1^{n_1}, \ldots, \ell_r^{n_r})$, where $r > 0$, $n_1, \ldots, n_r \geq 1$ and $\ell_1, \ldots, \ell_r$ are anti-Lyndon words. By Definitions 6.3 and 6.7, there exist integers j, h, p_h, q_h, $1 \leq j \leq k-1$, $1 \leq h \leq r$, $p_h \geq 1$, $q_h \geq 1$, $p_h + q_h \leq n_h$, such that m_j ends with $\ell_h^{p_h}$ and m_{j+1} starts with $\ell_h^{q_h}$. Thus, by Definition 6.3, ℓ_h is a prefix of m_j. Moreover, ℓ_h is a proper prefix of m_j. Indeed otherwise $\ell_h = m_j \leq_p m_{j+1}$ which is impossible because $m_j \ll m_{j+1}$ ($(m_1, \ldots, m_k)$ is an inverse Lyndon factorization). Thus ℓ_h is a nonempty border of m_j. The word ℓ_h is also a prefix of m_{j+1} and this contradicts the fact that $(m_1, \ldots, m_k)$ has the border property. ◀

7 The canonical inverse Lyndon factorization: The algorithm

In this section we state another relevant result of the paper related to the main one stated in Section 5. We have shown that a nonempty word w can have more than one inverse Lyndon factorization but w has a unique inverse Lyndon factorization with the border property (Example 4.7, Proposition 5.3). Below we highlight that this unique factorization is the canonical one defined in [8, 10].

This special inverse Lyndon factorization is denoted by ICFL because it is the counterpart of the Lyndon factorization CFL of w, when we use (I)inverse words as factors. Indeed, in [8] it has been proved that $\mathrm{ICFL}(w)$ can be computed in linear time and it is uniquely determined for a word w.

In Proposition 7.7 we show another interesting property of ICFL: the last factor of the factorization is the longest suffix that is an inverse Lyndon word. Based on this result we provide a new simpler linear algorithm for computing ICFL.

We begin by recalling previously proved results on ICFL, namely Proposition 7.7 in [8] and Proposition 9.5 in [10]. They are merged into Proposition 7.1.

▶ **Proposition 7.1.** *For any $w \in \Sigma^+$, $\mathrm{ICFL}(w)$ is a grouping of $\mathrm{CFL}_{in}(w)$. Moreover, $\mathrm{ICFL}(w)$ has the border property.*

Corollary 7.2 is a direct consequence of Propositions 5.3, 6.9 and 7.1.

▶ **Corollary 7.2.** *For each $w \in \Sigma^+$, $\mathrm{ICFL}(w)$ is a compact factorization and it is is the unique inverse Lyndon factorization of w having the border property.*

Since $\mathrm{ICFL}(w)$ is the unique inverse Lyndon factorization with the border property, from now on these two notions will be synonymous. Proposition 7.3 has been proved in [11].

▶ **Proposition 7.3.** *Let $w \in \Sigma^+$, let $\mathrm{CFL}_{in}(w) = (\ell_1, \ldots, \ell_h)$ and let $(\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s)$ be the decomposition of $\mathrm{CFL}_{in}(w)$ into its non-increasing maximal chains for the prefix order. Let $w_1, \ldots, w_s$ be words such that $\mathrm{CFL}_{in}(w_j) = \mathcal{C}_j$, $1 \leq j \leq s$. Then $\mathrm{ICFL}(w)$ is the concatenation of the sequences $\mathrm{ICFL}(w_1), \ldots, \mathrm{ICFL}(w_s)$, that is,*

$$\mathrm{ICFL}(w) = (\mathrm{ICFL}(w_1), \ldots, \mathrm{ICFL}(w_s)) \tag{7.1}$$

We can now state some results useful to prove the correctness of our algorithm. First we observe that, thanks to Corollary 7.2 and Proposition 7.3, to compute ICFL we can limit ourselves to the case in which CFL_{in} is a chain with respect to the prefix order.

▶ **Lemma 7.4.** *Let $\ell_1, \ldots, \ell_h$ be anti-Lyndon words over Σ that form a non-increasing chain for the prefix order, that is, $\ell_1 \geq_p \ell_2 \geq_p \ldots \geq_p \ell_h$. If $\ell_1 \neq \ell_2$, then $\ell_1 \not\leq_p \ell_2 \cdots \ell_h$.*

Proof. By contradiction, assume that ℓ_1 is a prefix of $\ell_2 \cdots \ell_h$. Then, $\ell_1 = \ell_2 \cdots \ell_t z$ where either $z = 1$ and $2 < t \leq h$ or z is a nonempty prefix of ℓ_{t+1}, $2 \leq t < h$. Thus either ℓ_t or z is a nonempty border of ℓ_1, a contradiction in both cases. ◀

▶ Remark 7.5. [10] Let x, y be two different borders of $w \in \Sigma^+$. If x is shorter than y, then x is a border of y.

▶ **Proposition 7.6.** *Let $w \in \Sigma^+$ and assume that $\mathrm{CFL}_{in}(w)$ form a non-increasing chain for the prefix order. If $(m_1, \ldots, m_k)$ is a factorization of w such that each m_j, $1 \leq j \leq k$, is a concatenation of compact factors in $\mathrm{CFL}_{in}(w)$, then $(m_1, \ldots, m_k)$ has the border property.*

Proof. Let $w \in \Sigma^+$ and assume that $\mathrm{CFL}_{in}(w)$ form a non-increasing chain for the prefix order. Let $(m_1, \ldots, m_k)$ be a factorization of w such that each m_j, $1 \leq j \leq k$, is a concatenation of compact factors in $\mathrm{CFL}_{in}(w)$. The proof is by induction on k. If $k = 1$, then the conclusion follows immediately. Assume $k > 1$.

Let $w' \in \Sigma^+$ be such that $w = m_1 w'$. It is clear that $(m_2, \ldots, m_k)$ is a factorization of w' such that each m_j, $2 \leq j \leq k$, is a concatenation of compact factors in $\mathrm{CFL}_{in}(w')$. Thus, by the induction hypothesis, $(m_2, \ldots, m_k)$ has the border property. It remains to prove that each nonempty border of m_1 is not a prefix of m_2. The proof is straightforward if m_1 is unbordered, thus assume that m_1 is bordered.

Let $\mathrm{CFL}_{in}(w) = (\ell_1^{n_1}, \ldots, \ell_r^{n_r})$, where $\ell_1^{n_1}, \ldots, \ell_r^{n_r}$ are the compact factors in $\mathrm{CFL}_{in}(w)$, that is, $\ell_1, \ldots, \ell_r$ are anti-Lyndon words such that $\ell_1 \geq_p \ldots \geq_p \ell_r$. Since m_1 is a concatenation of compact factors in $\mathrm{CFL}_{in}(w)$, there is h, $1 \leq h < r$ such that

$$m_1 = \ell_1^{n_1} \cdots \ell_h^{n_h}.$$

Notice that ℓ_h is a nonempty border of m_1. Furthermore, since ℓ_h is unbordered, ℓ_h is the shortest nonempty border of m_1.

If there were a word z which is a nonempty border of m_1 and also a prefix of m_2, by Remark 7.5, ℓ_h would be a prefix of m_2. Therefore, ℓ_h would be a prefix of the word $\ell_{h+1}^{n_{h+1}} \cdots \ell_r^{n_r}$ which contradicts Lemma 7.4. ◀

▶ **Proposition 7.7.** *Let $w \in \Sigma^+$ and let $\mathrm{ICFL}(w) = (m_1, \ldots, m_k)$ be the unique inverse Lyndon factorization of w having the border property. Then m_k is the longest suffix of w which is an inverse Lyndon word.*

Proof. Let $w \in \Sigma^+$ and let $(m_1, \ldots, m_k)$ be the unique inverse Lyndon factorization of w having the border property. If $k = 1$ we are done. Thus suppose $k > 1$. By contradiction, suppose that m_k is not the longest suffix of w that is an inverse Lyndon word. Let s be such longest suffix. Thus, there exist a nonempty suffix x of m_j, $1 \leq j < k$ such that $s = x m_{j+1} \cdots m_k$. Furthermore x must be a proper suffix of m_j or we would have $s = m_j \cdots m_k \ll m_{j+1} \cdots m_k$ contradicting the hypothesis that s is inverse Lyndon.

We claim that $x \ll m_{j+1}$. Indeed, since m_j is an inverse Lyndon word, it holds $x \preceq m_j$. Thus, if $x \ll m_j$ or $x = m_j$, it immediately follows that $x \ll m_{j+1}$. Otherwise, $x \leq_p m_j$ and x is a nonempty border of m_j. By Lemma 5.2 applied to $(m_1, \ldots, m_k)$, with $x = \alpha$, there must exist a prefix β of m_{j+1} such that $x \ll \beta$, hence $x \ll m_{j+1}$.

Since $x \ll m_{j+1}$, we have $s = xm_{j+1} \cdots m_k \ll m_{j+1} \cdots m_k$, contradicting the hypothesis that s is an inverse Lyndon word. ◀

▶ Remark 7.8. Let $w \in \Sigma^+$ and let $\mathrm{ICFL}(w) = (m_1, \ldots, m_k)$ with $k > 1$. Let w' be such that $w = w'm_k$. Then, by Propositions 5.3 and 7.7, we obtain $\mathrm{ICFL}(w) = (\mathrm{ICFL}(w'), m_k)$.

Proposition 7.9 allows us to determine the longest suffix m' of a word w such that m' is an inverse Lyndon word.

▶ **Proposition 7.9.** *Let $w \in \Sigma^+$ be an inverse Lyndon word, and let $\ell \in \Sigma^+$ be an anti-Lyndon word. Then:*

1. *If $\ell \ll w$, then for every $k \geq 1$, $\ell^k w$ is not an inverse Lyndon word.*
2. *If ℓw is not an inverse Lyndon word, then $\ell \ll w$. Furthermore, for every $k \geq 1$, w is the longest suffix of $\ell^k w$ that is an inverse Lyndon word.*

Proof. By Lemma 2.2, the proof of 1. is immediate. Suppose ℓw is not inverse Lyndon. Then, there exists a proper suffix s of ℓw such that $\ell w \preceq s$, hence $\ell w \ll s$. Since ℓ is anti-Lyndon, for every proper suffix x of ℓ it follows $x \ll \ell$ and consequently $xw \ll \ell w$. Thus, s must be a suffix of w. Since w is an inverse Lyndon word, one of the following three cases holds: (1) $w = s$; (2) $s <_p w$; (3) $s \ll w$. By $\ell w \ll s$, in each of the three cases it is evident that $\ell w \ll w$. Thus there are $r, t, t' \in \Sigma^*$ and $a, b \in \Sigma$ with $a < b$ such that $\ell w = rat$, $w = rbt'$. If $|\ell| \geq |ra|$, then clearly $\ell \ll w$. Otherwise, $|\ell| \leq |r|$ and there is $r' \in \Sigma^*$ such that $r = \ell r'$. Consequently, by $\ell w = rat = \ell r'at$, we obtain $w = r'at$. Hence $w = rbt' = \ell r'bt' = r'at$ which contradicts the fact that w is an inverse Lyndon word.

For every $k \geq 1$, w is a suffix of $\ell^k w$ that is an inverse Lyndon word. Let x be a proper nonempty suffix of ℓ. Of course $x \ll \ell$. The word xw is not an inverse Lyndon word, otherwise we would have $\ell \ll w \preceq xw \ll \ell w$, a contradiction. Moreover, by Lemma 2.2, for any j, $1 \leq j < k$, we have $x\ell^j w \ll \ell^j w$ and $x\ell^j w$ is not an inverse Lyndon word. Finally, by 1., $\ell^k w$ is not an inverse Lyndon word. ◀

Algorithm 1 Compute ICFL(w), the unique compact factorization of w having the border property.

```
1:  function FACTORIZE($w$)
2:      $(\ell_1^{e_1}, \ldots, \ell_n^{e_n}) \leftarrow$ COMPACTFACTORS($w$)        ▷ Compute compact factors of $w$
3:      $\mathcal{F} \leftarrow \varnothing$
4:      $m' \leftarrow \ell_n^{e_n}$
5:      for $t = n - 1$ downto 1 do                       ▷ Work one compact factor at a time
6:          if $\ell_t \ll m'$ then                                    ▷ Proposition 7.9
7:              $\mathcal{F} \leftarrow (m', \mathcal{F})$
8:              $m' \leftarrow \ell_t^{e_t}$
9:          else
10:             $m' \leftarrow \ell_t^{e_t} \cdot m'$
11:     $\mathcal{F} \leftarrow (m', \mathcal{F})$
12:     return $\mathcal{F}$
```

We now describe Algorithm 1. Function FACTORIZE(w) will compute the unique compact factorization of w having the border property. First, at line 2, the decomposition of w into its compact factors is computed. Then, the factorization of w is carried out from right to left. Specifically, in accordance with Proposition 7.7, the for-loop at lines 5–10 will search for the longest suffix m' of w that is an inverse Lyndon word. The update of m' is managed by iteratively applying Proposition 7.9 at line 6. Once such longest suffix is found (that is, when the condition at line 6 is true) it is added to the growing factorization $\mathcal{F}$ and a new search for the longest suffix for the remaining portion of the string is initiated. Otherwise, line 10, the suffix is extended. In the end, the complete factorization is returned.

▶ **Example 7.10.** Let $\Sigma = \{a, b, c, d\}$, $a < b < c < d$, and let us run $\textsc{Factorize}(w)$ on $w = dabadabdabdadac$. First, at line 2, we get the sequence $(\ell_1, \ell_2^2, \ell_3) = (daba, (dab)^2, dadac)$. Then, at lines 3–4 we set $\mathcal{F} = \emptyset$ and $m' = \ell_3 = dadac$. We begin the for-loop at lines 5–10 in which i is set to 2 and 1, in turn. With $i = 2$ the test of line 6 succeeds, since $\ell_2 = dab \ll dadac = m'$, and so we set $\mathcal{F} = (dadac)$ and $m' = \ell_2^2 = (dab)^2$. At the second iteration, with $i = 1$, the test of line 6 again succeeds, since $\ell_1 = daba \ll (dab)^2 = m'$, thus we set $\mathcal{F} = ((dab)^2, dadac)$ and $m' = \ell_1 = daba$. We now fall out of the loop to line 11 where we set $\mathcal{F} = (daba, (dab)^2, dadac) = \mathrm{ICFL}(w)$.

7.1 Correctness and complexity

We now prove that Algorithm 1 is correct, that is that it will compute the unique inverse Lyndon factorization of w having the border property, namely $\mathrm{ICFL}(w)$. Formally:

▶ **Lemma 7.11.** *Let $w \in \Sigma^+$, and let $\mathcal{F}$ be the result of $\textsc{Factorize}(w)$. Then, $\mathcal{F} = \mathrm{ICFL}(w)$.*

Proof. Let $(\ell_1^{e_1}, \ldots, \ell_n^{e_n})$ be the decomposition of w into its compact factors, and let $L_t = \ell_t^{e_t} \cdots \ell_n^{e_n}$. We will denote by m'_t (resp. $\mathcal{F}_t$) the value of m' (resp. $\mathcal{F}$) at the end of iteration t. We will prove the following loop invariant: at the end of iteration t, sequence $(m'_t, \mathcal{F}_t)$ is a compact factorization of L_t having the border property. The claimed result will follow by Corollary 7.2.

Initialization. Prior to entering the loop, $(m'_n, \mathcal{F}_n) = (\ell_n^{e_n})$, where the last equality follows from Proposition 7.7.

Maintenance. Let $t \leq n - 1$. By the induction hypothesis, $\mathrm{ICFL}(L_{t+1}) = (m'_{t+1}, \mathcal{F}_{t+1})$. Suppose $\ell_t \ll m'_{t+1}$. Then, by 1. of Proposition 7.9, $\ell_t \cdot m'_{t+1}$ is not inverse Lyndon and m'_{t+1} is the longest suffix of $\ell_t^{e_t} \cdot m'_{t+1}$ that is an inverse Lyndon word. Thus, by Proposition 7.7 m'_{t+1} is the last factor of any compact factorization of $\ell_t^{e_t} \cdot m'_{t+1}$. Hence, $(m'_t, \mathcal{F}_t) = (\ell_t^{e_t}, m'_{t+1}, \mathcal{F}_{t+1})$ is a compact factorization of L_t having the border property. Now, consider the case where $\ell_t \not\ll m'_{t+1}$. Then, by the contrapositive of 2. of Proposition 7.9, $\ell_t \cdot m'_{t+1}$ is inverse Lyndon and thus, again by 2. of Proposition 7.9, $\ell_t^{e_t} \cdot m'_{t+1}$ is inverse Lyndon. Therefore, $(m'_t, \mathcal{F}_t) = (\ell_t^{e_t} \cdot m'_{t+1}, \mathcal{F}_{t+1})$ is a compact factorization having the border property.

Termination. After iteration $t = 1$, sequence $(m'_1, \mathcal{F}_1) = \mathrm{ICFL}(L_1) = \mathrm{ICFL}(w)$.

Finally, line 11 sets $\mathcal{F} = (m'_1, \mathcal{F}_1) = \mathrm{ICFL}(w)$. ◀

Function $\textsc{Factorize}(w)$ has time complexity that is linear in the length of w. Indeed, the sequence of compact factors obtained at line 2 can be computed in linear time in the length of w by a simple modification of Duval's algorithm (see [23]). After that, each iteration t of loop 5–10 can be implemented to run in time $\mathcal{O}(|\ell_t|)$. Indeed, condition $\ell_t \ll m'$ can be checked by naively comparing ℓ_t against m'. Furthermore, the update of m' and $\mathcal{F}$ can be done in constant time: in fact, ℓ_t, $\ell_t^{e_t}$, m' and $\mathcal{F}$ can all be implemented as pairs of indexes (in case of the former three) or as a list of indexes (in case of the latter) of w.

8 Conclusions

We discover the special connection between the Lyndon factorization under the inverse lexicographic ordering, named CFL_{in} and the canonical inverse Lyndon factorization, named ICFL: there exists a unique inverse Lyndon factorization having the border property and this unique factorization is ICFL. Moreover each inverse factor of ICFL is obtained by concatenating compact factors of CFL_{in}. These properties give a constrained structure to ICFL that deserve to be further explored to characterize properties of words. In particular, we believe the characterization of ICFL as a compact factorization, proved in the paper, could highlight novel properties related the compression of a word, as investigated in [20]. In particular, the number of compact factors seems to be a measure of repetitiveness of the word to be also used in speeding up suffix sorting of a word.

Finally, we believe that the characterization of ICFL in terms of CFL_{in} may be used to extend to ICFL the *conservation property* proved in [10] for CFL. This property shows that the Lyndon factorization of a word w preserves common factors with the factorization of a superstring of w. This extends the conservation of Lyndon factors explored for the product $u \cdot v$ of two words u and v [1, 20].

References

1 Alberto Apostolico and Maxime Crochemore. Fast parallel Lyndon factorization with applications. *Mathematical systems theory*, 28(2):89–108, 1995.

2 Hideo Bannai, Juha Kärkkäinen, Dominik Köppl, and Marcin Piatkowski. Constructing and indexing the bijective and extended Burrows-Wheeler transform. *Information and Computation*, 297:105153, 2024. doi:10.1016/j.ic.2024.105153.

3 Hideo Bannai, I Tomohiro, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. A new characterization of maximal repetitions by Lyndon trees. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 562–571, 2015.

4 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Encyclopedia of Mathematics and its Applications 129, Cambridge University Press, 2009.

5 Nico Bertram, Jonas Ellert, and Johannes Fischer. Lyndon words accelerate suffix sorting. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

6 Elena Biagi, Davide Cenzato, Zsuzsanna Lipták, and Giuseppe Romana. On the number of equal-letter runs of the Bijective Burrows-Wheeler Transform. In *CEUR Workshop Proceedings*, volume 3587, pages 129–142. R. Piskac c/o Redaktion Sun SITE, Informatik V, RWTH Aachen, 2023.

7 Paola Bonizzoni, Matteo Costantini, Clelia De Felice, Alessia Petescia, Yuri Pirola, Marco Previtali, Raffaella Rizzi, Jens Stoye, Rocco Zaccagnino, and Rosalba Zizza. Numeric Lyndon-based feature embedding of sequencing reads for machine learning approaches. *Inf. Sci.*, 607:458–476, 2022.

8 Paola Bonizzoni, Clelia De Felice, Rocco Zaccagnino, and Rosalba Zizza. Inverse Lyndon words and inverse Lyndon factorizations of words. *Adv. Appl. Math.*, 101:281–319, 2018.

9 Paola Bonizzoni, Clelia De Felice, Rocco Zaccagnino, and Rosalba Zizza. Lyndon words versus inverse Lyndon words: Queries on suffixes and bordered words. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications - 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings*, volume 12038 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 2020. doi:10.1007/978-3-030-40608-0_27.

10 Paola Bonizzoni, Clelia De Felice, Rocco Zaccagnino, and Rosalba Zizza. On the longest common prefix of suffixes in an inverse Lyndon factorization and other properties. *Theor. Comput. Sci.*, 862:24–41, 2021.

11 Paola Bonizzoni, Clelia De Felice, Rocco Zaccagnino, and Rosalba Zizza. From the Lyndon factorization to the Canonical Inverse Lyndon factorization: back and forth. under submission, ArXiv, 2024.

12 Kuo-Tsai Chen, Ralph H. Fox, and Roger C. Lyndon. Free Differential calculus, IV. The Quotient Groups of the Lower Central Series. *Ann. Math.*, 68:81–95, 1958.

13 Christian Choffrut and Juhani Karhumäki. Combinatorics of Words. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 1*, pages 329–438. Springer-Verlag, Berlin, Heidelberg, 1997.

14 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings.* Cambridge University Press, 2007.

15 Olivier Delgrange and Eric Rivals. Star: an algorithm to search for tandem approximate repeats. *Bioinformatics*, 20(16):2812–2820, 2004.

16 Jean-Pierre Duval. Factorizing Words over an Ordered Alphabet. *J. Algorithms*, 4(4):363–381, 1983.

17 Harold Fredricksen and James Maiorana. Necklaces of beads in k colors and k-ary de Brujin sequences. *Discrete Math.*, 23(3):207–210, 1978.

18 Daniele A. Gewurz and Francesca Merola. Numeration and enumeration. *Eur. J. Comb.*, 33(7):1547–1556, 2012.

19 Sukhpal Singh Ghuman, Emanuele Giaquinta, and Jorma Tarhio. Alternative Algorithms for Lyndon Factorization. In *Proceedings of the Prague Stringology Conference 2014, Prague, Czech Republic, September 1-3, 2014*, pages 169–178, 2014.

20 Tomohiro I, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Lyndon factorization algorithms for SLP and LZ78 compressed text. *Theoretical Computer Science*, 656:215–224, 2016.

21 Dominik Köppl, Daiki Hashimoto, Diptarama Hendrian, and Ayumi Shinohara. In-place bijective Burrows-Wheeler Transforms. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.21.

22 M. Lothaire. *Algebraic Combinatorics on Words, Encyclopedia Math. Appl*, volume 90. Cambridge University Press, 1997.

23 M. Lothaire. *Applied Combinatorics on Words.* Cambridge University Press, 2005.

24 Roger Lyndon. On Burnside's problem. *Trans. Amer. Math. Soc.*, 77:202–215, 1954.

25 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Suffix array and Lyndon factorization of a text. *J. Discrete Algorithms*, 28:2–8, 2014.

26 Igor Martayan, Bastien Cazaux, Antoine Limasset, and Camille Marchet. Conway-Bromage-Lyndon (CBL): an exact, dynamic representation of k-mer sets. *bioRxiv*, 2024. doi:10.1101/2024.01.29.577700.

27 Christophe Reutenauer. Free Lie algebras. In *Handbook of Algebra, London Mathematical Society Monographs.* Oxford Science Publications, 1993.

Symmetric-Difference (Degeneracy) and Signed Tree Models

Édouard Bonnet
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Julien Duron
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

John Sylvester
Department of Computer Science, University of Liverpool, UK

Viktor Zamaraev
Department of Computer Science, University of Liverpool, UK

Abstract

We introduce a dense counterpart of graph degeneracy, which extends the recently-proposed invariant symmetric difference. We say that a graph has *sd-degeneracy* (for symmetric-difference degeneracy) at most d if it admits an elimination order of its vertices where a vertex u can be removed whenever it has a d-twin, i.e., another vertex v such that at most d vertices outside $\{u, v\}$ are neighbors of exactly one of u, v. The family of graph classes of bounded sd-degeneracy is a superset of that of graph classes of bounded degeneracy or of bounded flip-width, and more generally, of bounded symmetric difference. Unlike most graph parameters, sd-degeneracy is not hereditary: it may be strictly smaller on a graph than on some of its induced subgraphs. In particular, every n-vertex graph is an induced subgraph of some $O(n^2)$-vertex graph of sd-degeneracy 1. In spite of this and the breadth of classes of bounded sd-degeneracy, we devise $\tilde{O}(\sqrt{n})$-bit adjacency labeling schemes for them, which are optimal up to the hidden polylogarithmic factor. This is attained on some even more general classes, consisting of graphs G whose vertices bijectively map to the leaves of a tree T, where transversal edges and anti-edges added to T define the edge set of G. We call such graph representations *signed tree models* as they extend the so-called tree models (or twin-decompositions) developed in the context of twin-width, by adding transversal anti-edges.

While computing the degeneracy of a graph takes linear time, we show that determining its symmetric difference is para-co-NP-complete. This may seem surprising as symmetric difference can serve as a short-sighted first approximation of twin-width, whose computation is para-NP-complete. Indeed, we show that deciding if the symmetric difference of an input graph is at most 8 is co-NP-complete. We also show that deciding if the sd-degeneracy is at most 6 is NP-complete, contrasting with the symmetric difference.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases symmetric difference, degeneracy, adjacency labeling schemes, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.32

Related Version *Full Version*: `https://arxiv.org/abs/2405.09011` [23]

1 Introduction

There are two theories of sparse graphs: the so-called Sparsity Theory pioneered by Nešetřil and Ossona de Mendez [18], and the theory behind the equivalent notions of bounded degeneracy, maximum average degree, subgraph density, and arboricity. One of the many merits of the former theory is to capture efficient first-order model checking within subgraph-closed classes, with the so-called *nowhere dense* classes [12]. *Monadic stability* constitutes a dense analogue of nowhere denseness with similar algorithmic properties [10]. The second

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 32; pp. 32:1–32:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

theory has, as we will see, simple but useful connections with the chromatic number and adjacency labeling schemes. One of our two main motivations is to introduce and explore dense analogues of it.

The degeneracy of a graph G is the minimum integer d such that every induced subgraph of G has a vertex of degree at most d. As removing vertices may only decrease the degree of the remaining ones, checking that the degeneracy is at most d can be done greedily. This prompts the following equivalent definition, equal to the *coloring number*[1] minus one [11, 9]. A graph has degeneracy at most d if there is a total order, called *degeneracy ordering*, on its vertices such that every vertex v has at most d neighbors following v in the order. The degeneracy is then the least integer d such that an ordering witnessing degeneracy at most d exists. Given such an ordering, a graph can be properly $(d+1)$-colored by a greedy strategy: use the smallest available color looping through vertices in the given order. Another advantage of the definition via degeneracy ordering is that it yields a polynomial-time algorithm to compute the degeneracy. While the graph is nonempty, find a vertex of minimum degree, append it to the order, and remove it from the graph. Degeneracy is frequently used to bound the chromatic number from above. For instance, until recently [19] the Kostochka–Thomason degeneracy bound of graphs without K_t minor [15, 20] was the best way we knew of coloring these graphs.

Another application of bounding the degeneracy is to obtain implicit representations. Indeed graphs of bounded degeneracy admit $f(n)$-bit *adjacency labeling schemes* with $f(n) = O(\log n)$.[2] In other words, given a class of graphs of degeneracy at most d, there exists an algorithm, called *decoder*, such that the vertices of any n-vertex graph G from the class can be assigned *labels* (which are binary strings) of length $f(n)$ in such a way that the decoder can infer the adjacency of any two vertices u, v in G from their mere labels. An $O(\log n)$-bit labeling scheme is easy to design for any class $\mathcal{C}$ of bounded degeneracy. From an ordering of $G \in \mathcal{C}$ witnessing degeneracy d, the label of each vertex stores its own index in the ordering and the indices of its at most d neighbors that follow it in the order. Then the decoder just checks whether the index of one of u, v is among the indices of the neighbors of the other vertex. Note that each label has size at most $(d+1)\lceil \log n \rceil$. For example, this was recently used to show that every subgraph-closed class with single-exponential speed of growth admits such a labeling scheme [3].

Adjacency labeling schemes of size $O(\log n)$ are at the heart of the recently-refuted Implicit Graph Conjecture (IGC) [14, 17]. The IGC speculated that the information-theoretic necessary condition for a hereditary graph class to have an $O(\log n)$-bit labeling scheme is also sufficient. This necessary condition comes from the observation that a string of length $O(n \log n)$ obtained by concatenating all vertex labels is an encoding of the graph. Therefore a class of graphs that admits an adjacency labeling scheme of size $O(\log n)$ contains at most $2^{O(n \log n)}$ (un)labeled n-vertex graphs. Graph classes with such a bound on the number of (un)labeled n-vertex graphs are called *factorial*. In this terminology, the IGC can be stated as follows: any hereditary factorial graph class admits an $O(\log n)$-bit adjacency labeling scheme.

The IGC has been refuted by a wide margin; in a breakthrough work, Hatami and Hatami [13] showed that there are factorial hereditary graph classes for which any adjacency labeling scheme requires labels of length $\Omega(\sqrt{n})$. However, the refutation is based on a counting argument and does not pinpoint an explicit counterexample. There are a number

[1] not to be confused with the *chromatic* number

[2] Throughout the paper, log denotes the logarithm function in base 2.

of explicit factorial graph classes that could refute the IGC, but the conjecture is still open for these classes. Let us call EIGC (for Explicit Implicit Graph Conjecture) this very challenge. For instance, whether the IGC holds within intersection graphs of segments, unit disks, or disks in the plane, and more generally semi-algebraic graph classes, is unsettled. Despite the workable definitions of these classes, the geometric representations alone cannot lead to $O(\log n)$-bit labeling schemes [16]. If such labeling schemes exist, they are likely to utilize some non-trivial structural properties of these graphs.

The graph parameter *symmetric difference* was introduced to design a candidate to explicitly refute the IGC [2]. A graph G has symmetric difference at most d if in every induced subgraph of G there is a pair of vertices u, v such that there are at most d vertices different from u and v that are adjacent to exactly one of u, v. In other words, u and v are *d-twins*, i.e., they become twins after removing at most d vertices from the graph. One can construe symmetric difference as a dense analogue of the first definition of degeneracy given above. Symmetric difference is a hereditary graph parameter: it can only decrease when taking induced subgraphs. Like classes of bounded degeneracy, classes of bounded symmetric difference are factorial [2]. Symmetric difference generalizes degeneracy in the sense that any class of graphs of bounded degeneracy has bounded symmetric difference. Indeed, if a graph has degeneracy at most d, then it has symmetric difference as most $2d$: for any graph with an ordering witnessing degeneracy d, the first two vertices in the order are $2d$-twins. Notice that, on the other hand, complete graphs have unbounded degeneracy, but their symmetric difference is 0. Classes of bounded symmetric difference contains classes of bounded twin-width, and this containment is strict as twin-width is unbounded on degenerate graphs [5]. The existence of an $O(\log n)$-bit adjacency labeling scheme for graphs of bounded symmetric difference remains open.

Our contribution. We introduce another dense analogue of degeneracy based on the second given definition. The *sd-degeneracy* (for *symmetric-difference degeneracy*) of a graph G is the least integer d for which there is an ordering of the vertices of G such that every vertex v but the last one admits a d-twin in the subgraph of G induced by v and all the vertices following it in the order. It follows from the definitions that graphs with sd-degeneracy at most d form a superset of graphs with symmetric difference at most d. Contrary to what happens in the sparse setting with degeneracy, this superset is strict. In fact, there are classes with sd-degeneracy 1 and unbounded symmetric difference.

▶ **Proposition 1** ($\star$). *For any n-vertex graph G, there exists a graph of sd-degeneracy 1 with less than n^2 vertices containing G as an induced subgraph.*

By an aforementioned counting argument, the class of all graphs requires labeling schemes of size $\Theta(n)$. Therefore, by Proposition 1, the (non-hereditary) class of graphs with sd-degeneracy at most 1 requires adjacency labels of size $\Omega(\sqrt{n})$. Surprisingly, we match this lower bound with a labeling scheme, tight up to a polylogarithmic factor, for any class of bounded sd-degeneracy.

▶ **Theorem 2.** *The class of all graphs with sd-degeneracy at most d admits an $O(\sqrt{dn}\log^3 n)$-bit adjacency labeling scheme.*

The tool behind the proof of Theorem 2 is the second motivation of the paper. We wish to unify and extend twin-decompositions of low width (also called tree models) [4, 8] developed in the context of twin-width, and spanning paths (or Welzl orders) of low crossing number (or low alternation number) [22], which are useful orders in answering geometric range queries; also see [10] which utilizes these orders as part of efficient first-order model checking algorithms. We thus introduce *signed tree models*. A signed tree model of a graph G is a tree whose leaves are in one-to-one correspondence with the vertices of G, together with extra transversal edges and anti-edges, which fully determine (see the exact rules in Section 3) the edges of G. The novelty compared to the existing tree models is the presence of transversal anti-edges. We show that graphs with signed tree models of degeneracy at most d admit a labeling scheme as in Theorem 2. The latter theorem is then obtained by building such a signed tree model for any graph of sd-degeneracy d.

When given the vertex ordering witnessing sd-degeneracy d, the labeling scheme can be effectively computed. However, we show that computing the sd-degeneracy of a graph (hence, in particular a witnessing order) is NP-complete, even when the sd-degeneracy is guaranteed to be below a fixed constant. In the language of parameterized complexity, sd-degeneracy is para-NP-complete.

▶ **Theorem 3.** *Deciding if a graph has sd-degeneracy at most 1 is NP-complete.*

We show that, surprisingly, the other dense analogue of degeneracy, symmetric difference, is co-NP-complete. Again, the associate parameterized problem is para-co-NP-complete.

▶ **Theorem 4.** *Deciding if a graph has symmetric difference at most 8 is co-NP-complete.*

This is curious because sd-degeneracy and symmetric difference similarly extend to the dense world two equivalent definitions of degeneracy. Nevertheless, one can explain the apparent tension between Theorems 3 and 4: a vertex ordering witnesses an upper bound in the sd-degeneracy, whereas an induced subgraph witnesses a lower bound in the symmetric difference. We leave as an open question whether classes of bounded symmetric difference have labeling schemes of (poly)logarithmic size. This is excluded for bounded sd-degeneracy, for which we now know essentially optimal labeling schemes. While not an absolute barrier, the likely absence of polynomial certificates tightly upper bounding the symmetric difference complicates matters in settling this open question.

Organization. Section 2 gives definitions and notation. In Section 3 we introduce signed tree models, and prove that graphs of bounded sd-degeneracy admit signed tree models of bounded width. In Section 4 we show how to balance these signed tree models, and complete the proof of Theorem 2. In Section 5 we prove Theorem 4, and we prove Theorem 3 in the long version [23]. The proofs marked with a ⋆ have been moved to the appendix.

2 Preliminaries

We denote by $[i, j]$ the set of integers that are at least i and at most j, and $[i]$ is a shorthand for $[1, i]$. We follow standard asymptotic notation throughout, and additionally by $f(n) = \tilde{O}(g(n))$ we mean that there exists constants $c, n_0 > 0$ such that for any $n \geqslant n_0$ we have $f(n) \leqslant g(n) \log^c n$.

We denote by $V(G)$ and $E(G)$ the vertex set and edge set of a graph G, respectively. Given a vertex u of a graph G, we denote by $N_G(u)$ the set of neighbors of u in G (*open neighborhood*) and by $N_G[u]$ the set $N_G(u) \cup \{u\}$ (*closed neighborhood*). When H, G are two graphs, we may denote by $H \subseteq_i G$ (resp. $H \subseteq G$) the fact that H is an induced subgraph (resp. subgraph) of G, i.e., can be obtained by removing vertices of G (resp. by removing vertices and edges of G). We denote by $G[S]$ the subgraph of G induced by S, formed by removing every vertex of $V(G) \setminus S$. We use $G - S$ as a shorthand for $G[V(G) \setminus S]$, and $G - v$, for $G - \{v\}$.

Given two sets A and B, we denote by $A \triangle B$ their symmetric difference, that is, $(A \setminus B) \cup (B \setminus A)$. Given a graph G, and two distinct vertices $u, v \in V(G)$, we set

$$\mathrm{sd}_G(u,v) := |(N_G(u) \setminus \{v\}) \triangle (N_G(v) \setminus \{u\})|.$$

The *symmetric difference* of G, $\mathrm{sd}(G)$, is defined as $\max_{H \subseteq_i G} \min_{u \neq v \in V(H)} \mathrm{sd}_H(u,v)$. Symmetric difference was implicitly introduced in [2] and later explicitly defined in [1]. For example, if G is a planar graph, one can find in any induced subgraph H of G two vertices of degree less than 6. Hence planar graphs have symmetric difference bounded by 12. We call *sd-degeneracy* of G, denoted by $\mathrm{sdd}(G)$, the smallest non-negative integer d such that $|V(G)| = 1$ or there is a pair $u \neq v \in V(G)$ satisfying $\mathrm{sd}_G(u,v) \leqslant d$ and $G - v$ has sd-degeneracy at most d. We say that an ordering $v_1, v_2, \ldots, v_n$ of the vertices of G witnesses that the *sd-degeneracy* of G is at most d if for every $i \in [n-1]$, there is a $j > i$ such that $\mathrm{sd}_{G - \{v_k \;:\; k \in [i-1]\}}(v_i, v_j) \leqslant d$. It thus holds that for any graph G, $\mathrm{sdd}(G) \leqslant \mathrm{sd}(G)$, since for every $i \in [n]$, $G - \{v_k \;:\; k \in [i-1]\}$ is an induced subgraph of G. But, as shown by Proposition 1, there are some graphs with sd-degeneracy 1 and unbounded symmetric difference.

Two vertices u, v are said to be *d-twins* in a graph G if they are distinct and $|(N_G(u) \setminus N_G[v]) \cup (N_G(u) \setminus N_G[v])| \leqslant d$. The *$a \times b$ rook graph* has vertex set $\{(i,j) \;:\; i \in [a], j \in [b]\}$ and edge set $\{(i,j)(k,\ell) \;:\; (i,j) \neq (k,\ell),\ i = k \text{ or } j = \ell\}$. Equivalently it is the line graph of the bipartite complete graph $K_{a,b}$. For every $a, b \geqslant 3$, the symmetric difference of the $a \times b$ *rook graph* is $2(\min(a,b) - 1)$.

We will extensively use *tree orders*, i.e., partial orders defined by ancestor–descendant relationships in a rooted tree. We denote by $\prec_T$ the corresponding relation in rooted tree T. That is, $u \prec_T u'$ means that u is a *strict ancestor* of u' in T, and $u \preceq_T u'$ means that u is an *ancestor* of u', i.e., $u = u'$ or $u \prec_T u'$. We extend this partial order to elements of $\binom{V(T)}{2}$. An unordered pair uv is an *ancestor* of $u'v'$ in T, denoted by $uv \preceq_T u'v'$, whenever either $u \preceq_T u'$ and $v \preceq_T v'$, or $v \preceq_T u'$ and $u \preceq_T v'$ holds. We write $uv \prec_T u'v'$ when $uv \preceq_T u'v'$ and $\{u,v\} \neq \{u',v'\}$. A rooted binary tree is *full* if all its *internal nodes*, i.e., non-leaf nodes, have exactly two children. A rooted binary tree is *complete* if all its levels are completely filled, except possibly the last one, wherein leaves are left-aligned. The *depth* of a rooted tree is the maximum number of nodes in a root-to-leaf path. We denote by $L(T)$ the set of leafs of T.

3 Signed tree models

An unordered pair of vertices in T that is not in an ancestor–descendant relationship is called a *transversal pair of* T. Two transversal pairs $uv, u'v'$ of T *cross* if u, v have the same common ancestor as u', v' do, and neither uv is an ancestor of $u'v'$, nor $u'v'$ is an ancestor of uv. A *signed tree model* $\mathcal{T}$ is a triple $(T, A(T), B(T))$, where T is a full binary tree, $A(T)$ (for Android green, or Anti) is a set of transversal pairs of T, called *transversal anti-edges*, and $B(T)$ (for Blue, or Biclique) is a set of transversal pairs of T, called *transversal edges*, such that $A(T) \cap B(T) \neq \emptyset$ and no $uv, u'v' \in A(T) \cup B(T)$ cross. We may refer to the transversal anti-edges as *green edges*, and to the transversal edges as *blue edges*.

The *width* of the signed tree model $(T, A(T), B(T))$ is the degeneracy of the graph $(V(T), A(T) \cup B(T))$. Note that if $(V(T), A(T) \cup B(T))$ is d-degenerate, then $(V(T), A(T) \cup B(T) \cup E(T))$ is $(d+3)$-degenerate. The signed tree model is d-sparse if $|A(T) \cup B(T)| \leqslant d|V(T)|$. We observe that a signed tree model of width d is d-sparse, but an $O(1)$-sparse signed tree model can have width $\Omega(\sqrt{|V(T)|})$ (think of the disjoint union of a clique on $\sqrt{n}$ vertices with a set $n - \sqrt{n}$ independent vertices).

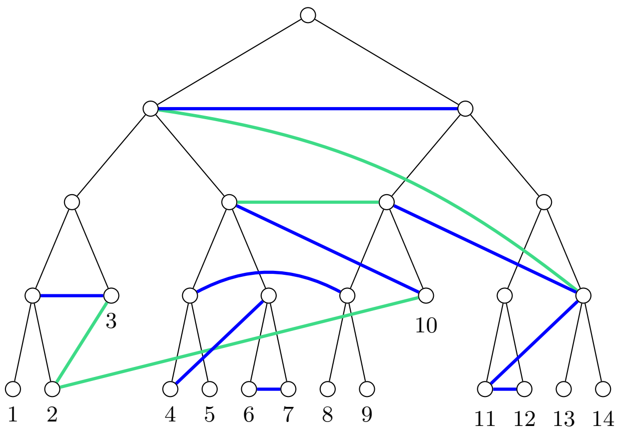


Figure 1 A signed tree model of a 14-vertex graph.

The signed tree model $\mathcal{T} := (T, A(T), B(T))$ defines a graph $G := G_{\mathcal{T}}$ with vertex set $L(T)$. Two leaves $u, v \in L(T)$ are adjacent in G if there is $u'v' \in B(T)$ such that $u' \preceq_T u$ and $v' \preceq_T v$, and there is no $u''v'' \in A(T)$ with $u'v' \prec_T u''v'' \preceq_T uv$. For example, in the representation of Figure 1, vertices 4 and 8 are adjacent in G because of the blue edge between their parents (below the green edge between their grandparents), but vertices 7 and 8 are non-adjacent because of the green edge between their grandparents (below the blue edge between their great-grand-parents). We may say that a graph G admits (or has) a signed tree model of width d if there is a signed tree model of this width that defines G. Every graph G admits a signed tree model as one can simply set $A(T) := \emptyset$, $B(T) := E(G)$ on an arbitrary full binary tree T with $L(T) = V(G)$. However this representation may have large width, while a more subtle one (linking nodes higher up in the tree) may have a lower width.

A signed tree model is said to be *clean* if every pair of siblings are linked by a green or blue edge. It is easy to turn a signed tree model into a clean one representing the same graph: simply add green edges between every pair of siblings that were previously not linked (by a blue or green edge). This operation may only increase the width of the signed tree model by 1. The advantage of working with a clean signed tree model is that for every pair of leaves u, v with least common ancestor w, there is at least one transversal edge or anti-edge connecting the paths (in T) between w and u and between w and v. Clean tree models will be useful in Section 4 when we balance the trees associated with the tree models.

Given a clean signed tree model $(T, A(T), B(T))$ and $u, v \in L(T)$, we denote by $e_T(u, v)$ the unique green or blue edge $u'v'$ such that $u'v' \preceq_T uv$ and no green or blue edge $u''v''$ satisfies $u'v' \prec_T u''v'' \preceq_T uv$. The edge $e_T(u, v)$ exists because the signed tree model is clean, and is unique because no green or blue edges may cross (or be equal). Then, u, v are adjacent in G if and only if $e_T(u, v) \in B(T)$, i.e., $e_T(u, v)$ is a blue edge. We first show that graphs of bounded sd-degeneracy (and in particular, of bounded symmetric difference) admit clean signed tree models of bounded width.

▶ **Lemma 5.** *Any graph of sd-degeneracy d admits a clean signed tree model of width $d + 1$.*

Proof. Let $v_1, \ldots, v_n$ be a vertex ordering that witnesses sd-degeneracy d for an n-vertex graph G. For $i \in [n]$, let $G_i := G - \{v_j \ : \ 1 \leqslant j \leqslant i - 1\}$. In particular, $G_1 = G$. Let u_i be a d-twin of v_i in G_i. Initially we consider a forest of n distinct 1-vertex rooted trees, each root labeled by a distinct vertex of G. We will build T (and in parallel, the transversal anti-edges and edges) by iteratively giving a common parent to two roots of this forest of n singletons. Note that different nodes of T may have the same label, as the labels will range in $V(G)$ whereas T has $2n-1$ nodes.

For i ranging from 1 to $n-1$:

- add a blue (resp. green) edge between v_i and u_i if $u_iv_i \in E(G)$ (resp. $u_iv_i \notin E(G)$),
- add a blue edge between v_i and the roots labeled by w for $w \in N_{G_i}(v_i) \setminus N_{G_i}[u_i]$,
- add a green edge between v_i and the roots labeled by w for $w \in N_{G_i}(u_i) \setminus N_{G_i}[v_i]$, and
- create a common parent, labeled by u_i, for the roots labeled u_i (left child) and v_i (right child).

This defines a full binary tree T such that $L(T) = V(G)$. In $(V(T), A(T) \cup B(T))$, the leaves labeled by v_1 and u_1 have degree at most $d+1$ and 1, respectively. Hence an immediate induction on $(T, A(T), B(T))$ (after removing these two leaves, and following the order $v_2, \dots, v_n$) shows that $(V(T), A(T) \cup B(T))$ is $(d+1)$-degenerate. As we only add transversal anti-edges and edges between pairs of roots, no pair in $A(T) \cup B(T)$ can cross. Indeed if x, y are two nodes of T that are both roots in some G_i, then it cannot happen that x', y' are also both roots of some $G_{i'}$ with $x \prec_T x'$ and $y' \prec_T y$. The first item further ensures that the signed tree model $(T, A(T), B(T))$ of width $d+1$ is clean.

Let us finally check that for every $u, v \in L(T)$, $e_T(u, v)$ is a blue edge if and only if $uv \in E(G)$. This is a consequence of the following property.

▷ Claim 6. Let x, y be two nodes of T labeled by u, v respectively. Let x' be a child of x, labeled by u', such that $x'y$ is neither a blue nor a green edge. Further assume that y was a root when the parent of x' (i.e., x) was created. Then, $uv \in E(G)$ if and only if $u'v \in E(G)$.

Proof. If x' is the left child of x, the conclusion holds since $u = u'$. We can thus assume that x' is the right child of x, and not the sibling of y since it would contradict that $x'y$ is neither a blue nor a green edge. Node x' was not linked to y by a blue or a green edge, so v cannot be a neighbor of exactly one of u, u'. ◁

Consider the moment $e_T(u, v)$ was added to the signed tree model, say between the then-roots x and y, labeled by u' and v', respectively. By the way blue and green edges are introduced, xy is a blue edge if $u'v' \in E(G)$, and xy is green if $u'v' \notin E(G)$. Thus we conclude by iteratively applying Claim 6. ◀

4 Balancing Signed Tree Models

For any signed tree model of width d of an n-vertex graph, we get an adjacency labeling scheme with labels of size $O(dh \log n)$, where h is the depth of T. Indeed, one can label a leaf v of T (i.e., vertex of G) by the identifiers (each of $\log(2n)$ bits) of all the nodes of the path from v to the root of T, followed by the identifiers of the outneighbors of these at most h nodes in a fixed orientation of $(V(T), A(T) \cup B(T))$ with maximum outdegree at most $d+1$, allocating an extra bit for the color of each corresponding edge. One can then decode the adjacency of any pair $u, v \in V(G)$ by looking at the color of $e_T(u, v)$. The latter is easy to single out, based on the labels of u and v.

▶ **Proposition 7.** *Let G be an n-vertex graph with a signed tree model of width d and depth h. Then, G admits an $O(dh \log n)$-bit adjacency labeling scheme.*

Unfortunately, the depth of the tree T of a signed tree model of low width obtained for an n-vertex graph of low sd-degeneracy could be as large as n. This makes a direct use of Proposition 7 inadequate. Instead, we first decrease the depth of the signed tree model, while controlling its sparsity. We rely on the following simple lemma.

▶ **Lemma 8.** *Let T be a full, complete tree, whose leaves read $1, \ldots, n \geqslant 2$ from left to right. Any interval $[i,j]$ with $i,j \in [n]$ is the disjoint union of the leaves of at most $2 \log n$ rooted subtrees of T.*

Proof. Let $X \subseteq V(T)$ be such that the leaves of the subtrees rooted at a node of X partition $[i,j]$, and X is of minimum cardinality among node subsets with this property. Let k be the first level of T intersected by X (with the root being at level 1). At most two nodes x, y of X are at level k (and exactly one node when $k = 2$), with $x = y$ or x to the left of y. Observe that if $x \neq y$, then x, y have to be consecutive along the left-to-right ordering of level k, but cannot be siblings (otherwise they can be substituted by their parent). At level $k+1$, at most two nodes can be part of X: the node just to the left of the leftmost child of x, and the node just to the right of the rightmost child of y. This property propagates to the last level. Thus $|X| \leqslant \max(2(\lceil \log n \rceil - 1), \lceil \log n \rceil) \leqslant 2 \log n$. Note indeed that there are $\lceil \log n \rceil + 1$ levels. ◀

From the previous proof it can also be seen that there is a unique minimum-cardinality set X representing $[i,j]$. In what follows, let us denote it by $X_{i,j}$. We also denote by $I_T(x)$ the set of leaves of the subtree of T rooted at $x \in V(T)$.

▶ **Observation 9.** *For every rooted tree T, and every $x, y \in V(T)$, if $I_T(x)$ and $I_T(y)$ intersect, then one is included in the other.*

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

▶ **Lemma 10.** *Let $(T, A(T), B(T))$ be a clean d-sparse signed tree model of an n-vertex graph G. Then, G admits a $4d \log^2 n$-sparse signed tree model $(T', A(T'), B(T'))$ of depth $\lceil \log n \rceil + 1$.*

Proof. Consider the left-to-right order on $L(T)$. To ease the notation, say that the leaves are labeled $1, 2, \ldots, n$ in this order. We choose for T' the full, complete binary tree whose leaves are also labeled by $1, 2, \ldots, n$ when read from left to right. For every transversal anti-edge (resp. edge) $xy \in A(T)$ (resp. $xy \in B(T)$), note that $I_T(x)$ and $I_T(y)$ are discrete intervals. Let $[i,j] := I_T(x)$ and $[i',j'] := I_T(y)$. We add to $A(T')$ (resp. $B(T')$) all the unordered pairs $x'y'$ with $x' \in X_{i,j}$ and $y' \in X_{i',j'}$. It may happen that some $x'y'$ is added both to $A(T')$ and $B(T')$. In which case, $x'y'$ originates from both $x_0y_0 \in A(T)$ and $x_1y_1 \in B(T)$ such that $x_0y_0 \prec_T x_1y_1$ or $x_1y_1 \prec_T x_0y_0$. In the former case, we remove $x'y'$ from $B(T')$ (and only keep it in $A(T')$), and in the latter, we remove $x'y'$ from $A(T')$ (and only keep it in $B(T')$). This finishes the construction of $\mathcal{T}' := (T', A(T'), B(T'))$.

Let us first argue that no pairs of green or blue edges cross in T'. Assume for the sake of contradiction that $a'b', c'd' \in A(T') \cup B(T')$ satisfy $a' \prec_{T'} c'$ and $d' \prec_{T'} b'$. Let $ab, cd \in A(T) \cup B(T)$ be the green or blue edges that created $a'b', c'd'$, respectively. As $I_T(a) \supseteq I_{T'}(a')$, $I_T(c) \supseteq I_{T'}(c')$, and $a' \prec_{T'} c'$, $I_T(a)$ and $I_T(c)$ intersect. Thus by Observation 9, $I_T(a) \subseteq I_T(c)$ or $I_T(c) \subseteq I_T(a)$. By minimality of the sets $X_{i,j}$, $I_T(c)$ cannot include $I_{T'}(a')$. Thus $I_T(c) \subset I_T(a)$, so $a \prec_T c$. Analogously $d \prec_T b$, which implies that ab and cd cross in T. Therefore $\mathcal{T}'$ is a signed tree model.

By design, the depth of T' is $\lceil \log n \rceil + 1$. As $\mathcal{T} := (T, A(T), B(T))$ is d-sparse, it has at most $(2n-1)d$ transversal (anti-)edges. Each blue or green edge of $\mathcal{T}$ gives rise to at most $(2 \log n)^2$ blue or green edges of $\mathcal{T}'$, by Lemma 8. Hence $\mathcal{T}'$ is $4d \log^2 n$-sparse.

Let us finally check that $\mathcal{T}'$ still represents G. Fix $u, v \in V(G)$ and $xy := e_T(u,v)$. Let $x'y'$ be the green or blue edge of $\mathcal{T}'$ originating from xy such that $u \in I_{T'}(x'), v \in I_{T'}(y')$. We claim that $x'y'$ cannot have been removed (see the technicality at the end of the construction of $\mathcal{T}'$), nor can $x''y'' \in A(T') \cup B(T')$ hold with $x'y' \prec_T x''y''$. Indeed, by the arguments of the second paragraph, the green or blue edge e of $\mathcal{T}$ giving rise to $x''y''$ would be such that $xy \prec_T e \preceq_T uv$, contradicting the definition of $e_T(u,v)$. ◀

We finally need this folklore observation.

▶ **Observation 11.** *Every m-edge graph has degeneracy at most $\lceil \sqrt{2m} \rceil - 1$.*

Proof. It is enough to show that any m-edge graph G has a vertex of degree at most $\lceil \sqrt{2m} \rceil - 1$. If all the vertices of G have degree at least $\lceil \sqrt{2m} \rceil$, then $m \geqslant \frac{1}{2} n \lceil \sqrt{2m} \rceil$. But also $n \geqslant \lceil \sqrt{2m} \rceil + 1$ for a vertex to possibly have $\lceil \sqrt{2m} \rceil$ neighbors. Thus $m \geqslant \frac{1}{2} \sqrt{2m}(\sqrt{2m}+1) > m$, a contradiction. ◀

Combining Lemmas 5 and 10, Observation 11, , and Proposition 7 yields Theorem 2.

Proof of Theorem 2. Let G be an n-vertex graph of sd-degeneracy d. By Lemma 5, G admits a clean signed tree model of width at most $d+1$, hence $(d+1)$-sparse. Thus by Lemma 10, G has a $4(d+1)\log^2 n$-sparse signed tree model $\mathcal{T}$ of depth $\lceil \log n \rceil + 1$. By Observation 11, $\mathcal{T}$ has width at most

$$\sqrt{16(d+1)n \log^2 n} = 4\sqrt{(d+1)n} \log n.$$

Therefore, by Proposition 7, G has a $O(\sqrt{dn} \log^3 n)$-bit labeling scheme. ◀

5 Symmetric Difference is para-co-NP-complete

For any fixed even integer $d \geqslant 8$, we show that the following problem is NP-complete: Does the input graph G have an induced subgraph with at least two vertices and no pair of d-twins? We call such an induced subgraph a $(d+1)$-*diverse graph*. The membership of this problem to NP is straightforward, as a $(d+1)$-diverse induced subgraph H of G is a polynomial-sized witness. One can indeed check in polynomial-time that H has at least two vertices, and that for every pair u, v of vertices of H, at least $d+1$ other vertices of H are neighbors of exactly one of u, v.

The *d-twin graph* $T_d(G)$ of a graph G is a graph with vertex set $V(G)$ and edges between every pair of d-twins.

▶ **Observation 12.** *The vertices of a $(d+1)$-diverse induced subgraph of G form an independent set of $T_d(G)$.*

Given any 3-SAT formula φ with at most three occurrences of each variable, clauses of size two or three, and at least three clauses, we build a graph $G := G(\varphi)$ such that G has a $(d+1)$-diverse induced subgraph if and only if φ is satisfiable. Such a restriction of 3-SAT is known to be NP-complete [21].

5.1 Bubble gadget

A *bubble gadget* B (or *bubble* for short) is a $w \times w$ rook graph, with $w := \frac{d}{2} + 2$, deprived of the two rightmost vertices of its top row. We say that B is *properly attached* to the rest of the graph if each vertex of the top row (of width $\frac{d}{2}$) and of the rightmost column (of height $\frac{d}{2} + 1$) has one or two neighbors outside the gadget, whereas the other vertices of B have no neighbors outside $V(B)$. Let S be the set of neighbors of the bubble outside of B.

We say that B is *neatly attached to* S if it is properly attached to S, and further, vertices of the top row and rightmost column have *exactly* one outside neighbor, and at most one vertex of S has neighbors in both the top row and rightmost column. The *neat* attachments that we will use, in this section and the next, satisfy $2 \leqslant |S| \leqslant 5$. Hence they can be described by a tuple of size between 2 and 5, listing the number of neighbors of vertices in S among $V(B)$, starting with the top row and ending with the rightmost column. For instance, Figure 2 depicts a neat $(2,2,2,7)$-attachment. A bubble properly attached to S is in a delicate state. It may entirely survive in a $(d+1)$-diverse induced subgraph of G.

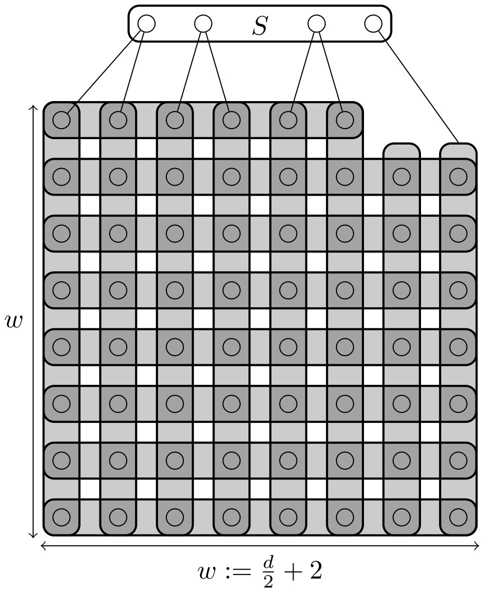


Figure 2 A neatly $(2, 2, 2, 7)$-attached bubble gadget, with $d = 12$.

▶ **Observation 13.** *Let B be a bubble gadget properly attached to S in G. No pair of vertices of B are d-twins in $G[V(B) \cup S]$.*

Proof. In B the only pairs with symmetric difference at most d, in fact exactly d, consist of a vertex in the top row and another vertex in its column, or two vertices of the same row in the two rightmost columns. In both cases, these pairs have symmetric difference at least $d+1$ in $G[V(B) \cup S]$ since vertices of the top row or rightmost column have at least one neighbor in S, while all other vertices of B have no neighbor in S. ◀

However, deletions that cause one vertex of the top row or two vertices of the rightmost column to no longer have outside neighbors cause the bubble to completely collapse.

▶ **Lemma 14** (⋆). *Let B be a bubble gadget properly attached to S in G. Let H be any $(d+1)$-diverse induced subgraph of G, such that at least one vertex of the top row or at least two vertices of the rightmost column has no neighbor in $V(H) \setminus V(B)$. Then, H contains at most one vertex of B.*

In the current section, for the hardness of symmetric difference, all the bubble gadgets will be neatly attached. Furthermore, every vertex a bubble is attached to will have at least one neighbor on the top row, or at least two neighbors in the rightmost column. Thus the deletion of *any* vertex a bubble B is attached to will result, by Lemma 14, in deleting all the vertices of B but at most one.

5.2 Variable and clause gadgets

The *variable gadget* of variable x used in φ is simply two vertices $x, \neg x$ adjacent to a set N_x of $t := \frac{d}{2} + 1$ shared neighbors. Since each literal appears positively and negatively in φ (otherwise the valuation of the literal is clear), vertices x and $\neg x$ have one or two other neighbors in G corresponding to the clause they belong to, as we will soon see. The vertices of N_x will have other neighbors split into at most four bubble gadgets. This too will be

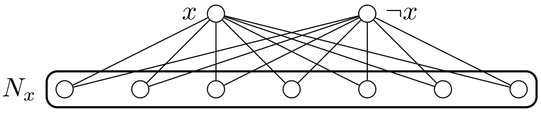


Figure 3 The variable gadget of x with $d = 12$.

described shortly. The *clause gadget* of clause c consists of a pair of adjacent vertices v_c, d_c. We make v_c (but not d_c) adjacent to the two or three vertices corresponding to the literals of c.

5.3 Construction of $G(\varphi)$

Unsurprisingly, we add one variable gadget per variable, and one clause gadget per clause of φ. Let $x_1, \dots, x_n$ be a numbering of the variables, and $c_1, \dots, c_m$, of the clauses. We neatly attach a bubble gadget to S_j made of the five vertices $z_j, v_{c_j}, d_{c_j}, v_{c_{j+1}}, d_{c_{j+1}}$ for every $j \in [m-1]$, with (in this order) a $(1, \lfloor d/4 \rfloor, \lfloor d/4 \rfloor, \lceil d/4 \rceil, \lceil d/4 \rceil)$-attachment, where z_j is a vertex of some N_x. The choice of z_j is irrelevant, but we take all the vertices z_j pairwise distinct. This is possible since there are at most $3n/2$ clauses, and more than $dn/2$ vertices contained in the union of the sets N_x. For every $j \in [m-1]$, we make $\{v_{c_j}, d_{c_j}\}$ fully adjacent to $\{v_{c_{j+1}}, d_{c_{j+1}}\}$. The construction of G is almost complete; see Figure 4 for an illustration.

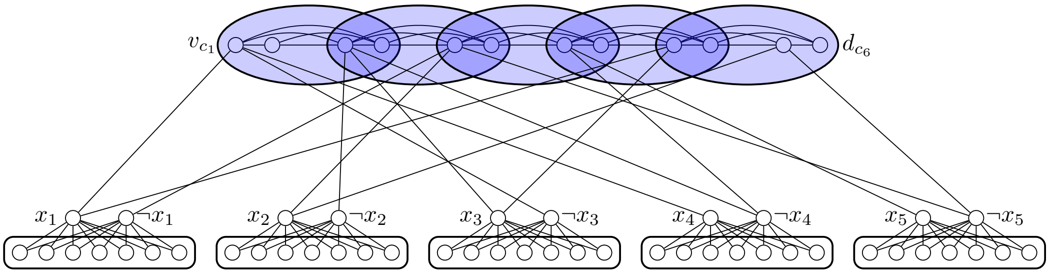


Figure 4 The essential part of G built so far, for a 3-CNF formula φ whose first two clauses are $x_1 \vee \neg x_3 \vee x_4$ and $\neg x_2 \vee x_3 \vee \neg x_4$. The blue ellipses represent the bubbles attached to the four enclosed vertices (recall that the bubble is attached to a fifth vertex among the sets N_x).

At this point, all the vertices v_{c_j}, d_{c_j} such that $j \in [2, m-1]$ have exactly $\lfloor d/4 \rfloor + \lceil d/4 \rceil = d/2$ neighbors in (two) bubble gadgets. Let $y_1, \dots, y_{nt}$ be the ordering of the vertices in $\bigcup_x N_x$ from left to right in how they appear in Figure 4. We neatly attach a bubble gadget to (v_{c_1}, d_{c_1}, y_1) by a $(\lceil d/4 \rceil, \lceil d/4 \rceil, d + 1 - 2\lceil d/4 \rceil)$-attachment. Similarly, we neatly attach a bubble gadget to $(v_{c_m}, d_{c_m}, y_{nt})$ by a $(\lceil d/4 \rceil, \lceil d/4 \rceil, d + 1 - 2\lceil d/4 \rceil)$-attachment. Finally for every $i \in [nt-2]$, we neatly attach a bubble gadget to S'_i made of the three vertices y_i, y_{i+1}, y_{i+2} with a $(1, d/2, d/2)$-attachment. This finishes the construction.

We make some observations. As all the bubble gadgets are neatly attached, no two vertices outside a bubble gadget B can share a neighbor in B.

▶ **Observation 15.** *For every $j \in [m]$, v_{c_j}, d_{c_j} each have exactly $d/2$ neighbors in bubble gadgets (all of which are non-adjacent to any other vertex outside their respective bubble).*

Vertices in $\bigcup_x N_x$ have more neighbors in bubbles.

▶ **Observation 16.** *Every $v \in \bigcup_x N_x$ has at least $d/2 + 1$ neighbors in bubble gadgets (all of which are non-adjacent to any other vertex outside their respective bubble).*

5.4 Correctness

We can now show the following strengthening of Theorem 4.

▶ **Theorem 17.** *For every fixed even integer $d \geqslant 8$, deciding if an input graph has symmetric difference at most d is co-NP-complete.*

As the graph $G := G(\varphi)$ presented in Section 5.3 can be constructed in polynomial time, we shall simply check the equivalence between the satisfiability of φ and the existence of a $(d+1)$-diverse induced subgraph of $G(\varphi)$. We recall that, by definition, G has *not* symmetric difference at most d if and only if it has a $(d+1)$-diverse induced subgraph.

▶ **Lemma 18.** *If φ is satisfiable, then G admits a $(d+1)$-diverse induced subgraph.*

Proof. Let $\mathcal{A}$ be a satisfying assignment of φ. For each variable x of φ, we delete vertex $\neg x$ if $\mathcal{A}$ sets x to true, and we delete vertex x otherwise (if $\mathcal{A}$ sets x to false). Let us call H the obtained induced subgraph of G (with at least two vertices). We claim that H has no pair of d-twins, and successively rule out such pairs

(i) within the same bubble,
(ii) between a vertex in a bubble B and a vertex outside B (but possibly in another bubble),
(iii) between two vertices both outside every bubble gadget.

(i) As H contains all the vertices of G on which bubble gadgets are attached, by Observation 13, no two distinct vertices in the same bubble are d-twins.

(ii) Let us fix a bubble gadget B attached to S, and two vertices $u \in V(B)$ and $v \in V(H) \setminus V(B)$. First observe that u has at least $d/2+1$ neighbors in $V(B)$ (hence in H) that are not neighbors of v. All the vertices $v \in V(H) \setminus (V(B) \cup S)$ have at least $d/2$ neighbors in H that are not neighbors of u. For these vertices v, $\text{sd}_H(u,v) > d$. We thus focus on the case when $v \in S$. We can assume that v is some v_{c_j} or d_{c_j}, as any other vertices have at least $d/2$ neighbors outside B. We can further assume that u is in the top row or rightmost column of B, otherwise it has d neighbors that are not neighbors of u (and u has at least one private neighbor). Now we observe that

$$\text{sd}_H(u,v) \geqslant |N_H(u) \setminus N_H[v]| + |N_H(v) \setminus N_H[u]| \geqslant d/2+1+d/2-1-\lceil d/4 \rceil + \lfloor d/4 \rfloor + 2 \geqslant d+1,$$

where $d/2+1$ lower bounds the number of neighbors of u whose neighborhood is included in $V(B)$, $d/2-1-\lceil d/4 \rceil$ lower bounds the number of neighbors of u in the top row or rightmost column of B that are not adjacent to v, $\lfloor d/4 \rfloor$ lower bounds the number of neighbors of v in another bubble than B, and 2 accounts for the at least two neighbors $v_{c_{j-1}}, d_{c_{j-1}}$ or $v_{c_{j+1}}, d_{c_{j+1}}$ of v, whichever exist. (Here we need that there are at least two clauses.)

(iii) Let u, v be two distinct vertices outside every bubble gadget. Vertex u (resp. v) has at least $d/2$ neighbors that are not neighbors of v (resp. u). This holds by Observations 15 and 16, and the fact that every vertex x_i or $\neg x_i$ is adjacent to N_{x_i}, while no other vertex outside the bubble gadgets is adjacent to any vertex in N_{x_i}. Furthermore, as $|N_{x_i}| = d/2+1$ and vertices in $\bigcup_x N_x$ have at least $d/2+1$ neighbors in bubble gadgets, the only pairs that could be d-twins in H are made of two vertices in clause gadgets. As there are at least three clauses in φ, two vertices u, v from distinct clause gadgets have at least two additional private neighbors. Thus we can assume that $u = v_{c_j}$ and $v = d_{c_j}$ for some $j \in [m]$. As $\mathcal{A}$ is a satisfying assignment, at least one vertex x or $\neg x$ adjacent to v_{c_j} has survived in H. Hence $\text{sd}_H(u,v) \geqslant d/2 + d/2 + 1 = d+1$. ◀

▶ **Lemma 19** ($\star$)**.** *If φ is not satisfiable, then G has no $(d+1)$-diverse induced subgraph.*

6 Discussion and open problems

Degeneracy can be defined either by degeneracy ordering for vertices, or by the existence of vertices of small degree in all the induced subgraphs. And despite sd-degeneracy and symmetric difference arising as dense counterparts to these two equivalent definitions, they are not equivalent: classes of bounded symmetric difference are strictly contained in classes of bounded sd-degeneracy. Using signed tree models, we achieve an adjacency labeling scheme for classes of bounded sd-degeneracy that is tight up to logarithmic factors. The necessity of these additional logarithmic factors remains questionable. Moreover, an optimal adjacency labeling scheme for classes of bounded symmetric difference is yet to be found.

▶ **Question 1.** *Is there an $O(\sqrt{n})$-adjacency labeling scheme for classes of bounded sd-degeneracy? Is there an $O(\log n)$-adjacency labeling scheme for classes of bounded symmetric difference?*

On the other hand, we prove a surprising phenomenon: not only both symmetric difference and sd-degeneracy lead to classes that are hard to recognize, but they respectively lead to para-NP-complete and para-co-NP complete problems. However, the existence of a polynomial-time approximation for remains open.

▶ **Question 2.** *Is there a polynomial-time algorithm to compute an approximation of symmetric difference (and sd-degeneracy)?*

References

1 Bogdan Alecu, Aistis Atminas, and Vadim V. Lozin. Graph functionality. *J. Comb. Theory, Ser. B*, 147:139–158, 2021. doi:10.1016/j.jctb.2020.11.002.

2 Aistis Atminas, Andrew Collins, Vadim V. Lozin, and Victor Zamaraev. Implicit representations and factorial properties of graphs. *Discret. Math.*, 338(2):164–179, 2015. doi:10.1016/j.disc.2014.09.008.

3 Édouard Bonnet, Julien Duron, John Sylvester, Viktor Zamaraev, and Maksim Zhukovskii. Tight bounds on adjacency labels for monotone graph classes. *arXiv preprint arXiv:2310.20522*, 2023.

4 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.35.

5 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. *Combinatorial Theory, 2 (2)*, 2022.

6 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width V: linear minors, modular counting, and matrix multiplication. In *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023*, volume 254 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.15.

7 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

8 Édouard Bonnet, Jaroslav Nesetril, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *arXiv preprint arXiv:2102.06880*, 2021.

9 Reinhard Diestel. *Graph Theory*. Springer Berlin Heidelberg, 2017.

10 Jan Dreier, Ioannis Eleftheriadis, Nikolas Mählmann, Rose McCarty, Michal Pilipczuk, and Szymon Torunczyk. First-order model checking on monadically stable graph classes. *CoRR*, abs/2311.18740, 2023. doi:10.48550/arXiv.2311.18740.

11 Paul Erdős and András Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica*, 17(1-2):61–99, 1966.

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

13 Hamed Hatami and Pooya Hatami. The implicit graph conjecture is false. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 1134–1137. IEEE, 2022.

14 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. In *Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC 1988)*, pages 334–343, 1988.

15 Alexandr V. Kostochka. Lower bound of the Hadwiger number of graphs by their average degree. *Combinatorica*, 4(4):307–316, 1984.

16 Colin McDiarmid and Tobias Müller. Integer realizations of disk and segment graphs. *Journal of Combinatorial Theory, Series B*, 103(1):114–143, 2013.

17 John Harold Muller. *Local structure in graph classes.* PhD thesis, Georgia Institute of Technology, 1988.

18 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics.* Springer, 2012. doi:10.1007/978-3-642-27875-4.

19 Sergey Norin, Luke Postle, and Zi-Xia Song. Breaking the degeneracy barrier for coloring graphs with no K_t minor. *Advances in Mathematics*, 422:109020, 2023.

20 Andrew Thomason. An extremal function for contractions of graphs. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 95, pages 261–265. Cambridge University Press, 1984.

21 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discret. Appl. Math.*, 8(1):85–89, 1984. doi:10.1016/0166-218X(84)90081-7.

22 Emo Welzl. Partition trees for triangle counting and other range searching problems. In Herbert Edelsbrunner, editor, *Proceedings of the Fourth Annual Symposium on Computational Geometry (SoCG 1988)*, pages 23–33. ACM, 1988. doi:10.1145/73393.73397.

23 Édouard Bonnet, Julien Duron, John Sylvester, and Viktor Zamaraev. Symmetric-difference (degeneracy) and signed tree models, 2024. arXiv:2405.09011.

A Proof of Proposition 1

Take any n-vertex graph G, and fix an arbitrary ordering $v_1, v_2, \dots, v_n$ of its vertices. There is a graph G' with at most $(n-1)(n-3)$ vertices that both contains G as an induced subgraph, and has sd-degeneracy at most 1. The graph G' can be built by adding to G, for every $i \in [n-1]$, up to $n-3$ vertices $v_{i,1}, \dots, v_{i,h_i}$ gradually "interpolating" between the neighborhood of v_i and that of v_{i+1} (in G). For instance, $v_{i,1}, \dots, v_{i,h'_i}$ remove one-by-one the neighbors of v_i that are not neighbors of v_{i+1}, and $v_{i,h'_i+1}, \dots, v_{i,h_i}$ add one-by-one the neighbors of v_{i+1} that are not neighbors of v_i. (We put no edge between a pair of added vertices $v_{i,p}, v_{i',p'}$.) Then an ordering witnessing sd-degeneracy at most 1 is $v_1, v_{1,1}, v_{1,2}, \dots, v_{1,h_1}, v_2, v_{2,1}, v_{2,2}, \dots, v_{2,h_2}, \dots, v_{n-1}, v_{n-1,1}, v_{n-1,2}, \dots, v_{n-1,h_{n-1}}, v_n$, for which the 1-twin of a vertex is its successor.

B More context on signed tree models

A wide range of structural graph invariants, called width parameters, can be expressed via so-called *tree layouts* (or at least parameters functionally equivalent to them can). A *tree layout* of an n-vertex graph G is a full binary tree T such that the leaves of T, that we may denote by $L(T)$, are in one-to-one correspondence with $V(G)$. Width parameters are typically defined through evaluating a particular function on bipartitions of $V(G)$ made by the two connected components of T when removing one edge of T. The width is then the minimum over tree layouts of the maximum over all such evaluations. In the definition of signed tree models, we depart from this viewpoint, and instead augment T with a sparse structure encoding the graph G.

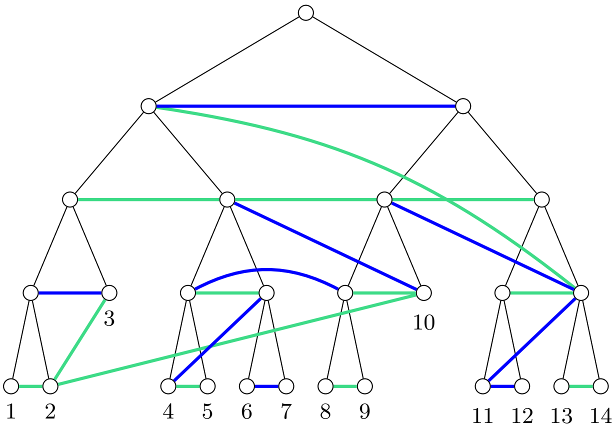


Figure 5 The signed tree model of Figure 1 made clean.

Every graph of twin-width d admits a signed tree model with $A(T) = \emptyset$ and width at most $d + 1$. *Tree models* or *twin-decompositions* are signed tree models with $A(T) = \emptyset$, and further technical requirements. We observe that similar objects to signed tree models were utilized in [6] to attain a fast matrix multiplication on matrices of low twin-width. We will not need a definition of twin-width, and refer the interested reader to [7]. In Section 1 we also mentioned Welzl orders with low alternation number [22], let us now elaborate on that.

A *Welzl order* of *alternation number* d for a graph G is a total order $<$ on $V(G)$ such that the neighborhood of every vertex is the union of at most d intervals along $<$. We claim that bipartite graphs $G = (X \uplus Y, E(G))$ with a Welzl order $<$ of alternation number d admit a signed tree model of width $2d$. Note that we can assume that for every $x \in X$ and $y \in Y$, $x < y$. We build a signed tree model $(T, A(T), B(T))$ of G as follows. Let us call *binary comb* a full binary tree whose internal nodes induce a path, rooted at an endpoint of this path. We make the root of T adjacent to the roots of two binary combs with $|X|$ and $|Y|$ leaves, respectively. The leaves are labeled from left to right with the vertices of G in the order $<$. To simplify the notations, assume that these labels describe $[n]$ in the natural order. To represent that vertex $i \in X$ has $[j, k] \subseteq Y$ in the partition of its open neighborhood into maximal intervals, we add a blue edge between leaf i and the parent of k, and a green edge between i and the parent of $j - 1$ (to stop the interval). Finally observe that $(V(T), A(T) \cup B(T))$ has maximum degree at most $2d$. (The subtree whose leaves are the vertices of X need not be a binary comb.)

A similar construction would work for graphs G of chromatic number q, and would yield a signed tree model of width $2(q - 1)d$. A more permissive definition of signed tree models, allowing leaf-to-ancestor transversal edges, would give models of width $2d$ for any graph with a Welzl order of alternation number d. However, with this alternative definition, the consequences of Section 4 would not follow. Hence we stick to the given definition of signed tree models.

C Proof of Lemma 14

We first deal with the case when a vertex v of the top row (in the entire B) has no neighbor in $V(H) \setminus V(B)$. By symmetry, assume that v is the topmost vertex of the first column. Vertex v is thus d-twin with all the other vertices of the first column. Hence by Observation 12, either v is not in H, or none of the $d/2+1$ vertices below v are in H.

If the latter holds, then any two vertices in the same column, outside the top row and rightmost column, are now d-twins. By Observation 12 within these vertices, H can only contain at most one vertex per column. In turn, the kept vertices are d-twins, so at most one can be kept overall. We conclude since the vertices of $N_H(S) \cap V(B)$ have at most two neighbors in S.

We now suppose that v is not in H. Then, in each row but the topmost, the vertices in the first and penultimate columns are d-twins. Thus, within each pair, at most one vertex can be in H. This implies that any two vertices in the same column, outside the top row and rightmost column, are now d-twins. Thus we conclude as in the previous paragraph.

We now deal with the case when two vertices x, y of the right most column have no neighbor in $V(H) \setminus V(B)$. By symmetry, we can assume that x is in the second row, and y is in the third row. Then x (resp. y) is d-twin with the vertex just to its left. After one vertex is removed in each pair, in each column but the last two, the vertices in the second and third rows have become d-twins. Therefore, H can only contain at most one vertex from all these pairs. We reach again the state that any two vertices in the same column, outside the top row and rightmost column, are d-twins, and conclude as previously.

D Proof of Lemma 19

For each variable x, the vertices $x, \neg x$ are 3-twins, thus at least one of them has to be removed in a $(d+1)$-diverse induced subgraph. The kept literals (if any) define a (partial) truth assignment. By assumption, this assignment does not satisfy at least one clause c_j. This implies that v_{c_j}, d_{c_j} are d-twins in the corresponding induced subgraph. Indeed, they each have exactly $d/2$ private neighbors in bubble gadgets, and no other private neighbor.

By Lemma 14, the bubble attached to S_j is reduced to at most one vertex, say w_j (if any). In turn, this makes the pairs $v_{c_{j-1}}, d_{c_{j-1}}$ and $v_{c_{j+1}}, d_{c_{j+1}}$ d-twins (when they exist). Indeed their symmetric difference is at most $3 + \lceil d/4 \rceil + 1 \leqslant d$, where 3 accounts for the three literals of the clause, and 1 for vertex w_j. This iteratively collapses every bubble attached to some $S_{j'}$ to a single vertex, as well as the two bubble gadgets attached to $\{v_{c_1}, d_{c_1}, y_1\}$ and $\{v_{c_m}, d_{c_m}, y_{nt}\}$, in say, w_0 and w_m. Now all the vertices w_j (for $j \in [0,m]$) are 6-twins, so at most one can be kept. We recall that at most one vertex per clause gadget could be kept. For j going from 1 to $m-1$, the vertex kept (if any) from the clause gadget of c_j is an 8-twin of the vertex kept in the next surviving clause gadget. This implies that from all the clause gadgets and all the bubble gadgets attached to them, one can only keep at most one vertex overall, say z. This vertex has degree at most 3 in the resulting induced subgraph.

Vertices y_1, y_2 are now d-twins, so the bubble gadget attached to S'_1 collapses to at most one vertex. Vertices y_1 and z are now 5-twins, so at most one can survive, which we keep calling z. Even if y_2 is kept, it is now a d-twin of y_3, thus at most one of y_2, y_3 can be kept. This implies the collapse of the bubble gadget attached to S'_2 to at most one vertex, absorbed by z. In turn, y_2 and z collapse to a single vertex. This process progressively eats up all the vertices y_j, and all the bubble gadgets attached to them. As soon as a vertex x or $\neg x$ has three remaining neighbors, it becomes a 6-twin of z, and is absorbed by it. We end up with the single vertex z.

First-Fit Coloring of Forests in Random Arrival Model

Bartłomiej Bosek
Institute of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Grzegorz Gutowski
Institute of Theoretical Computer Science, Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland

Michał Lasoń
Institute of Mathematics of the Polish Academy of Sciences, ul. Śniadeckich 8, 00-656 Warszawa, Poland

Jakub Przybyło
AGH University of Krakow, Faculty of Applied Mathematics, al. A. Mickiewicza 30, 30-059 Kraków, Poland

Abstract

We consider a graph coloring algorithm that processes vertices in order taken uniformly at random and assigns colors to them using First-Fit strategy. We show that this algorithm uses, in expectation, at most $(1 + o(1)) \cdot \ln n \,/\, \ln \ln n$ different colors to color any forest with n vertices. We also construct a family of forests that shows that this bound is best possible.

2012 ACM Subject Classification Theory of computation

Keywords and phrases First-Fit, Online Algorithms, Graph Coloring, Random Arrival Model

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.33

Funding *Bartłomiej Bosek*: Partially supported by grant no. 2023/51/B/ST6/02833 from National Science Centre, Poland.
Grzegorz Gutowski: Partially supported by grant no. 2023/51/B/ST6/02833 from National Science Centre, Poland.
Michał Lasoń: Partially supported by grant no. 2019/34/E/ST1/00087 from National Science Centre, Poland.

1 Introduction

A *proper k-coloring* of a graph $G = (V, E)$ is a function $c : V \mapsto \{1, 2, \ldots, k\}$ that assigns a color $c(v)$ to each vertex $v \in V$ so that any adjacent vertices are colored differently, i.e., for each edge $\{u, w\} \in E$, $c(u) \neq c(w)$ is satisfied. For a given graph G, the smallest number k for which G admits a proper k-coloring is the *chromatic number* of G and is denoted by $\chi(G)$. Graph coloring is one of the most prominent disciplines within graph theory, with plenty of variants, applications, and deep connections to theoretical computer science. Coloring problems arise naturally in various job scheduling and resource allocation optimization scenarios.

The graph coloring problem is also very popular and well motivated in the online setting, with applications in job scheduling, dynamic storage allocation and resource management [9, 11, 12]. In the *online graph coloring* problem, an online algorithm receives as input a graph $G = (V, E)$ presented in an online fashion. The vertices of V are revealed one after one, in a *presentation order* $v_1 \ll v_2 \ll \ldots \ll v_n$. When a new vertex v_t is revealed in the t-th round, for $1 \leqslant t \leqslant n$, all the edges connecting v_t with vertices in $V_{t-1} = \{v_1, \ldots, v_{t-1}\}$ are also revealed. An online algorithm $\mathcal{A}$ has to immediately assign a feasible color to v_t, that is,

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 33; pp. 33:1–33:10
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a color that is different from those assigned to the neighbors of v_t in V_{t-1}. The goal is to minimize the total number of colors used.

The performance of an online graph coloring algorithm $\mathcal{A}$ is often measured by its *competitive ratio*. If we denote by $\chi_{\mathcal{A}}(G, \lll)$ the number of colors used by $\mathcal{A}$ when coloring a graph G in a presentation order $\lll$, then the competitive ratio of $\mathcal{A}$ is the maximum ratio $\chi_{\mathcal{A}}(G, \lll) \,/\, \chi(G)$ over all graphs G and all orders $\lll$. This means, that for a single graph, the analysis focuses on the worst case scenario, and the presentation order is often considered to be selected by an adversary. If $\mathcal{A}$ is a randomized algorithm, then it is usual to measure the competitive ratio with respect to the expected number of colors used when coloring G in order $\lll$ over all the random choices of $\mathcal{A}$.

Contrary to the offline setting, even coloring 2-colorable graphs is not easy for online algorithms. It is impossible to construct an online algorithm that would use any constant number of colors to color all 2-colorable graphs. Thus, the research focuses on restricted graph classes, and competitive ratio is often expressed as a function of the number of vertices in a graph. In fact, even for inputs restricted to online forests with n vertices, the best competitive ratio that can be achieved by either randomized or deterministic online coloring algorithm is $\Theta(\log n)$ colors [3, 6, 1]. The optimal number of colors used by any deterministic online algorithm to color 2-colorable graphs with n vertices is known to be somewhere between $2\log_2 n - 10$ (by a result of Gutowski et al. [5]) and $2\log_2 n$ (see a paper by Lovász, Saks, and Trotter [10]).

An algorithm known as First-Fit is arguably the simplest and the most understood of all online deterministic coloring algorithms. When a vertex v_t is revealed in the t-th round, First-Fit picks the least positive integer i which does not occur as a color of any of the previously colored neighbors of v_t and assigns i as a color of v_t. First-Fit performs well on online trees, achieving the competitive ratio of $\Theta(\log n)$ within this class. To be more precise, Bean [3] and, independently, Gyárfás and Lehel [6] proved that First-Fit uses at most $\log_2 n + 1$ colors on forests with n vertices. Irani [8] generalized this result, showing that First-Fit uses $O(d \log n)$ colors on d-degenerate graphs with n vertices. Later, Balogh et. al. [2] with corrections of Chang and Hsu [4] improved the above result to at most $(\log_{\frac{d+1}{d}} n + 2)$ colors. For bipartite graphs, there is an easy construction by Lovász, Saks, and Trotter [10] that shows that First-Fit uses as many as $\frac{n}{2}$ colors to color a 2-colorable graph with n vertices.

In this paper, we are interested in the *random arrival model* for online algorithms, that tries to focus on the average case, rather than the worst case scenario. In this model, the presentation order of a graph is not determined by an adversary, but instead it is selected uniformly at random from all possible permutations of the vertex set. The *performance ratio* of a deterministic online algorithm $\mathcal{A}$ on a graph G is measured by the expectation $\mathbb{E}_{\lll}\left[\frac{\chi_{\mathcal{A}}(G,\lll)}{\chi(G)}\right]$ over the random choice of $\lll$. The performance ratio of $\mathcal{A}$ on a class of graphs $\mathcal{G}$ is the the maximum performance ratio taken over all graphs in $\mathcal{G}$.

In particular, our intention is to start the systematic study of the performance ratio of First-Fit coloring algorithm in random arrival model. For a class of graphs $\mathcal{G}$, let $\mathrm{RFF}_{\mathcal{G}}(n)$ denote the maximum performance ratio of First-Fit taken over all graphs in $\mathcal{G}$ with n vertices, i.e.,

$$\mathrm{RFF}_{\mathcal{G}}(n) = \max_{G \in \mathcal{G}, |G|=n} \mathbb{E}_{\lll}\left[\frac{\chi_{\mathrm{FF}}(G, \lll)}{\chi(G)}\right].$$

For an easy comparison to the adversarial model, we define:

$$\mathrm{FF}_{\mathcal{G}}(n) = \max_{G\in\mathcal{G},|G|=n} \max_{\ll} \left[\frac{\chi_{\mathrm{FF}}(G,\ll)}{\chi(G)}\right].$$

In this paper, we prove the following result, which establishes the performance of First-Fit on the class of forests.

▶ **Theorem 1.** *For the class $\mathcal{F}$ of forests, we have* $\mathrm{RFF}_{\mathcal{F}}(n) = ({}^1\!/\!{}_2 \pm o(1)) \cdot \ln n \,/\, \ln\ln n$.

As mentioned above, we have $\mathrm{FF}_{\mathcal{F}}(n) = \Theta(\log n)$. Theorem 1 shows that the randomization of the presentation order gives a noticeable, yet rather moderate increase in performance compared to the adversarial model.

One can also consider the following natural off-line graph coloring algorithm. Given a graph G, it selects an order $\ll$ of vertices of G uniformly at random. Then it uses First-Fit strategy to color vertices of G in order $\ll$. Theorem 1 shows that this algorithm uses $O(\log n \,/\, \log\log n)$ colors in expectation to color any forest. This algorithm does not compete with easy 2-coloring algorithms based on graph traversal. Our objective, however, is to establish the groundwork for a systematic analysis of First-Fit, and other simple online algorithms, in the random arrival model for other graph classes, such as d-degenerate graphs, bipartite graphs, and subsequently, k-colorable graphs. It should be noted that for 3-colorable graphs, the most effective known randomized online algorithm utilizes expected $O\big(n^{1/2}\big)$ colors, as proved by Halldórsson [7]. Considering that the best lower bound for 3-colorable graphs is in the order of $\Omega\big(\log^2 n\big)$, as demonstrated by Vishwanathan [13], it appears that the direction we are pursuing holds promise for intriguing outcomes.

Another rationale for this line of inquiry is that First-Fit in random arrival model serves as an illustration for the following distributed coloring algorithm that works in the synchronous model. Each vertex of the graph represents a computation node, and edges represent communication links. To divide nodes into independent subsets, each node performs a sleep for a random number of units of time, and then assigns to itself the first possible number not assigned to any of the neighbors. This algorithm uses a small number of messages and is quite fast. Although we refrain from further delving into this setting, we underscore the versatility afforded by our analysis.

The paper is organized as follows. In Section 2 we show that First-Fit uses at most $(1 + o(1)) \cdot \ln n \,/\, \ln\ln n$ different colors in expectation on any forest with n vertices. In Section 3 we construct a family of trees for which First-Fit uses $(1 - o(1)) \cdot \ln n \,/\, \ln\ln n$ different colors in expectation. The final section contains brief comments and some open problems regarding the analyzed problem.

2 Upper bound

In this section we show that First-Fit uses at most $O(\log n \,/\, \log\log n)$ different colors in expectation on any forest with n vertices. For this purpose let us first make a basic observation. For any fixed graph $G = (V, E)$ and any order $\ll$ of V, we denote by $G^{\ll}$ the acyclic directed graph obtained from G by orienting every edge $\{u, v\} \in E$ so that it is oriented from u to v if and only if $u \ll v$.

▶ **Observation 2.** *For any graph $G = (V, E)$, a presentation order $\ll$ of V, a positive integer i, and a vertex $v \in V$, if First-Fit assigns color i to v when coloring G in order $\ll$ then there is a directed path in $G^{\ll}$ with i vertices and v as the last vertex.*

Proof. We prove the observation by induction with respect to i. The statement is trivial for $i = 1$, so let us assume $i \geqslant 1$ and that the thesis holds for all values up to i. Suppose that First-Fit assigns color $i + 1$ to vertex v in some round of the algorithm. At this point, v has a neighbor u with $u \ll v$ that was assigned with color i in some earlier round. By induction hypothesis, there is a directed path P in $G^{\ll}$ with i vertices and u as the last vertex. Since $u \ll v$, and $G^{\ll}$ is acyclic we know that v does not belong to P. Thus, we get the desired path by extending P with directed edge (u, v). ◀

We call a simple path P in a directed graph to be *bidirected* if it is either a directed path, or P can be split into two directed paths, each starting at one of the end points of P, and both sharing the same last vertex.

▶ **Corollary 3.** *For any forest $T = (V, E)$, a presentation order $\ll$ of V, and a positive integer $i \geqslant 2$, if First-Fit uses color i when coloring T in order $\ll$ then there is a bidirected path in $T^{\ll}$ with $2i - 2$ vertices.*

Proof. For $i = 2$, by Observation 2 we get a directed path with 2 vertices. Suppose $i \geqslant 3$, and that First-Fit assigns color i to some vertex v of T. At this point, v has a neighbor u that is colored $i - 1$, and a different neighbor w that is colored $i - 2$. By Observation 2 there is a directed path with $i - 1$ vertices and u as the last vertex, and a directed path with $i - 2$ vertices and w as the last vertex. As T is a forest, these paths are vertex disjoint. Since $u \ll v$, and $w \ll v$, we get the desired bidirected path in $T^{\ll}$ by extending both paths with directed edges (u, v) and (w, v). ◀

▶ **Lemma 4.** *For the class $\mathcal{F}$ of forests, every $n \geqslant 3$, and an $\alpha_n = \frac{\ln\ln\ln n+1}{\ln\ln n-\ln\ln\ln n-1}$, we have:*

$$\mathrm{RFF}_{\mathcal{F}}(n) \leqslant \frac{(1+\alpha_n)\ln n}{2\ln\ln n} + \frac{3}{2}.$$

Proof. It is straightforward to verify that $\mathrm{RFF}_{\mathcal{F}}(3) = 1$ and $\mathrm{RFF}_{\mathcal{F}}(4) < 2$, hence the lemma holds for $n = 3, 4$. Let us assume that $n \geqslant 5$. Then, since the function $\ln\ln x - \ln\ln\ln x - 1 > 0$ for every $x > e$ except $x = e^e$, then $\ln\ln n - \ln\ln\ln n - 1 > 0$, and hence $\alpha_n > 0$. Consider any forest T with n vertices. Let $k = \left\lceil \frac{(1+\alpha_n)\ln n}{\ln\ln n} \right\rceil$, and observe that our goal is to prove that First-Fit uses at most $k + 2$ colors in expectation when coloring T in a random order. Indeed, we can assume that $\chi(T) = 2$, as otherwise T is an independent set, and First-Fit uses only one color when coloring T in any order. Note that $(1 + \alpha_n)\ln n/\ln\ln n > \ln n/\ln\ln n > 1$. Hence, $k \geqslant 2$. By Corollary 3, for every order $\ll$ of the vertices of T and a positive integer $i \geqslant 2$, if $\chi_{FF}(T, \ll) = i$, then there is a bidirected path with $2i - 2$ vertices in $T^{\ll}$. Consider any two vertices x, y of T. There is at most one simple path in T with end points x and y. If this path exists, and is a path with exactly $2i - 2$ vertices, then the probability that this path is bidirected in $T^{\ll}$ for a random order $\ll$ is exactly $2^{2i-3} / (2i - 2)!$. Otherwise, the probability that there is such a path with end points x and y equals 0.

Thus, by the union bound, the probability that there is at least one bidirected path with $2i - 2$ vertices in $T^{\ll}$ is upper bounded by $\left(n^2 / 2\right) \cdot \left(2^{2i-3} / (2i - 2)!\right)$. Hence, the expected number of colors used by First-Fit in a random order satisfies:

$$\mathbb{E}_{\ll}\left[\chi_{FF}(T, \ll)\right] = \sum_{i=1}^{\infty} i \cdot \mathbb{P}(\chi_{FF}(T, \ll) = i) \leqslant k + 1 + \sum_{i=k+2}^{\infty} \frac{i \cdot n^2 \cdot 2^{2i-4}}{(2i-2)!} \leqslant$$

$$= k + 1 + \frac{n^2 \cdot 4^{k-1}}{(2k)!} \cdot \sum_{i=k+2}^{\infty} \frac{i \cdot 2^{2(i-(k+1))}}{(2k+1)\cdot(2k+2)\cdot\ldots\cdot(2i-3)\cdot(2i-2)} \leqslant$$

$$\begin{aligned}
&\leqslant k+1+\frac{n^2\cdot 4^{k-1}}{(2k)!}\cdot\sum_{i=k+2}^{\infty}\frac{2^{2(i-(k+1))}}{1\cdot(2k+2)\cdot\ldots\cdot(2i-3)\cdot 2}\leqslant\\
&\leqslant k+1+\frac{n^2\cdot 4^{k-1}}{(2k)!}\cdot\sum_{i=1}^{\infty}\frac{2^{2i-1}}{(2i-1)!}=\\
&=k+1+\frac{n^2\cdot 4^{k-1}}{(2k)!}\cdot\sinh 2\leqslant\\
&\leqslant k+1+\frac{n^2\cdot 4^{k}}{(2k)!}. \qquad (1)
\end{aligned}$$

For $k=\left\lceil\frac{(1+\alpha_n)\ln n}{\ln\ln n}\right\rceil$, as $x\ln x - x$ is an increasing function for $x\geqslant 1$, and $\alpha_n>0$, $k\geqslant 2$, we have that

$$\begin{aligned}
\ln\left(\frac{(2k)!}{4^k}\right)&\geqslant 2k\ln(2k)-2k-k\ln 4=2k\ln k-2k\geqslant\\
&\geqslant 2\cdot\frac{(1+\alpha_n)\ln n}{\ln\ln n}\cdot\ln\left(\frac{(1+\alpha_n)\ln n}{\ln\ln n}\right)-2\cdot\frac{(1+\alpha_n)\ln n}{\ln\ln n}\geqslant\\
&\geqslant 2\cdot\frac{(1+\alpha_n)\ln n}{\ln\ln n}\cdot\left(\ln\left(\frac{\ln n}{\ln\ln n}\right)-1\right)=\\
&=2\cdot\frac{\ln\ln n}{\ln\ln n-\ln\ln\ln n-1}\cdot\frac{\ln n}{\ln\ln n}\cdot(\ln\ln n-\ln\ln\ln n-1)=\\
&=\ln(n^2),
\end{aligned}$$

and as a consequence, $\frac{(2k)!}{4^k}\geqslant n^2$, hence by (1),

$$\mathbb{E}_{\ll}\left[\chi_{FF}(T,\ll)\right]\leqslant k+2.$$

As this holds for any forest T with n vertices, we conclude that

$$\mathrm{RFF}_{\mathcal{F}}(n)\leqslant\frac{(1+\alpha_n)\ln n}{2\ln\ln n}+\frac{3}{2},$$

which ends the proof. ◀

As $\alpha_n=O(\log\log\log n\,/\,\log\log n)$ in Lemma 4, we immediately get the following corollary.

▶ **Corollary 5.**

$$\mathrm{RFF}_{\mathcal{F}}(n)\leqslant(1/2+o(1))\cdot\ln n\,/\,\ln\ln n\,.$$

3 Lower bound

In this section, for any given $0<\gamma<1$, and a positive integer k we construct a tree T, such that First-Fit uses k colors to color T in a random presentation order with probability at least $1-\gamma$. Thus, $\mathbb{E}_{\ll}[\chi_{FF}(T,\ll)]\geqslant k(1-\gamma)$. The number of vertices in the constructed tree is of the order k^{dk} where d is a constant depending on γ. This implies that $\mathrm{RFF}_{\mathcal{F}}(n)=\Omega(\log n\,/\,\log\log n)$.

For the purpose of analysis, we use a slightly modified, yet equivalent random model. Namely, rather than choosing a permutation in a straightforward manner, we utilize a natural two-stage process. We first associate with every vertex v of a graph G an independent random variable $X_v \sim U[0,1]$ uniformly distributed over $[0,1]$ interval. We call X_v to be the *position* of a vertex v. Then, we order the vertices according to their positions, i.e., so that $u \ll v$ if and only if $X_u \leqslant X_v$. Note that such an order is uniquely determined with probability 1. Moreover, by the symmetry of the process, each resulting vertex permutation is equiprobable.

▶ **Lemma 6.** *For the class $\mathcal{F}$ of forests, we have:*

$$\mathrm{RFF}_{\mathcal{F}}(n) \geqslant (1/2 - o(1)) \cdot \ln n \,/\, \ln\ln n\,.$$

Proof. Fix any $0 < \gamma < 1$, and any integer $k \geqslant 3$. Let $c = 10/\gamma^2$ and $r = \lceil ck \ln k \rceil$. Set $\varepsilon_i = i\gamma/k$ for $i = 1, 2, \ldots, k$. We recursively define rooted trees $T^r_1, T^r_2, \ldots, T^r_k$ as follows. The tree T^r_1 is a single vertex, and for $i = 1, 2, \ldots, k-1$, the tree T^r_{i+1} is constructed of r copies of each of the trees T^r_j with $j = 1, 2 \ldots, i$ by joining their roots to a single additional vertex – the root of T^r_{i+1} (hence the root of T^r_{i+1} has degree ri). See Figure 1.

Consider First-Fit coloring of T^r_i in a random presentation order, for any fixed $i \in \{1, 2, \ldots, k\}$. We assume that the presentation order is given by the positions $X_v \in [0,1]$ drawn uniformly at random for every vertex v of T^r_i. By the construction of T^r_i, the longest simple path ending at the root of T^r_i includes at most i vertices. From Observation 2 we immediately obtain the following claim.

▷ Claim 7. The color assigned by First-Fit to the root of T^r_i when coloring T^r_i in any order, does not exceed i.

We define B_i to be the random event that First-Fit assigns color smaller then i to the root vertex of T^r_i when coloring T^r_i in a random presentation order.

▷ Claim 8. For every $i = 1, 2, \ldots, k$ we have

$$\mathbb{P}(B_i) \leqslant \varepsilon_i = \frac{i\gamma}{k}.$$

Proof. We prove the claim by induction on i. For $i = 1$, First-Fit assigns color 1 to the only vertex of T^r_1 and hence $\mathbb{P}(B_1) = 0$. Now, for the induction step, we fix any $1 \leqslant i \leqslant k-1$, and assume that

$$\mathbb{P}(B_j) \leqslant \varepsilon_j \tag{2}$$

holds for every $j = 1, 2, \ldots, i$. We shall prove that $\mathbb{P}(B_{i+1}) \leqslant \varepsilon_{i+1}$. For that we focus on the coloring of $T = T^r_{i+1}$. Denote the root of T by v and let $v^{(q)}_j$ with $j = 1, \ldots, i$, $q = 1, \ldots, r$ be the neighbors of v in T where each $v^{(q)}_j$ is the root of one of the r copies of T^r_j attached to v – we denote this copy as $T^{(q)}_j$. See Figure 1. We denote the (random) color First-Fit assigns to any vertex w by $c(w)$.

Suppose the position of v is fixed and equals x. Note that if for some $1 \leqslant j \leqslant i$ there is some q such that $c(v^{(q)}_j) \geqslant j$ and the position of $v^{(q)}_j$ is smaller than x, then $c(v) \neq j$. Indeed, due to Observation 2 and the fact that $v^{(q)}_j$ is positioned before v we have that $c(v^{(q)}_j) \leqslant j$ and it is assigned with color j by the assumption $c(v^{(q)}_j) \geqslant j$. Therefore, if there is such a q

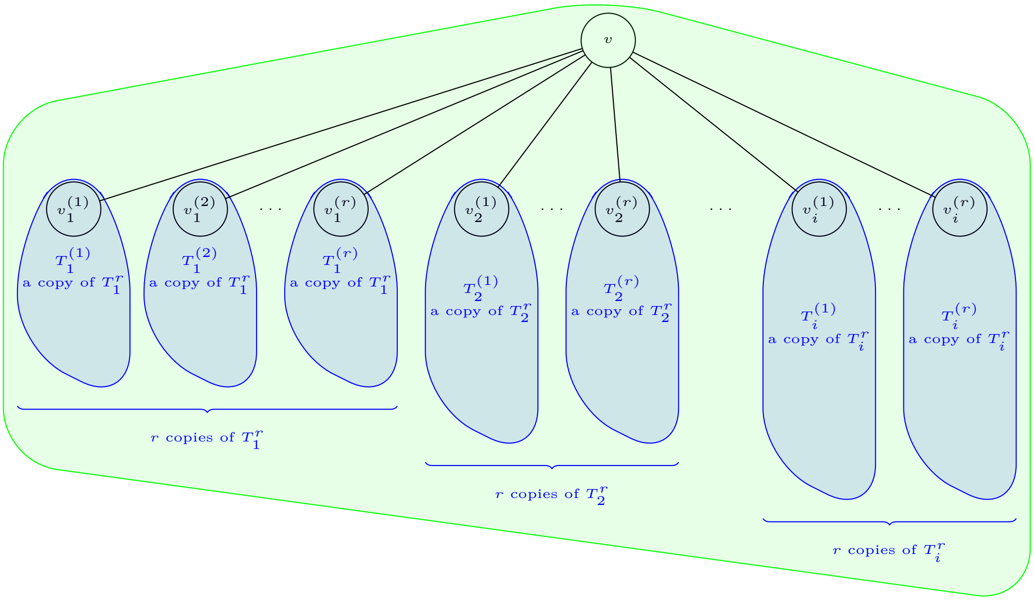

v
$v_1^{(1)}$
$v_1^{(2)}$
$v_1^{(r)}$
$v_2^{(1)}$
$v_2^{(r)}$
$v_i^{(1)}$
$v_i^{(r)}$
$\cdots$
$T_1^{(1)}$ a copy of T_1^r
$T_1^{(2)}$ a copy of T_1^r
$T_1^{(r)}$ a copy of T_1^r
$T_2^{(1)}$ a copy of T_2^r
$T_2^{(r)}$ a copy of T_2^r
$T_i^{(1)}$ a copy of T_i^r
$T_i^{(r)}$ a copy of T_i^r
r copies of T_1^r
r copies of T_2^r
r copies of T_i^r

Figure 1 Recursive construction of the tree T_{i+1}^r.

for every $1 \leqslant j \leqslant i$, then we get $c(v) \geqslant i+1$. This allows for the following inequality,

$$\begin{aligned}\mathbb{P}\left(\overline{B_{i+1}} \mid X_v = x\right) &\geqslant \mathbb{P}\left(\forall j \leqslant i\ \exists q \leqslant r : c(v_j^{(q)}) \geqslant j \wedge X_{v_j^{(q)}} < x \mid X_v = x\right) = \\ &= \prod_{j=1}^{i} \mathbb{P}\left(\exists q \leqslant r : c(v_j^{(q)}) \geqslant j \wedge X_{v_j^{(q)}} < x \mid X_v = x\right) = \\ &= \prod_{j=1}^{i}\left(1 - \prod_{q=1}^{r} \mathbb{P}\left(c(v_j^{(q)}) < j \vee X_{v_j^{(q)}} > x \mid X_v = x\right)\right)\end{aligned} \tag{3}$$

where the two last equalities above follow by the independence of the corresponding events for a fixed value of x. Obviously $\mathbb{P}(X_{v_j^{(q)}} > x \mid X_v = x) = 1 - x$, as positions of the vertices are independent. Further, for any assignment X of positions to all vertices of T, one can consider First-Fit coloring of $T_j^{(q)}$ in order given by the restriction of X to the vertices of $T_j^{(q)}$. Color assigned to the root of $T_j^{(q)}$ in this restricted coloring is not greater then the color assigned to this vertex in the coloring of T. Thus, $\mathbb{P}(c(v_j^{(q)}) < j \mid X_v = x) \leqslant \mathbb{P}(B_j)$, and by (2), for every $j \leqslant i$, we have:

$$\mathbb{P}\left(c(v_j^{(q)}) < j \vee X_{v_j^{(q)}} > x \mid X_v = x\right) \leqslant \min\left\{1, 1 - x + \varepsilon_j\right\}. \tag{4}$$

By (3) and (4), we thus obtain that

$$\begin{aligned}\mathbb{P}\left(B_{i+1}\right) = 1 - \mathbb{P}\left(\overline{B_{i+1}}\right) = 1 - \int_0^1 \mathbb{P}\left(\overline{B_{i+1}} \mid X_v = x\right) dx &\leqslant \\ \leqslant 1 - \int_0^1 \prod_{j=1}^{i} \max\left\{0, 1 - (1 - x + \varepsilon_j)^r\right\} dx &\leqslant \\ \leqslant 1 - \int_{\varepsilon_i}^1 (1 - (1 - x + \varepsilon_i)^r)^i dx, &\end{aligned} \tag{5}$$

where the last inequality follows by the fact that $\varepsilon_j \leqslant \varepsilon_i$ for $j \leqslant i$. Substituting $y = 1 - x + \varepsilon_i$ in (5) we further obtain:

$$\mathbb{P}(B_{i+1}) \leqslant 1 - \int_{\varepsilon_i}^{1} (1 - y^r)^i dy. \tag{6}$$

Let $f(y) = (1 - y^r)^i$. Observe that $f(0) = 1$, $f(1) = 0$, and f is strictly decreasing in $[0, 1]$. Thus, we obtain that $\int_0^{\varepsilon_i} f(y)dy \leqslant \varepsilon_i$, and by (6),

$$\begin{aligned} P(B_{i+1}) &\leqslant \varepsilon_i + 1 - \int_0^1 f(y)dy \leqslant \varepsilon_i + 1 - \left(1 - \frac{\gamma}{2k}\right) \cdot f\left(1 - \frac{\gamma}{2k}\right) \leqslant \\ &\leqslant \varepsilon_i + \frac{\gamma}{2k} + 1 - f\left(1 - \frac{\gamma}{2k}\right). \end{aligned} \tag{7}$$

In order to prove that $P(B_{i+1}) \leqslant \varepsilon_{i+1} = \varepsilon_i + \frac{\gamma}{k}$ it thus remains to show that $f(1 - \frac{\gamma}{2k}) \geqslant 1 - \frac{\gamma}{2k}$. Note that

$$f\left(1 - \frac{\gamma}{2k}\right) = \left(1 - \left(1 - \frac{\gamma}{2k}\right)^r\right)^i \geqslant \left(1 - \left(1 - \frac{\gamma}{2k}\right)^{ck\ln k}\right)^k = \left(1 - \left(1 - \frac{\gamma}{2k}\right)^{\frac{2k}{\gamma}\frac{\gamma}{2}c\ln k}\right)^k.$$

Now, for $\alpha = \frac{2k}{\gamma} \geqslant 1$ we have that $(1 - \frac{1}{\alpha})^\alpha < \frac{1}{e}$. Thus,

$$f\left(1 - \frac{\gamma}{2k}\right) \geqslant \left(1 - \frac{1}{e^{\frac{\gamma c \ln k}{2}}}\right)^k = \left(1 - \frac{1}{k^{\frac{\gamma c}{2}}}\right)^k.$$

As for $0 < \beta = \frac{1}{k^{\frac{\gamma c}{2}}} < 1$ we have that $1 - \beta > e^{-\frac{\beta}{1-\beta}}$, we thus further obtain that

$$f\left(1 - \frac{\gamma}{2k}\right) \geqslant e^{-\frac{\beta k}{1-\beta}} = e^{-\frac{k}{k^{\frac{\gamma c}{2}}-1}} \geqslant 1 - \frac{k}{k^{\frac{\gamma c}{2}} - 1}.$$

Now, for $c = \frac{10}{\gamma^2}$, and using $k \geqslant 3$ we get

$$f\left(1 - \frac{\gamma}{2k}\right) \geqslant 1 - \frac{k}{k^{\frac{5}{\gamma}} - 1} \geqslant 1 - \frac{1}{k^{\frac{3}{\gamma}}}.$$

Observe that $k^{\frac{3}{\gamma}} > \frac{2k}{\gamma}$ for $\gamma \in [0, 1]$, and $k \geqslant 3$. This finally gives that $f(1 - \frac{\gamma}{2k}) \geqslant 1 - \frac{\gamma}{2k}$, and consequently, by (7), $P(B_{i+1}) \leqslant \varepsilon_{i+1}$, which ends the proof of Claim 8. ◁

By Claims 7 and 8, First-Fit assigns color k to the root of T_k^r with probability at least $1 - \varepsilon_k$. Using Claim 7 (or Observation 2), one can thus easily deduce the following claim.

▷ Claim 9. $\mathbb{P}(\chi_{FF}(T_k^r, \ll) = k) \geqslant 1 - \gamma$.

▷ Claim 10. T_k^r has exactly $(r + 1)^{k-1}$ vertices.

Proof. We prove the claim by induction on k. For $k = 1$ the claim trivially holds. Let us assume that $k \geqslant 1$ and that the claim holds for all the trees $T_1^r, \ldots, T_k^r$. By the construction of T_{k+1}^r,

$$|T_{k+1}^r| = 1 + r \cdot \sum_{i=1}^{k} |T_i^r| = 1 + r \cdot \sum_{i=1}^{k} (r + 1)^{i-1} = 1 + r \cdot \frac{1 - (r + 1)^k}{1 - (r + 1)} = (r + 1)^k. \qquad ◁$$

For any value of k, we set $\gamma = \frac{1}{\ln k}$, and for this choice of γ we get $c = 10\ln^2 k$ and $r = \lceil 10k\ln^3 k \rceil$. Let us denote by n_k the number of vertices in T_k^r. Then the following hold for k large enough. Firstly, by Claim 10, we have $\ln n_k \geqslant k$. Consequently, again by Claim 10,

$$\begin{aligned}\ln n_k &\leqslant (k-1)\ln\left(2+10k\ln^3 k\right) \leqslant k\left(\ln k + 4\ln\ln k\right)\\ &\leqslant k\left(\ln\ln n_k + 4\ln\ln\ln n_k\right). \qquad (8)\end{aligned}$$

By Claim 9 and (8), we thus finally obtain that

$$\begin{aligned}\mathbb{E}_{\ll}\left[\chi_{FF}(T_k^r, \ll)\right] &\geqslant k\cdot\left(1-\frac{1}{\ln k}\right) \qquad (9)\\ &\geqslant \frac{\ln n}{\ln\ln n + 4\ln\ln\ln n}\cdot\left(1-\frac{1}{\ln\ln n - 2\ln\ln\ln n}\right) = g(n). \qquad (10)\end{aligned}$$

Note that $g(n)$ is an increasing function for large enough n. Note also that n_k is also an increasing function of k. Consider any (large enough) n such that $n_k \leqslant n \leqslant n_{k+1}$ for some k. Then, since $\mathrm{RFF}_{\mathcal{F}}(n)$ is a nondecreasing function of n, by (9) and (10),

$$\begin{aligned}\mathrm{RFF}_{\mathcal{F}}(n) &\geqslant \mathrm{RFF}_{\mathcal{F}}(n_1) \geqslant \frac{1}{2}\cdot\mathbb{E}_{\ll}\left[\chi_{FF}(T_k^r, \ll)\right] \geqslant \frac{1}{2}\cdot k\cdot\left(1-\frac{1}{\ln k}\right)\\ &\geqslant \frac{1}{2}\cdot\left((k+1)\cdot\left(1-\frac{1}{\ln(k+1)}\right)-1\right) \geqslant \frac{1}{2}\cdot\left(g(n_{k+1})-1\right)\\ &\geqslant \frac{1}{2}\cdot\left(g(n)-1\right) = \left(\frac{1}{2}-o(1)\right)\frac{\ln n}{\ln\ln n},\end{aligned}$$

which finishes the proof of Lemma 6. ◀

4 Final comments

Finally, Theorem 1 is obtained by combining the matching bounds of Corollary 5 and Lemma 6. This concludes our investigation of the efficiency of First-Fit in the random arrival model on the class of forests. First-Fit is efficient on this class even in the adversarial model. Still, the randomization of the presentation order allows for some increase in performance. This raises the hope that First-Fit is more effective in the average case than it is in the worst case also on some other graph classes.

The systematic analysis of First-Fit in the random arrival model on other graph classes should continue for the class $\mathcal{B}$ of bipartite graphs. There, First-Fit is known to be extremely inefficient in the adversarial model with $\mathrm{FF}_{\mathcal{B}} = \Theta(n)$. However, there is another simple algorithm [10], a clever modification of First-Fit, that uses $O(\log n)$ colors to color any bipartite graph with n vertices. A natural extension of our research would be to assess the First-Fit algorithm in the random arrival model on bipartite graphs. We anticipate that the outcome will not deviate significantly from the results obtained for forests.

▶ **Conjecture 11.** *First-Fit in the random arrival model uses* $\mathrm{poly}(\log n)$ *colors in expectation when coloring any bipartite graph with* n *vertices.*

Addressing the aforementioned conjecture represents a significant cognitive pursuit. However, a truly remarkable feat would be to establish some nontrivial bounds for 3-colorable graphs. We want to thank Anna Zych-Pawlewicz for inspiring the work on this project.

References

1 Susanne Albers and Sebastian Schraink. Tight bounds for online coloring of basic graph classes. In *ESA 2017: 25th Annual European Symposium on Algorithms, Vienna, Austria, September 4-8, 2017, Proceedings*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.7.

2 József Balogh, Stephen G. Hartke, Qi Liu, and Gexin Yu. On the first-fit chromatic number of graphs. *SIAM Journal on Discrete Mathematics*, 22(3):887–900, 2008. doi:10.1137/060672479.

3 Dwight R. Bean. Effective coloration. *The Journal of Symbolic Logic*, 41(2):469–480, 1976. doi:10.2307/2272247.

4 Gerard Jennhwa Chang and Hsiang-Chun Hsu. First-fit chromatic numbers of d-degenerate graphs. *Discrete Mathematics*, 312(12-13):2088–2090, 2012. doi:10.1016/j.disc.2012.03.029.

5 Grzegorz Gutowski, Jakub Kozik, Piotr Micek, and Xuding Zhu. Lower bounds for online graph colorings. In *ISAAC 2014: 25th International Symposium on Algorithms and Computation, Jeonju, Korea, December 15-17, 2014, Proceedings*, volume 8889 of *Lecture Notes in Computer Science*, pages 507–515. Springer, 2014. doi:10.1007/978-3-319-13075-0_40.

6 András Gyárfás and Jenő Lehel. On-line and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, 1988. doi:10.1002/jgt.3190120212.

7 Magnús M. Halldórsson. Parallel and on-line graph coloring. *Journal of Algorithms*, 23(2):265–280, 1997. doi:10.1006/jagm.1996.0836.

8 Sandy Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994. doi:10.1007/BF01294263.

9 Hal A. Kierstead. Coloring graphs on-line. In *Online Algorithms: The State of the Art*, pages 281–305. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. doi:10.1007/BFb0029574.

10 László Lovász, Michael Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75(1-3):319–325, 1989. Graph theory and combinatorics (Cambridge, 1988). doi:10.1016/0012-365X(89)90096-4.

11 Dániel Marx. Graph colouring problems and their applications in scheduling. *Periodica Polytechnica Electrical Engineering*, 48(1-2):11–16, 2004. URL: https://pp.bme.hu/ee/article/view/926.

12 Lata Narayanan. Channel assignment and graph multicoloring. In *Handbook of Wireless Networks and Mobile Computing*, chapter 4, pages 71–94. John Wiley & Sons, Inc., USA, 2002. doi:10.1002/0471224561.ch4.

13 Sundar Vishwanathan. Randomized online graph coloring. *Journal of Algorithms. Cognition, Informatics and Logic*, 13(4):657–669, 1992. doi:10.1016/0196-6774(92)90061-G.

On the Number of Quantifiers Needed to Define Boolean Functions

Marco Carmosino
MIT-IBM Watson AI Lab, Cambridge, MA, USA

Ronald Fagin
IBM Research-Almaden, San Jose, CA, USA

Neil Immerman
University of Massachusetts, Amherst, MA, USA

Phokion G. Kolaitis
University of California Santa Cruz, CA, USA
IBM Research-Almaden, San Jose, CA, USA

Jonathan Lenchner[1]
IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

Rik Sengupta
MIT-IBM Watson AI Lab, Cambridge, MA, USA

Abstract

The number of quantifiers needed to express first-order (FO) properties is captured by two-player combinatorial games called *multi-structural* games. We analyze these games on binary strings with an ordering relation, using a technique we call *parallel play*, which significantly reduces the number of quantifiers needed in many cases. Ordered structures such as strings have historically been notoriously difficult to analyze in the context of these and similar games. Nevertheless, in this paper, we provide essentially tight bounds on the number of quantifiers needed to characterize different-sized subsets of strings. The results immediately give bounds on the number of quantifiers necessary to define several different classes of Boolean functions. One of our results is analogous to Lupanov's upper bounds on circuit size and formula size in propositional logic: we show that every Boolean function on n-bit inputs can be defined by a FO sentence having $(1+\varepsilon)\frac{n}{\log(n)} + O(1)$ quantifiers, and that this is essentially tight. We reduce this number to $(1+\varepsilon)\log(n) + O(1)$ when the Boolean function in question is sparse.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Computational complexity and cryptography

Keywords and phrases logic, combinatorial games, Boolean functions, quantifier number

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.34

Related Version *Full Version*: `https://arxiv.org/abs/2407.00688` [2]

Funding *Rik Sengupta*: Supported by NSF CCF-1934846.

Acknowledgements The authors acknowledge Ryan Williams for numerous helpful discussions and conversations, Sebastian Pfau for an observation that improved the statement of the Parallel Play Lemma, and the anonymous reviewers for comments and suggestions that improved the quality of this manuscript.

[1] Corresponding author

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 34; pp. 34:1–34:16
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In 1981, Immerman [11] introduced *quantifier number* (QN) as a measure of the complexity of first-order (FO) sentences. For a function $g\colon \mathbb{N} \to \mathbb{N}$, he defined $\mathrm{QN}[g(n)]$ as the class of properties on n-element structures describable by a uniform sequence of FO sentences with $O(g(n))$ quantifiers. He then showed that on *ordered* structures, for $f(n) \geq \log n$, one has:

$$\mathrm{NSPACE}[f(n)] \subseteq \mathrm{QN}[(f(n))^2/\log n] \subseteq \mathrm{DSPACE}[(f(n))^2], \tag{1}$$

thereby establishing an important connection between QN and space complexity and so directly linking a logical object to classical complexity classes.

The same paper [11] described a two-player combinatorial game (which Immerman called the *separability game*), that captures quantifier number in the same way that the more well-known Ehrenfeucht-Fraïssé (EF) game [3, 7] captures quantifier rank (QR). The paper additionally showed that any property whatsoever of n-element *ordered* structures can be described with a sentence having a QR of $\log n + 3$. Since a QR of $\log n + 1$ is required just to distinguish a linear order of size n from smaller linear orders [17], QR has limited power to distinguish properties over ordered structures. QN is potentially a more fine-grained and powerful measure for this purpose. However, owing to the inherent difficulties of the analysis of Immerman's separability game, the study of the game and of QN in general lay dormant for forty years, until the game was rediscovered and renamed the *multi-structural* (MS) game in [4]. In that paper the authors made initial inroads into understanding how to analyze the game, leading to several follow-up works [1, 5, 18]. Other related games to study the number of quantifiers were recently introduced in [9], and close cousins of MS games were used to study formula size in [8, 10]. In [10] the authors study a related problem to ours – they examine the (existential) sentences of minimum size needed to express a particular set of string properties. However, even without the existential restriction, the connection between the minimum size of a sentence and its minimum number of quantifiers is not obvious. It is possible for a property to be expressible only by a much longer sentence with fewer quantifiers than one with more quantifiers.

The MS game is played by two players, Spoiler (**S**, he/him) and Duplicator (**D**, she/her), on two *sets* $\mathcal{A}, \mathcal{B}$ of structures. Essentially, **S** tries to break all partial isomorphisms between all pairs of structures (one from $\mathcal{A}$ and the other from $\mathcal{B}$) over a prescribed number of rounds, whereas **D** tries to maintain a partial isomorphism between *some* pair of structures. Unlike in EF games, **D** has more power in MS games, since she can make arbitrarily many copies of structures before her moves, enabling her to play all possible responses to **S**'s moves. The fundamental theorem for MS games [4, 11] (see Theorem 1) states that **S** has a winning strategy for the r-round MS game on $(\mathcal{A}, \mathcal{B})$ if and only if there is a FO sentence φ with at most r quantifiers that is true for every structure in $\mathcal{A}$ but false for every structure in $\mathcal{B}$. We call such a φ a *separating sentence* for $(\mathcal{A}, \mathcal{B})$. In general, our eventual objective will be to separate a set $\mathcal{A}$ of n-bit strings from all other n-bit strings (i.e., from its complement $\mathcal{A}^{\mathrm{C}}$). This is a particularly interesting question because of its intimate connection to the complexity of *Boolean functions.*

Boolean Functions. Any Boolean function on n-bit strings is specified by two complementary sets, $\mathcal{A}, \mathcal{A}^{\mathrm{C}} \subseteq \{0,1\}^n$, representing the input strings that get mapped to 1 and 0 respectively. For such a function $f\colon \{0,1\}^n \to \{0,1\}$, we say that a FO sentence φ in the vocabulary of strings *defines* the function f if φ is a separating sentence for $(f^{-1}(1), f^{-1}(0))$. Hence, the key results of this paper can be thought of as giving sharp bounds on the number of quantifiers needed to define Boolean functions. Our main results about the definability of Boolean functions are Theorems A and B below.

▶ **Theorem A.** *Given an arbitrary $\varepsilon > 0$, every Boolean function on n-bit strings can be defined by a FO sentence having $(1+\varepsilon)\frac{n}{\log(n)} + O_\varepsilon(1)$ quantifiers, where the $O_\varepsilon(1)$ additive term depends only on ε and not n. Moreover, there are Boolean functions on n-bit strings that require $\frac{n}{\log(n)} + O(1)$ quantifiers to define.*

Say that a family, $\mathscr{F} = \{f_n\}_{n=1}^{\infty}$, of Boolean functions on n-bit strings, is *sparse* if the cardinality of the set of strings mapping to 1 under each f_n is polynomial in n. For example, if $\mathcal{L}$ is a sparse language, then the family of Boolean functions, defined for each n, by the characteristic function of $\mathcal{L}$ restricted to n-bit inputs, is sparse [6,15].

▶ **Theorem B.** *Given an arbitrary $\varepsilon > 0$, and a sparse family, $\mathscr{F} = \{f_n\}_{n=1}^{\infty}$, of Boolean functions on n-bit strings, each function $f_n \in \mathscr{F}$ can be defined by a FO sentence having $(1+\varepsilon)\log(n) + O_\varepsilon(1)$ quantifiers, where the $O_\varepsilon(1)$ additive term depends only on ε and not n. Moreover, there are sparse families of Boolean functions on n-bit strings, the functions of which require $\log(n)$ quantifiers to define.*

Theorem A follows from Theorems 18 and 19 (in Section 5), whereas Theorem B follows from Theorem 16 and Proposition 14 (in Section 5). Theorem 18 can be viewed as a first-order logic analog of the upper bounds obtained by Lupanov for minimum circuit size [13] and minimum propositional formula size [14] to capture an arbitrary Boolean function. Note that *any* property whatsoever of n-bit strings can be captured trivially by a sentence with n existential quantifiers. Similar to Lupanov's bounds, our result shows that we can shave off a factor of $\log(n)$ from this trivial upper bound. Furthermore, Theorem 19 establishes via a counting argument that there are functions with a QN lower bound that essentially matches our worst-case upper bound – a result that can be viewed as a first-order logic analog of the Riordan-Shannon lower bound [16] for propositional formula size.

Parallel Play. A key technical contribution we make in this paper is the Spoiler strategy of *parallel play*, which widens the scope of winning strategies for **S** compared to previous work. The essential idea is for **S** to partition the sets $\mathcal{A}$ and $\mathcal{B}$ into subsets $\mathcal{A}_1 \sqcup \ldots \sqcup \mathcal{A}_k$ and $\mathcal{B}_1 \sqcup \ldots \sqcup \mathcal{B}_k$, and then play k MS "sub-games" in parallel on $(\mathcal{A}_i, \mathcal{B}_i)$. In certain circumstances, **S** can then combine his strategies for each of those sub-games into a strategy for the entire game, and thereby save many superfluous moves. Applying the fundamental theorem, this results in a very small number of quantifiers in the corresponding separating sentence.

Outline of the Paper. This paper is organized as follows. In Section 2, we set up some preliminaries. In Section 3, we precisely formulate what we call the *Parallel Play Lemma* (Lemma 5) and the *Generalized Parallel Play Lemma* (Lemma 6). In Section 4, we develop results on linear orders that are similar to but more nuanced than those in [4,5], with the extra nuance being critical for our subsequent string separation results. In Section 5, we present our results on separating disjoint sets of strings. In Section 6, we wrap up with some conclusions and open problems.

Owing to space constraints, in some places we provide proof sketches, and refer the reader to the full proofs in the appendix of the full version of the paper [2].

2 Preliminaries

Fix a vocabulary τ with finitely many relation and constant symbols. We typically designate structures in boldface ($\mathbf{A}$), their universes in capital letters (A), and sets of structures in calligraphic typeface ($\mathcal{A}$). This last convention includes sets of pebbled structures (see below).

We always use $\log(\cdot)$ to designate the base-2 logarithm. Furthermore, in several results in Section 5, we have an $O(1)$ additive term. This term will always be independent of n. Any additional dependence will be stated in the form of a subscript on the O, e.g., $O_t(1)$ would denote a term independent of n, but dependent on the choice of some parameter t.

Pebbled Structures and Matching Pairs. Consider a palette $\mathcal{C} = \{\mathrm{r}, \mathrm{b}, \mathrm{g}, \ldots\}$ of *pebble colors*, with infinitely many pebbles of each color available. A τ-structure $\mathbf{A}$ is *pebbled* if some of its elements $a_1, a_2, \ldots \in A$ have pebbles on them. There can be at most one pebble of each color on a pebbled structure. There can be multiple pebbles (of different colors) on the same element $a_i \in A$. Occasionally, when the context is clear, we will use the term *board* synonymously with "pebbled structure".

If $\mathbf{A}$ is a τ-structure, and the first few pebbles are placed on elements $a_1, a_2, a_3 \ldots \in A$, we designate the resulting pebbled τ-structure as $\langle \mathbf{A} \mid a_1, a_2, a_3, \ldots \rangle$. Note that $\mathbf{A}$ can be viewed as a pebbled structure $\langle \mathbf{A} \mid \rangle$ with the empty set of pebbles.

By convention, we use r, b, and g for the first three pebbles we play (in that order), as a visual aid in our proofs. Hence, the pebbled structure $\langle \mathbf{A} \mid a_1, a_2, a_3 \rangle$ has pebbles r on $a_1 \in A$, b on $a_2 \in A$, and g on $a_3 \in A$. Note that a_1, a_2, and a_3 need not be distinct.

We say that the pebbled structures $\langle \mathbf{A} \mid a_1, \ldots, a_k \rangle$ and $\langle \mathbf{B} \mid b_1, \ldots, b_k \rangle$ are a *matching pair* if the map $f \colon A \to B$ defined by:

- $f(a_i) = b_i$ for all $1 \leq i \leq k$
- $f(c^{\mathbf{A}}) = c^{\mathbf{B}}$ for all constants c in τ

is an isomorphism on the induced substructures. Note that $\langle \mathbf{A} \mid a_1, \ldots, a_k \rangle$ and $\langle \mathbf{B} \mid b_1, \ldots, b_k \rangle$ can form a matching pair even when $\mathbf{A} \ncong \mathbf{B}$.

Multi-Structural Games. Assume $r \in \mathbb{N}$, and let $\mathcal{A}$ and $\mathcal{B}$ be two sets of pebbled structures, each pebbled with the *same* set $\{x_1, \ldots, x_k\} \subseteq \mathcal{C}$ of pebble colors. The *r-round multi-structural (MS) game on* $(\mathcal{A}, \mathcal{B})$ is defined as the following two-player game, played by two players, **Spoiler** (**S**, he/him) and **Duplicator** (**D**, she/her). In each round i for $1 \leq i \leq r$, **S** chooses either $\mathcal{A}$ or $\mathcal{B}$, and an **unused** color $y_i \in \mathcal{C}$; he then places ("plays") a pebble of color y_i on an element of *every* board in the chosen set ("side"). In response, **D** makes as many copies as she wants of each board on the other side, and plays a pebble of color y_i on an element of each of those boards. **D** wins the game if at the end of round r, there is a board in $\mathcal{A}$ and a board in $\mathcal{B}$ forming a matching pair. Otherwise, **S** wins. For readability, we always call the two sets $\mathcal{A}$ and $\mathcal{B}$, even though the structures change over the course of a game in two ways:

- $\mathcal{A}$ or $\mathcal{B}$ can increase in size over the r rounds, as **D** can make copies of the boards.
- The number of pebbles on each of the boards in $\mathcal{A}$ and $\mathcal{B}$ increases by 1 in each round.

We usually refer to $\mathcal{A}$ as the *left* side, and $\mathcal{B}$ as the *right* side.

Let $\mathcal{A}$ and $\mathcal{B}$ be two sets of pebbled structures, with each pebbled structure containing pebbles colored with $\{x_1, \ldots, x_k\} \subseteq \mathcal{C}$. Let $\varphi(x_1, \ldots, x_k)$ be a FO formula with free variables $\{x_1, \ldots, x_k\}$. We say φ is a *separating formula* for $(\mathcal{A}, \mathcal{B})$ (or φ *separates* $\mathcal{A}$ and $\mathcal{B}$) if:

- for every $\langle \mathbf{A} \mid a_1, \ldots, a_k \rangle \in \mathcal{A}$ we have $\mathbf{A}[a_1/x_1, \ldots, a_k/x_k] \models \varphi$,
- for every $\langle \mathbf{B} \mid b_1, \ldots, b_k \rangle \in \mathcal{B}$ we have $\mathbf{B}[b_1/x_1, \ldots, b_k/x_k] \models \neg\varphi$.

The following key theorem [4,11], stated here without proof, relates the logical characterization of a separating formula with the combinatorial property of a game strategy.

▶ **Theorem 1** (Fundamental Theorem of MS Games, [4, 11]). **S** *has a winning strategy in the r-round MS game on $(\mathcal{A}, \mathcal{B})$ iff there is a formula with $\leq r$ quantifiers separating $\mathcal{A}$ and $\mathcal{B}$.*

In the theorem above, if $\mathcal{A}$ and $\mathcal{B}$ are sets of *unpebbled* structures, and φ is a sentence, we call φ a *separating sentence* for $(\mathcal{A}, \mathcal{B})$.

We note that $\mathbf{D}$ has a clear optimal strategy in the MS game, called the *oblivious* strategy: for each of $\mathbf{S}$'s moves, $\mathbf{D}$ can make enough copies of each pebbled structure on the other side to play all possible responses at the same time. If $\mathbf{D}$ has a winning strategy, then the oblivious strategy is winning. For this reason, the MS game is essentially a single-player game, where $\mathbf{S}$ can simulate $\mathbf{D}$'s responses himself.

We make an easy observation here without proof, that will help us *discard* some boards during gameplay; we can remove them without affecting the result of the game. This will help us in the analysis of several results in the paper.

▶ **Observation 2.** *During gameplay in any instance of the MS game, consider a board $\langle \mathbf{A} \mid a_1, \ldots, a_k \rangle$ such that there is no board on the other side forming a matching pair with it. Then, $\langle \mathbf{A} \mid a_1, \ldots, a_k \rangle$ can be removed from the game without affecting the result.*

Linear Orders. Let $\tau_{\mathsf{ord}} = \langle <;\ \mathsf{min}, \mathsf{max} \rangle$ be the vocabulary of orders, where $<$ is a binary predicate, and min and max are constant symbols. For every $\ell \geq 1$, we shall use L_ℓ to refer to a structure of type τ_{ord}, which interprets $<$ as a total linear order on $\ell + 1$ elements, and min and max as the first and last elements in that total order respectively. Note that there is only one linear order for any fixed value of ℓ. When unambiguous, we may suppress the subscript and refer to the linear order as simply L.

We define the *length* of a linear order L as the size of its universe minus one (equivalently, as the number of edges if the linear order were represented as a path graph). Hence, the length of L_ℓ is ℓ. Since we only consider $\ell \geq 1$, the length is always positive, and min and max are necessarily distinct. Our convention is different from [4] and [5], where the length of a linear order was the number of elements, and the vocabulary had no built-in constants. Note that having min and max is purely for convenience; each can be defined and reused at the cost of two quantifiers.

Let L be a linear order with elements $a < b$. The linear order $L[a, b]$ is the induced linear order on all elements from a to b, both inclusive. If the variables x and y have been interpreted by L so that $x^L = a$ and $y^L = b$, then we shall use $L[x, y]$ and $L[a, b]$ interchangeably; we adopt a similar convention for constants. If pebbles r and b have been placed on L on a and b respectively, we use $L[\mathrm{r}, \mathrm{b}]$ to mean $L[a, b]$.

We will frequently need to consider sets of linear orders. For $\ell \geq 1$, we will use the notation $L_{\leq \ell}$ to denote the set of linear orders of length at most ℓ, and $L_{> \ell}$ to denote the set of linear orders of length greater than ℓ.

Strings. Let $\tau_{\mathsf{string}} = \langle <,\ S\ ;\ \mathsf{min}, \mathsf{max} \rangle$ be the vocabulary of binary strings, where $<$ is a binary predicate, S is a unary predicate, and min and max are constant symbols. We encode a string $w = (w_1, \ldots, w_n) \in \{0, 1\}^n$ by the τ_{string}-structure $\mathbf{B}_w$ having universe $B_w = \{1, \ldots, n\}$, relation $<$ interpreted by the linear order on $\{1, \ldots, n\}$, relation $S = \{i \mid w_i = 1\}$, and min and max interpreted as 1 and n respectively.

For an n-bit string w, and i, j such that $1 \leq i \leq j \leq n$, denote by $w[i, j]$ the substring $w_i \ldots w_j$ of w. Note that $w[i, j]$ corresponds to the induced substructure of $\mathbf{B}_w$ on $\{i, \ldots, j\}$. We will often interchangeably talk about the string w and the τ_{string}-structure $\mathbf{B}_w$, when the context is clear. As in τ_{ord}, having min and max in the vocabulary is purely for convenience.

3 Parallel Play

In this section, we prove our key lemma, that shows how, in certain cases, **S** can combine his winning strategies in two sub-games, playing them in parallel in a single game that requires no more rounds than the longer of the two sub-games.

To understand why this is helpful, note that in general, if a formula φ is of the form $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, the number of quantifiers in φ is the sum of the number of quantifiers in φ_1 and φ_2, even if the two subformulas have the same quantifier structure. We will see that playing parallel sub-games roughly corresponds to taking a φ of the form $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ where the subformulas have the same quantifier prefix, and writing φ with the same quantifier prefix as φ_1 or φ_2, saving half the quantifiers we normally require.

Suppose **S** has a winning strategy for an instance $(\mathcal{A}, \mathcal{B})$ of the r-round MS game. In principle, the choice of which side **S** plays on could depend on **D**'s previous responses. However, note that any strategy $\mathcal{S}$ used by **S** that wins against the oblivious strategy also wins against any other strategy that **D** plays. Therefore, we may WLOG restrict ourselves to strategies used by **S** against **D**'s oblivious strategy. It follows that the choice of which side to play on in every round is completely determined by the instance $(\mathcal{A}, \mathcal{B})$, and independent of any of **D**'s responses. Let $\mathcal{S}$ be such a winning strategy for **S**. We now define the *pattern* of $\mathcal{S}$, which specifies which side **S** plays on in each round, when following $\mathcal{S}$.

▶ **Definition 3.** *Suppose $\mathcal{A}$ and $\mathcal{B}$ are sets of pebbled structures, and assume that* **S** *has a winning strategy $\mathcal{S}$ for the r-round* MS *game on $(\mathcal{A}, \mathcal{B})$. The* pattern *of $\mathcal{S}$, denoted* $\mathsf{pat}(\mathcal{S})$, *is an r-tuple $(Q_1, \dots, Q_r) \in \{\exists, \forall\}^r$, where:*

$$Q_i = \begin{cases} \exists & \textit{if } \mathbf{S} \textit{ plays in } \mathcal{A} \textit{ in round } i, \\ \forall & \textit{if } \mathbf{S} \textit{ plays in } \mathcal{B} \textit{ in round } i. \end{cases}$$

We say that **S** wins the game with pattern $(Q_1, \dots, Q_r)$ *if* **S** *has a winning strategy $\mathcal{S}$ for the game in which* $\mathsf{pat}(\mathcal{S}) = (Q_1, \dots, Q_r)$.

The following lemma is implicit in the proof of Theorem 1.

▶ **Lemma 4.** *For any two sets $\mathcal{A}$ and $\mathcal{B}$ of pebbled τ-structures, the following are equivalent:*

1. **S** *wins the r-round* MS *game on $(\mathcal{A}, \mathcal{B})$ with pattern $(Q_1, \dots, Q_r)$.*
2. *$(\mathcal{A}, \mathcal{B})$ has a separating formula with r quantifiers and quantifier prefix $(Q_1, \dots, Q_r)$.*

Note that Lemma 4 implies that, as long as there is a separating formula φ for $(\mathcal{A}, \mathcal{B})$ with r quantifiers, **S** has a winning strategy for the r-round MS game on $(\mathcal{A}, \mathcal{B})$ that "follows" φ; namely, if $\varphi = Q_1 \dots Q_r \psi$, then in round i, **S** plays in $\mathcal{A}$ if $Q_i = \exists$, and in $\mathcal{B}$ if $Q_i = \forall$. Hence, for the rest of the paper, we will refer to **S** moves in $\mathcal{A}$ and $\mathcal{B}$ as *existential* and *universal* moves respectively. We are now ready to state our main lemma from this section.

▶ **Lemma 5** (Parallel Play Lemma). *Let $\mathcal{A}$ and $\mathcal{B}$ be two sets of pebbled structures, and let $r \in \mathbb{N}$. Suppose that $\mathcal{A}$ and $\mathcal{B}$ can be partitioned as $\mathcal{A} = \mathcal{A}_1 \sqcup \mathcal{A}_2$ and $\mathcal{B} = \mathcal{B}_1 \sqcup \mathcal{B}_2$ respectively, such that for $1 \leq i \leq 2$,* **S** *has a winning strategy $\mathcal{S}_i$ for the r-round MS game on $(\mathcal{A}_i, \mathcal{B}_i)$, satisfying the following conditions:*

1. *Both $\mathcal{S}_i$'s have the same pattern $P = \mathsf{pat}(\mathcal{S}_1) = \mathsf{pat}(\mathcal{S}_2)$.*
2. *At the end of the sub-games, both of the following are true:*
   - *There does not exist a board in $\mathcal{A}_1$ and a board in $\mathcal{B}_2$ forming a matching pair.*
   - *There does not exist a board in $\mathcal{A}_2$ and a board in $\mathcal{B}_1$ forming a matching pair.*

Then **S** *wins the r-round MS game on $(\mathcal{A}, \mathcal{B})$ with pattern P.*

Proof. **S** plays the r-round MS game on $(\mathcal{A}, \mathcal{B})$ by playing his winning strategy $\mathcal{S}_1$ on $(\mathcal{A}_1, \mathcal{B}_1)$, and his winning strategy $\mathcal{S}_2$ on $(\mathcal{A}_2, \mathcal{B}_2)$, *simultaneously* in parallel. This is a well-defined strategy, since every $\mathcal{S}_i$ has the same pattern P. At the end of the game:

- for $i = j$, no board from $\mathcal{A}_i$ forms a matching pair with a board from $\mathcal{B}_j$, since **S** wins the sub-game $(\mathcal{A}_i, \mathcal{B}_i)$.
- for $i \neq j$, no board from $\mathcal{A}_i$ forms a matching pair with a board from $\mathcal{B}_j$, by assumption.

Therefore, no matching pair remains after round r, and so, **S** wins the game. The pattern for this strategy is P by construction. ◀

We observe that Lemma 5 can be generalized in two ways. Firstly, we could split into k sub-games instead of two. Secondly, we can weaken assumption 1 in the statement of the lemma, so that each of the patterns is a subsequence of some r-tuple $P = \{\exists, \forall\}^r$. This is because **S** can simply extend the strategy $\mathcal{S}_i$ with pattern P_i to a strategy $\mathcal{S}'_i$ with pattern P, where for every "missing" entry in the tuple P, **S** makes a dummy move on the corresponding side. We state a generalized version below without a proof.

▶ **Lemma 6** (Generalized Parallel Play Lemma). *Let $\mathcal{A}$ and $\mathcal{B}$ be two sets of pebbled structures, and let $r \in \mathbb{N}$. Let $P \in \{\exists, \forall\}^r$ be a sequence of quantifiers of length r. Suppose that $\mathcal{A}$ and $\mathcal{B}$ can be partitioned as $\mathcal{A} = \mathcal{A}_1 \sqcup \ldots \sqcup \mathcal{A}_k$ and $\mathcal{B} = \mathcal{B}_1 \sqcup \ldots \sqcup \mathcal{B}_k$ respectively, such that for all $1 \leq i \leq k$, **S** has a winning strategy $\mathcal{S}_i$ for the r_i-round MS game on $(\mathcal{A}_i, \mathcal{B}_i)$ (where $r_i \leq r$), satisfying the following conditions:*

1. *For all i, $\mathsf{pat}(\mathcal{S}_i)$ is a subsequence of P.*
2. *At the end of the sub-games, for $i \neq j$, there does not exist a board in $\mathcal{A}_i$ and a board in $\mathcal{B}_j$ forming a matching pair.*

*Then **S** wins the r-round MS game on $(\mathcal{A}, \mathcal{B})$ with pattern P.*

Note that Lemmas 5 and 6 can be applied in conjunction with Observation 2 as long as there is at least one structure remaining on either side, since a winning strategy (and therefore its corresponding pattern) is unaffected if some of the pebbled structures in the instance are deleted. Furthermore, in many cases, we can provide a strategy for **S** where condition 2 in Lemmas 5 and Lemma 6 will be automatically met after the first move, and therefore will continue to be satisfied at the end of the game. We shall use these two facts implicitly in the proofs that follow.

4 Linear Orders

As noted in Section 1, the results in this section are similar to those in [4,5], but somewhat more nuanced, leading ultimately to the *quantifier alternation theorems* (Theorems 12 and 13). Instead of the unwieldy function $g(\cdot)$ studied in those papers, we study the simpler function $q(\cdot)$, which, given an integer ℓ, returns the minimum number of quantifiers needed to separate $L_{\leq \ell}$ from $L_{>\ell}$. A key result, not appreciated in [4,5], is that the number of quantifiers needed to separate two linear orders of different sizes never exceeds the quantifier rank needed by more than one (Theorem 11).

Let $r(\ell)$ (resp. $q(\ell)$) be the minimum QR (resp. QN) needed to separate $L_{\leq \ell}$ and $L_{>\ell}$. Let $q_\forall(\ell)$ (resp. $q_\exists(\ell)$) be the minimum number of quantifiers needed to separate $L_{\leq \ell}$ and $L_{>\ell}$ with a sentence whose prenex normal form starts with $\forall$ (resp. $\exists$). Note that $q(\ell) = \min(q_\forall(\ell), q_\exists(\ell))$. The values of $r(\ell)$ are well understood [17]:

▶ **Theorem 7** (Quantifier Rank, [17]). *For $\ell \geq 1$, we have $r(\ell) = 1 + \lfloor \log(\ell) \rfloor$.*

Since QR lower bounds QN, we have $r(\ell) \leq q(\ell)$ for all ℓ. On the other hand, for each $\ell > 0$, we will show that **S** can always separate $L_{\leq \ell}$ from $L_{>\ell}$ in a multi-structural game of at most $r(\ell) + 1$ rounds, which shows that $q(\ell) \leq r(\ell) + 1$.

For notational convenience, we denote by $\mathrm{MSL}_{\exists,r}(\ell)$ an r-round MS game on $(L_{\leq \ell}, L_{>\ell})$, in which **S** *must* play an existential first round move. We use $\mathrm{MSL}_{\forall,r}(\ell)$ analogously, where the first round move *must* be universal. Observe that, *a priori*, any such game may be winnable by either **S** or **D**. Since we are primarily interested in upper bounds, we restrict our attention only to **S**-winnable games. We call such games simply *winnable*.

4.1 The Closest-to-Midpoint with Alternation Strategy

In this section, we describe a divide-and-conquer recursive strategy for **S** to play winnable game instances $\mathrm{MSL}_{Q,r}(\ell)$. This strategy will give us upper bounds on $q_\exists(\ell)$ and $q_\forall(\ell)$, which we will then relate to $r(\ell)$.

We define the *closest-to-midpoint* of a linear order $L[x, y]$ as the element halfway between the elements corresponding to x and y if $L[x, y]$ has even length, or the element just left of center if $L[x, y]$ has odd length.

The **S**-winning strategy is called *Closest-to-Midpoint with Alternation* (CMA). The pattern for this strategy will alternate between $\exists$ and $\forall$, splitting each game recursively into two smaller sub-games that can be played in parallel using Lemma 5. In these sub-games, placed pebbles will take on the roles of min and max. **S** continues in this way until the sub-games are on linear orders of length 2 or less, at which point he can win them easily.

The idea is for **S** to obey the following two rules throughout, except possibly the last three rounds:

- **S** starts on his designated side (determined by Q), and then alternates in every round;
- on every board, **S** plays on the closest-to-midpoint of a linear order $L[x, y]$, chosen carefully to ensure he essentially "halves" the length of the instance every round.

Note that one consequence of the second point above is that **S** will *never* play on max.

Before getting to a formal description of the strategy, let us illustrate the main idea through a worked example. Consider the (winnable) game $\mathrm{MSL}_{\exists,4}(5)$. In round 1, **S** plays on the closest-to-midpoint of all boards in $L_{\leq 5}$ (by the two conditions in the CMA strategy). Before **D**'s response, we reach the position shown in Figure 1.

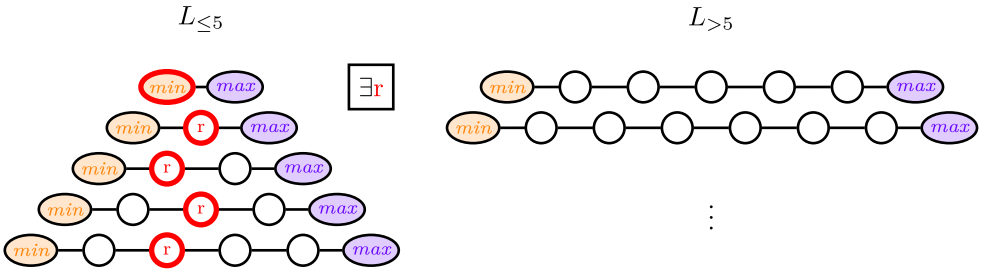


Figure 1 The position after **S**'s round 1 move in the game $\mathrm{MSL}_{\exists,4}(5)$. The pebble r is on the closest-to-midpoint of every board on the left.

Now assume **D** responds obliviously. We can first use Observation 2 to discard all boards on the right with r on max. By virtue of **S**'s first move, every board $\langle L \mid a_1 \rangle$ on the left satisfies *both* $L[\mathsf{min}, \mathrm{r}] \leq 2$, and $L[\mathrm{r}, \mathsf{max}] \leq 3$. Now consider any board $\langle L' \mid a'_1 \rangle$ on the right. Note that either $L'[\mathsf{min}, \mathrm{r}] > 2$, or $L'[\mathrm{r}, \mathsf{max}] > 3$. Partition the right side as $\mathcal{B}_1 \sqcup \mathcal{B}_2$, where every $\langle L' \mid a'_1 \rangle \in \mathcal{B}_1$ satisfies $L'[\mathsf{min}, \mathrm{r}] > 2$, and every $\langle L' \mid a'_1 \rangle \in \mathcal{B}_2$ satisfies $L'[\mathrm{r}, \mathsf{max}] > 3$.

In round 2, **S** makes a universal move (by the first condition in the CMA strategy). In all boards in $\mathcal{B}_1$, he plays pebble b on the closest-to-midpoint of $L'[\mathsf{min}, \mathrm{r}]$; similarly, in all boards in $\mathcal{B}_2$, he plays pebble b on the closest-to-midpoint of $L'[\mathrm{r}, \mathsf{max}]$. Note that in either case, **S** plays b on an element which is not on r, min, or max.

After **D** responds obliviously, we can use Observation 2 to discard all boards on the left where b is on min, max, or r. Since in particular this discards all boards on the left with r on min, we can again use Observation 2 to discard all boards from the right which have r on min. Every remaining board in $\mathcal{B}_1$ (resp. $\mathcal{B}_2$) corresponds to the isomorphism class $\mathsf{min} < \mathrm{b} < \mathrm{r} < \mathsf{max}$ (resp. $\mathsf{min} < \mathrm{r} < \mathrm{b} < \mathsf{max}$). The remaining boards on the left also correspond to exactly one of those classes. Partition the left side as $\mathcal{A}_1 \sqcup \mathcal{A}_2$ accordingly.

Now, because of this difference in isomorphism classes, we will never obtain a matching pair from $\mathcal{A}_1$ and $\mathcal{B}_2$ (or from $\mathcal{A}_2$ and $\mathcal{B}_1$). Furthermore, for the rest of the game, **S** will *only* play inside $L[\mathsf{min}, \mathrm{r}]$ on all boards in $\mathcal{A}_1$ and $\mathcal{B}_1$, and inside $L[\mathrm{r}, \mathsf{max}]$ on all boards in $\mathcal{A}_2$ and $\mathcal{B}_2$. Suppose, in response to such a move on $\mathcal{A}_1$, **D** plays outside the range $L[\mathsf{min}, \mathrm{r}]$ on a board from $\mathcal{B}_1$; the resulting board cannot form a partial match with any board from $\mathcal{A}_1$ (since there is a discrepancy with r), or with any board from $\mathcal{A}_2$ (as observed already). Therefore, this board from $\mathcal{B}_1$ can be discarded using Observation 2. A similar argument applies if **D** ever responds outside the corresponding range in $\mathcal{B}_2$, $\mathcal{A}_1$, or $\mathcal{A}_2$.

It follows that the sub-game $(\mathcal{A}_1, \mathcal{B}_1)$ (resp. $(\mathcal{A}_2, \mathcal{B}_2)$) corresponds *exactly* to the game $\mathrm{MSL}_{\forall,3}(2)$ (resp. $\mathrm{MSL}_{\forall,3}(3)$) where **S** has already made his first move using the CMA strategy by playing a universal move on the closest-to-midpoints of the (relevant) linear orders. Since **S** will alternate sides throughout, the patterns for both sub-game strategies will be the same.

We can now apply Lemma 5. Observe that the lengths of the instances in the sub-games have been roughly halved, at the cost of a single move. The game then proceeds as shown in Figure 2. The leaves of the tree correspond to base cases (analyzed in Section 4.2). The pattern of the strategy is preserved along all branches.

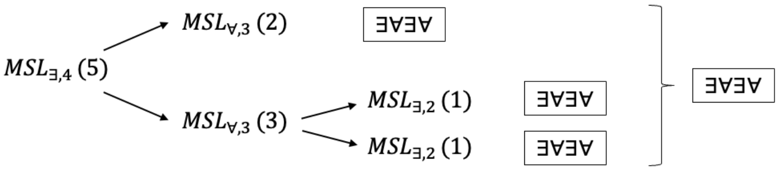


Figure 2 The $\mathrm{MSL}_{\exists,4}(5)$ game tree. Each leaf is decorated with the associated quantifier prefix. All paths can be played in parallel using Lemma 6 using the pattern $(\exists, \forall, \exists, \forall)$.

4.2 Formalizing the Strategy

The first step in formalizing the CMA strategy for **S** is to define four base cases, which we shall call *irreducible* games. We assert the following (see Appendix A in [2]).

1. $\mathrm{MSL}_{\forall,1}(1)$ is winnable with the pattern $(\forall)$.
2. $\mathrm{MSL}_{\exists,2}(1)$ is winnable with the pattern $(\exists, \forall)$.
3. $\mathrm{MSL}_{\forall,2}(2)$ is winnable with the pattern $(\forall, \forall)$.
4. $\mathrm{MSL}_{\forall,3}(2)$ is winnable with the pattern $(\forall, \exists, \forall)$.

The game $\mathrm{MSL}_{\exists,1}(1)$ is not winnable and hence not considered.

We now give a formalization of the inductive step. For a given quantifier $Q \in \{\exists, \forall\}$ and its complementary quantifier $\bar{Q}$, consider the game $\mathrm{MSL}_{Q,k}(\ell)$. Note that if **S** employs the CMA strategy the game splits into the two sub-games $\mathrm{MSL}_{\bar{Q},k-1}(\ell')$ and $\mathrm{MSL}_{\bar{Q},k-1}(\ell'')$. We designate this split as:

$$\mathrm{MSL}_{Q,k}(\ell) \to \mathrm{MSL}_{\bar{Q},k-1}(\ell') \oplus \mathrm{MSL}_{\bar{Q},k-1}(\ell'').$$

We will show in the proof of Lemma 9 that these sub-games can be played recursively, in parallel. When **S** reaches an irreducible sub-game, he plays the winning patterns asserted above. We claim the following about the rules for splitting. The proof is in [2].

▷ Claim 8 (Splitting Rules). For $k \geq 3$, we have:

$$\begin{aligned}
&\text{(i) } \mathrm{MSL}_{\exists,k}(2\ell) \to \mathrm{MSL}_{\forall,k-1}(\ell) \oplus \mathrm{MSL}_{\forall,k-1}(\ell), && \ell \geq 1\\
&\text{(ii) } \mathrm{MSL}_{\exists,k}(2\ell+1) \to \mathrm{MSL}_{\forall,k-1}(\ell) \oplus \mathrm{MSL}_{\forall,k-1}(\ell+1), && \ell \geq 1\\
&\text{(iii) } \mathrm{MSL}_{\forall,k}(2\ell) \to \mathrm{MSL}_{\exists,k-1}(\ell) \oplus \mathrm{MSL}_{\exists,k-1}(\ell-1), && \ell \geq 2\\
&\text{(iv) } \mathrm{MSL}_{\forall,k}(2\ell+1) \to \mathrm{MSL}_{\exists,k-1}(\ell) \oplus \mathrm{MSL}_{\exists,k-1}(\ell), && \ell \geq 1
\end{aligned} \tag{2}$$

Of course, the CMA strategy starts out seemingly promisingly, splitting with both initial sub-games starting on the same side; we must ensure that the strategy continues to be *well-defined*, i.e., this continues throughout the recursion stack, especially since the sub-games can have different lengths. We show this in Lemma 9, whose proof is in [2].

▶ **Lemma 9.** *The* CMA *strategy is well-specified. Moreover, for $k \geq 3$, if $MSL_{Q,k}(\ell) \to MSL_{\bar{Q},k-1}(\ell_1) \oplus MSL_{\bar{Q},k-1}(\ell_2)$ with $\ell_1 \geq \ell_2$, then the pattern of* **S***'s winning strategy for $MSL_{Q,k}(\ell)$ is Q concatenated with the pattern for the winning strategy for $MSL_{\bar{Q},k-1}(\ell_1)$.*

4.3 Bounding and Characterizing the Pattern

Define $q^*_\exists(\ell)$ (resp. $q^*_\forall(\ell)$) as the minimum $r \in \mathbb{N}$ such that **S** wins the game $\mathrm{MSL}_{\exists,r}(\ell)$ (resp. $\mathrm{MSL}_{\forall,r}(\ell)$) using the CMA strategy. Of course, we must have $q_\exists(\ell) \leq q^*_\exists(\ell)$ and $q_\forall(\ell) \leq q^*_\forall(\ell)$. Let $q^*(\ell) = \min(q^*_\exists(\ell), q^*_\forall(\ell))$. The following lemma (whose proof is omitted) follows from the complete description of the strategy from Section 4.2.

▶ **Lemma 10.** *We have $q^*_\forall(1) = 1$, $q^*_\exists(1) = 2$, and $q^*_\forall(2) = 2$. Also:*

$$\begin{aligned}
&q^*_\exists(2\ell) = q^*_\forall(\ell) + 1 \quad \textit{for } \ell \geq 1, && q^*_\exists(2\ell+1) = q^*_\forall(\ell+1) + 1 \quad \textit{for } \ell \geq 1,\\
&q^*_\forall(2\ell) = q^*_\exists(\ell) + 1 \quad \textit{for } \ell \geq 2, && q^*_\forall(2\ell+1) = q^*_\exists(\ell) + 1 \quad \textit{for } \ell \geq 1.
\end{aligned}$$

From Lemma 10 it is possible to recursively compute $q^*_\forall(\ell)$ and $q^*_\exists(\ell)$, and therefore $q^*(\ell)$ for all values of $\ell \geq 1$. These values are provided for $\ell \leq 127$ in Table 1.

We now state and prove the main result of this section.

▶ **Theorem 11.** *For all $\ell \geq 1$, we have:*

$$r(\ell) \leq q(\ell) \leq r(\ell) + 1.$$

Proof. The first inequality, $r(\ell) \leq q(\ell)$, is obvious. For the second, we will show that $q^*_\exists(\ell)$ and $q^*_\forall(\ell)$ are both bounded above by $r(\ell) + 1$ (and since $q(\ell) \leq q^*(\ell) = \min(q^*_\exists(\ell), q^*_\forall(\ell))$, so too for $q(\ell)$). Lemma 10 shows that the assertion is true for $\ell \leq 2$. Now, it can be shown recursively (see, e.g., Appendix A in [2]) that $q^*_\forall(2^k) = q^*_\exists(2^k) = k+1$ for $k \geq 1$. By Theorem 7, we also know that $r(2^k) = k+1$ for $k \geq 1$. So the three functions, $r(\cdot)$, $q^*_\forall(\cdot)$, and $q^*_\exists(\cdot)$, all equal each other at successive powers of two, and increase by one between these successive powers. Since all three functions are monotonic, they differ from one another by at most one. Therefore, we have $q^*_\exists(\ell) \leq r(\ell) + 1$ and $q^*_\forall(\ell) \leq r(\ell) + 1$. ◀

Table 1 Values of $q^*_\forall(\ell), q^*_\exists(\ell), q^*(\ell)$ and $r(\ell)$ for $1 \leq \ell \leq 127$.

ℓ	$q^*_\forall(\ell)$	$q^*_\exists(\ell)$	$q^*(\ell)$	$r(\ell)$
1	1	2	1	1
2	2	2	2	2
3	3	3	3	2
4	3	3	3	3
5	3	4	3	3
6-7	4	4	4	3
8-9	4	4	4	4
10	5	4	4	4
11-15	5	5	5	4
16-18	5	5	5	5
19-21	5	6	5	5
22-31	6	6	6	5
32-37	6	6	6	6
38-42	7	6	6	6
43-63	7	7	7	6
64-75	7	7	7	7
76-85	7	8	7	7
86-127	8	8	8	7

We wrap up this section with two results that will be useful in Section 5. For their proofs, we refer the reader to the full version [2].

▶ **Theorem 12** (Alternation Theorem, Smaller vs. Larger). *For every $\ell \geq 1$, there is a separating sentence σ_ℓ for $(L_{\leq \ell}, L_{>\ell})$ with $q^*(\ell)$ quantifiers (and so at most $\log(\ell) + 2$ quantifiers), such that the quantifier prefix of σ_ℓ strictly alternates and ends with a $\forall$.*

▶ **Theorem 13** (Alternation Theorem, One vs. All). *For every $\ell \geq 1$, there is a sentence φ_ℓ separating L_ℓ from all other linear orders having an alternating quantifier prefix (ending with a $\forall$) and consisting of $q^*(\ell) + 2$ quantifiers (and so at most $\log(\ell) + 4$ quantifiers).*

5 Strings

In this section, we pursue our main objective: string separation results, in order to characterize the complexity of Boolean functions. We would like to bound the number of quantifiers required for these separations as a function of both the length n of the strings, as well as the sizes of the sets.

In general, we would like to separate a set of n-bit strings from the set of all other n-bit strings; recall from Section 1 that we can think of this as separating the 1 instances from the 0 instances for a Boolean function on n-bit inputs. To do so, we first need to develop a basic technique for *distinguishing* one string from another.

▶ **Proposition 14** (One vs. One). ***Upper Bound:*** *For every pair w, w' of n-bit strings such that $w \neq w'$, there is a sentence $\varphi_{w,w'}$ with $\log(n) + 6$ quantifiers separating $(\{w\}, \{w'\})$. This sentence $\varphi_{w,w'}$ (in prenex form) has an alternating quantifier prefix ending with $\forall$.*
Lower Bound: *For all sufficiently large n, there exist two n-bit strings w, w', such that separating them requires $\lfloor \log(n) \rfloor$ quantifiers.*

Proof.

Upper Bound. Let $w, w' \in \{0,1\}^n$ be any two distinct n-bit strings. There is an index $i \in [n]$ such that $w_i \neq w'_i$. Let $\mathcal{A} = \{w\}$ and $\mathcal{B} = \{w'\}$. We will show that **S** wins the MS game on $(\mathcal{A}, \mathcal{B})$ in $\log(n) + 6$ rounds.

In round 1, **S** plays pebble r on the $\mathcal{A}$ side, on the element w_i in w, creating the pebbled string $\langle w \mid w_i \rangle$. Assume **D** responds obliviously on the $\mathcal{B}$ side. We can now immediately use Observation 2 to discard the resulting pebbled string $\langle w' \mid w'_i \rangle \in \mathcal{B}$, where the pebble r is on the element w'_i. Every remaining board in $\mathcal{B}$ is of the form $\langle w' \mid w'_j \rangle$, for $j \neq i$. Note that the substring $w'[1, j]$ has length j, which is different from i, the length of the substring $w[1, i]$ of $w \in \mathcal{A}$. So now, **S** can simply play the strategy from Theorem 13 to separate a linear order of length i from all other linear orders, which he wins in $\log(n) + 4$ rounds with an alternating pattern. This gives us the desired result, after at most one more dummy move to preserve alternation.

Lower Bound. Let $\ell = 2^k + 2$ for $k > 1$, and let $w = 0^{2^{k-1}}10 0^{2^{k-1}}$ and $w' = 0^{2^{k-1}}010^{2^{k-1}}$. If **S** plays entirely on one side of the respective 1s then he is effectively playing the MS game on $(L_{2^{k-1}}, L_{2^{k-1}-1})$. By Theorem 7, we have $r(2^{k-1}) = k = \lfloor \log(\ell) \rfloor$. Since QR lower bounds QN, the MS game played in this fashion requires at least $\lfloor \log(\ell) \rfloor$ rounds to win.

Now suppose that instead of playing entirely on the same side of the respective 1s, **S** plays on both sides of a 1 and/or on the 1 during these $\lfloor \log(\ell) \rfloor$ rounds. In this case, **D** can play obliviously to the left of the 1 when **S** plays to the left of the 1, obliviously to the right of the 1 when **S** plays to the right of the 1, and on the 1 whenever **S** plays on the 1, thereby keeping matching pairs simultaneously on both sides. The lower bound follows. ◀

We also need another helpful lemma, whose proof is in Appendix B of [2].

▶ **Lemma 15.** *Let $f \colon \mathbb{N} \to \mathbb{N}$ be a function satisfying $\lim_{n\to\infty} f(n) = \infty$, and let $t \geq 2$ be any integer. Then, for some number $N(t)$ depending on t, for all $n \geq N(t)$, we have $\lceil \log_t(f(n)) \rceil! \geq f(n)$.*

We now start with our string separation problems. The first problem we will consider will be when there is a single n-bit string in $\mathcal{A}$, and the $2^n - 1$ remaining n-bit strings in $\mathcal{B}$. Note that this corresponds to our Boolean function of interest being an indicator function.

▶ **Theorem 16** (One vs. All). *For all n, and for every $\varepsilon > 0$, it is possible to separate each n-bit string from all other n-bit strings by a sentence with $(1+\varepsilon)\log(n) + O_\varepsilon(1)$ quantifiers. This sentence (in prenex form) starts with a $\forall$, then has at most $\varepsilon \log(n) + 1$ occurrences of $\exists$, and then ends with an alternating quantifier prefix of length at most $\log(n) + O_\varepsilon(1)$.*

Proof Sketch (see [2] for full proof). Fix any $\varepsilon > 0$, and fix any integer $t \geq 2^{1/\varepsilon}$. By Lemma 15, we know there is some integer $N(t)$, such that for all $n \geq N(t)$, $\lceil \log_t(n) \rceil! \geq n$. For any such n, fix an arbitrary $w \in \{0,1\}^n$, and let $\mathcal{A} = \{w\}$, and $\mathcal{B} = \{0,1\}^n - \{w\}$.

Consider the MS game on $(\mathcal{A}, \mathcal{B})$. Every $w' \in \mathcal{B}$ differs from w in at least one bit. In round 1, **S** plays a universal move, placing a pebble on each $w' \in \mathcal{B}$ on an index that disagrees with w at that index. Assume **D** responds obliviously, so that there are n resulting pebbled strings in $\mathcal{A}$. For the next $\lceil \log_t(n) \rceil$ rounds, **S** plays only existential moves, placing the $\lceil \log_t(n) \rceil$ pebbles in distinct permutations on the n strings in $\mathcal{A}$, creating n distinct isomorphism classes[2] by Lemma 15. Once we discard structures from the two sides using Observation 2,

[2] An *isomorphism class* is a maximal set of partially isomorphic pebbled structures.

we are now left with n isomorphism classes, each of them defining a **one-vs-all** sub-game; in each of these sub-games, the round 1 pebble is placed at a different index in the single string on the left from any string on the right. Therefore, **S** can view this as a game simply about lengths, and can employ any **one-vs-all** linear order strategy. The entire game therefore reduces to n parallel instances of **one-vs-all** sub-games on linear orders.

By Lemma 6 and Theorem 13, **S** can now win these parallel games in $\log(n) + 4$ further moves. Together with the initial universal move and the preprocessing moves, the total number of rounds is:

$$\lceil \log_t(n) \rceil + \log(n) + 5 \leq \frac{\log(n)}{\log(t)} + \log(n) + 6 = \log(n)\left(1 + \frac{1}{\log(t)}\right) + 6 \leq (1+\varepsilon)\log(n) + 6.$$

Note that $N(t)$ depends only on t, which in turn depends only on ε. So when $n < N(t)$, the number of quantifiers can be absorbed directly into the $O_\varepsilon(1)$ additive term. ◀

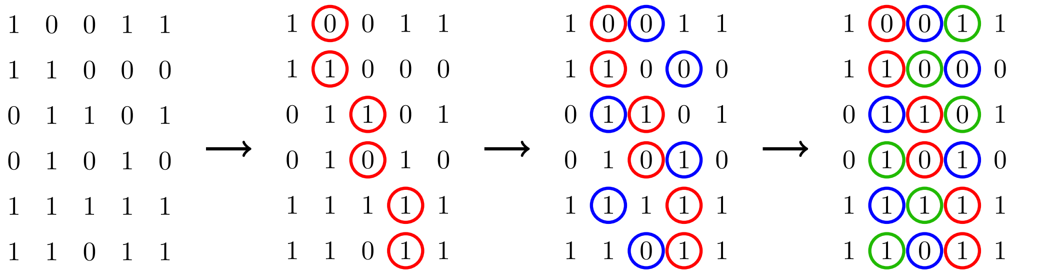


Figure 3 Illustration of the technique used by **S** to partition a set of structures into isomorphism classes. Here **S** plays three pebble moves to break the set of six strings into distinct isomorphism classes: r < b < g, r < g < b, and so on. Note that three pebbling moves suffice to give each string its own isomorphism class since $3! = 6$.

The next problem we will consider has polynomially many n-bit strings in $\mathcal{A}$, and the remaining n-bit strings in $\mathcal{B}$. This will correspond to our Boolean function of interest being a *sparse* function. Note that this immediately implies Theorem B in Section 1.

▶ **Theorem 17** (Polynomially Many vs. All). *Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function that satisfies $\lim_{n\to\infty} f(n) = \infty$ and $f(n) = O(n^k)$ for some constant k. Then, for all n, and for every $\varepsilon > 0$, it is possible to separate each set of $f(n)$ n-bit strings from all other n-bit strings by a sentence with $(1+\varepsilon)\log(n) + O_{k,\varepsilon}(1)$ quantifiers.*

Proof Sketch (see [2] for full proof). Assume $n > 2$, and pick a sufficiently large constant k such that $f(n) \leq n^k$ for all n. Next, pick $\varepsilon > 0$. Let $t \geq 4$ be a large enough integer so that $t \geq 2^{2k/\varepsilon}$. By Lemma 15, we know there is some integer $N(t)$, such that for all $n \geq N(t)$, we have $\lceil \log_t(f(n)) \rceil! \geq f(n)$. **S** once again plays $\lceil \log_t(f(n)) \rceil$ existential moves, separating the $f(n)$ strings in $\mathcal{A}$ into distinct isomorphism classes by using different permutations. Now, as in the proof of Theorem 16, **S** has reduced the games to $f(n)$ parallel **one-vs-all** string separation instances. So now, using Theorem 16, he can win these instances in parallel, using $(1 + \varepsilon/2)\log(n) + 6$ quantifiers for all $n \geq \max(N(t), N'(\varepsilon))$, for some $N'(\varepsilon)$ depending only on ε. The total number of rounds used by **S** is:

$$\begin{aligned}\lceil \log_t(f(n)) \rceil + (1+\varepsilon/2)\log(n) + 6 &\leq (1+\varepsilon/2)\log(n) + k\log_t(n) + 7\\ &= (1+\varepsilon/2)\log(n) + \frac{k\log(n)}{2k/\varepsilon} + 7\\ &\leq (1+\varepsilon)\log(n) + 7.\end{aligned}$$

Again, $N(t)$ depends only on t, which depends only on k and ε, whereas $N'(\varepsilon)$ depends only on ε. So when $n < \max(N(t), N'(\varepsilon))$, the number of quantifiers can be absorbed into an additive term that depends only on k and ε, giving us the $O_{k,\varepsilon}(1)$ term. ◀

Our final results concern separating arbitrary sets of n-bit strings from their complements. As discussed in Section 1, this corresponds exactly to defining arbitrary Boolean functions. Note that this will immediately imply Theorem A in Section 1.

▶ **Theorem 18** (Arbitrary vs. Arbitrary – *Upper Bound*). *For all n, and for every $\varepsilon > 0$, any two disjoint sets of n-bit strings are separable by a sentence with $(1+\varepsilon)\frac{n}{\log(n)} + O_\varepsilon(1)$ quantifiers.*

Proof Sketch (see [2] for full proof). We first observe that for any real number $r > 2$, **S** can play $m := \lceil n/\log_r(n)\rceil$ preprocessing existential moves, putting different permutations of these m pebbles on the strings in $\mathcal{A}$ (i.e., the left side). A Stirling's approximation argument similar to Lemma 15 shows that there is some $N(r)$ such that for all $n \geq N(r)$, this number m of preprocessing moves suffices to give each string in $\mathcal{A}$ its own isomorphism class. Note that once this is done, **S** has partitioned the original instance into $|\mathcal{A}|$ disjoint instances of **one-vs-all** games.

Now, given $\varepsilon > 0$, we first choose $r > 2$ small enough that $\log(r) < 1 + \varepsilon/2$. **S** now plays the preprocessing existential moves as described above to obtain $|\mathcal{A}|$ parallel **one-vs-all** instances. Now, by Theorem 16, he can win these instances in parallel using Lemma 6, using $(1+\varepsilon/2)\log(n) + 6$ rounds for all $n \geq \max(N(r), N'(\varepsilon))$, for some $N'(\varepsilon)$ depending only on ε. The total number of rounds needed, therefore, is:

$$\begin{aligned} m + (1+\varepsilon/2)\log(n) + 6 &\leq \frac{n}{\log(n)} \cdot \log(r) + \left(1 + \frac{\varepsilon}{2}\right)\log(n) + 7 \\ &\leq \left(1 + \frac{\varepsilon}{2}\right)\left(\frac{n}{\log(n)} + \log(n)\right) + 7 \\ &< (1+\varepsilon)\frac{n}{\log(n)} + 7 \end{aligned}$$

for all $n \geq \max(N(r), N'(\varepsilon), N''(\varepsilon))$, where for all $n \geq N''(\varepsilon)$, we have $(1+\varepsilon/2)\log(n) < (\varepsilon/2)\frac{n}{\log(n)}$. Since each of $N(r)$, $N'(\varepsilon)$, and $N''(\varepsilon)$ depends only on ε, the number of quantifiers for smaller n is absorbed into the $O_\varepsilon(1)$ term. ◀

Remarkably, we cannot improve the upper bound in Theorem 18 by any significant amount. The following proposition establishes this by means of a counting argument, also showing that Theorem A is tight.

▶ **Theorem 19** (Arbitrary vs. Arbitrary – *Lower Bound*). *For all sufficiently large n, there is a nonempty set of n-bit strings, $\mathcal{A} \subsetneq \{0,1\}^n$, such that every separating sentence φ for $(\mathcal{A}, \{0,1\}^n - \mathcal{A})$ must have at least $n/\log(n)$ quantifiers.*

Proof Sketch (see [2] for full proof). If we require k quantifiers to separate any instance on n-bit strings (for sufficiently large n), we can start by counting the number of pairwise inequivalent sentences that can be written with k quantifiers. Such a sentence has a quantifier prefix of length at most k ($\leq 2^{k+1}$ possibilities), followed by a quantifier-free part, which is a disjunction of types (2^{k^k} possibilities). This puts the total number of possible such formulas to be at most $2^k \cdot 2^{2^{k\log(k)}}$. We need this number to be at least $2^{2^n} - 2$, to account for all nonempty instances of the form $(\mathcal{A}, \{0,1\}^n - \mathcal{A})$, which require pairwise inequivalent sentences to separate. Solving this shows that we need $k \geq n/\log(n)$. ◀

6 Conclusions & Open Problems

We obtained nontrivial quantifier upper bounds with matching lower bounds (up to $(1+\varepsilon)$ factors) for a variety of string separation problems. All our upper bounds arise as a result of using the technique of parallel play.

Throughout this work, with very few exceptions, we used MS games to obtain *upper* bounds. It might seem unnecessary to exhibit upper bounds using game arguments, when it ordinarily suffices to exhibit separating sentences. However, the sentences implicitly arising from our game techniques are highly nontrivial to construct. In the case of QR, since taking disjunctions and conjunctions do not increase the quantifier rank, one can build up complex sentences out of simpler ones without paying any cost; we lose this convenience with QN, and therefore need more nuanced techniques, such as parallel play.

Natural directions to extend this work include the following:

1. It would be illuminating to understand the QN required to express particular string and graph properties. While our lower bound for the **one-vs-one** problem (Proposition 14) gave a pair of strings requiring $\log(n)$ quantifiers to separate, the counting argument in Proposition 19 does not exhibit a *specific* instance on n-bit strings that provably requires $n/\log(n)$ quantifiers to separate. Note that by (1), if we can find any property that requires $\omega(\log(n))$ quantifiers to capture, then that property lies outside of NL.
2. Is it possible to use our upper bound in Theorem 18 to obtain Lupanov's upper bound of $(1+\varepsilon)2^n/\log(n)$ on the minimum formula size needed separate two sets in propositional logic (or vice versa)?
3. It is known for ordered structures that with $O(\log n)$ quantifiers, one can express the BIT predicate, or equivalently, all standard arithmetic operations on elements of the universe [12]. In particular, with BIT, some properties that would otherwise require $\log(n)$ quantifiers can be expressed using $O(\log(n)/\log\log(n))$ quantifiers. Understanding the use of BIT and other numeric relations would be valuable.

References

1 Marco Carmosino, Ronald Fagin, Neil Immerman, Phokion Kolaitis, Jonathan Lenchner, and Rik Sengupta. Multi-structural games and beyond, 2023. doi:10.48550/arXiv.2301.13329.

2 Marco Carmosino, Ronald Fagin, Neil Immerman, Phokion Kolaitis, Jonathan Lenchner, and Rik Sengupta. On the number of quantifiers needed to define boolean functions, 2024. arXiv:2407.00688.

3 Andrzej Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta Mathematicae*, 49:129–141, 1961. doi:10.4064/fm-49-2-129-141.

4 Ronald Fagin, Jonathan Lenchner, Kenneth W. Regan, and Nikhil Vyas. Multi-structural games and number of quantifiers. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13, Rome, Italy, 2021. IEEE. doi:10.1109/LICS52264.2021.9470756.

5 Ronald Fagin, Jonathan Lenchner, Nikhil Vyas, and R. Ryan Williams. On the number of quantifiers as a complexity measure. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 48:1–48:14, Vienna, Austria, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.48.

6 Steven Fortune. A note on sparse complete sets. *SIAM Journal on Computing*, 8(3):431–433, 1979. doi:10.1137/0208034.

7 Roland Fraïssé. Sur quelques classifications des systèmes de relations. *Université d'Alger, Publications Scientifiques, Série A*, 1:35–182, 1954. doi:10.2307/2963939.

8 Martin Grohe and Nicole Schweikardt. The succinctness of first-order logic on linear orders. *Log. Methods Comput. Sci.*, 1(1), 2005. doi:10.2168/LMCS-1(1:6)2005.

9 L. Hella and K. Luosto. Game characterizations for the number of quantifiers. *Mathematical Structures in Computer Science*, pages 1–20, 2024.

10 Lauri Hella and Jouko Väänänen. The size of a formula as a measure of complexity. In Asa Hirvonen, Juha Kontinen, Roman Kossak, and Andrés Villaveces, editors, *Logic Without Borders: Essays on Set Theory, Model Theory, Philosophical Logic and Philosophy of Mathematics*, pages 193–214. De Gruyter, Berlin, München, Boston, 2015. doi:doi:10.1515/9781614516873.193.

11 Neil Immerman. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981. doi:10.1016/0022-0000(81)90039-8.

12 Neil Immerman. *Descriptive Complexity.* Springer, New York USA, 1999.

13 Oleg Lupanov. On a method of circuit synthesis. *Izvestia VUZ Radiofizika*, 1(1):120–140, 1958.

14 Oleg Lupanov. On the realization of functions of logical algebra by formulae of finite classes (formulae of limited depth). *Problems of Cybernetics*, 6(6):1–14, 1965. Upper bounds on sizes of formulas for all functions (English translation of Problemy Kibernetiki 6 (1961) 5-14.).

15 Stephen R. Mahaney. Sparse complete sets for NP: Solution of a conjecture of berman and hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982. doi:10.1016/0022-0000(82)90002-2.

16 John Riordan and C E Shannon. The number of two-terminal series-parallel networks. *Journal of Mathematics and Physics*, 21(1-4):83–93, 1942. doi:10.1002/sapm194221183.

17 Joseph G. Rosenstein. *Linear Orderings.* Academic Press, New York USA, 1982.

18 Harry Vinall-Smeeth. From quantifier depth to quantifier number: Separating structures with k variables, 2024. doi:10.48550/arXiv.2311.15885.

The Complexity of Simplifying ω-Automata Through the Alternating Cycle Decomposition

Antonio Casares
University of Warsaw, Poland

Corto Mascle
LaBRI, Université de Bordeaux, France

Abstract

In 2021, Casares, Colcombet and Fijalkow introduced the Alternating Cycle Decomposition (ACD), a structure used to define optimal transformations of Muller into parity automata and to obtain theoretical results about the possibility of relabelling automata with different acceptance conditions. In this work, we study the complexity of computing the ACD and its DAG-version, proving that this can be done in polynomial time for suitable representations of the acceptance condition of the Muller automaton. As corollaries, we obtain that we can decide typeness of Muller automata in polynomial time, as well as the parity index of the languages they recognise.

Furthermore, we show that we can minimise in polynomial time the number of colours (resp. Rabin pairs) defining a Muller (resp. Rabin) acceptance condition, but that these problems become NP-complete when taking into account the structure of an automaton using such a condition.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Omega-regular languages, Muller automata, Zielonka tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.35

Related Version *Full Version*: https://arxiv.org/abs/2401.03811 [13]

Funding Antonio Casares: Supported by the Polish National Science Centre (NCN) grant "Polynomial finite state computation" (2022/46/A/ST6/00072).

This document contains hyperlinks. Each occurrence of a notion is linked to its *definition*. On an electronic device, the reader can click on words or symbols (or just hover over them on some PDF readers) to see their definition.

1 Introduction

1.1 Context

Automata for the synthesis problem. Since the 60s, automata over infinite words have provided a fundamental tool to study problems related to the decidability of different logics [5, 38]. Recent focus has centered on the study of synthesis of controllers for reactive systems with the specification given in Linear Temporal Logic (LTL). The original automata-theoretic approach by Pnueli and Rosner [37] remains at the heart of the state-of-the-art LTL-synthesis tools [19, 29, 33, 35]. Their method consists in translating the LTL formula into a deterministic ω-automaton which is then used to build an infinite duration game; a winning strategy in this game provides a correct controller for the system.

Different acceptance conditions. There are different ways of specifying which runs of an automaton over infinite words are accepting. Generally, we label the transitions of the automaton with some output colours, and we then indicate which colours should be seen (or not) infinitely often. This can be expressed in a variety of ways, obtaining different acceptance conditions, such as parity, Rabin or Muller. The complexity of such acceptance

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 35; pp. 35:1–35:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

conditions is crucial in the performance of algorithms dealing with automata and games over infinite words. For instance, parity games can be solved in quasi-polynomial time [6] and parity games solvers are extremely performing in practice [24], while solving Rabin and Muller games is, respectively, NP-complete [18] and PSPACE-complete [22]. Moreover, many existing algorithms for solving these games are polynomial on the size of the game graph, and are exponential only on parameters from the acceptance condition: Muller games can be solved in time $\mathcal{O}(k^{5k}n^5)$ [6, Theorem 3.4], where n is the size of the game and k is the number of colours used by the acceptance condition, and Rabin games can be solved in time $\mathcal{O}(n^{r+3}rr!)$ [36, Theorem 7], where r is the number of Rabin pairs of the acceptance condition. Also, the emptiness check of Muller automata with the condition represented by a Boolean formula ϕ (Emerson-Lei condition) can be done in time $\mathcal{O}(2^k kn^2|\phi|)$ [2, Theorem 1].

Some important objectives are therefore: (1) transform an automaton $\mathcal{A}$ using a complex acceptance conditions into an automaton $\mathcal{B}$ using a simpler one, and (2) simplify as much as possible the acceptance condition used by an automaton $\mathcal{A}$ (without adding further states).

The Zielonka tree and Zielonka DAG. The Zielonka tree is an informative representation of Muller conditions, introduced for the study of strategy complexity in Muller games [42, 17]. Zielonka showed that we can use this structure to tell whether a Muller language can be expressed as a Rabin or a parity language [42, Section 5]. Moreover, it has been recently proved that the Zielonka tree provides minimal deterministic parity automata recognising a Muller condition [10, 31], and can thus be used to transform Muller automata using this condition into equivalent parity automata.

A natural alternative is to consider the more succinct DAG-version of this structure: the Zielonka DAG. Hunter and Dawar studied the complexity of building the Zielonka DAG from an explicit representation of a Muller condition, and the complexity of solving Muller games for these different representations [23]. Recently, Hugenroth showed that many decision problems concerning Muller automata become tractable when using the Zielonka DAG to represent the acceptance condition [21].

The ACD: Theoretical applications. In 2021, Casares, Colcombet and Fijalkow [9] proposed the Alternating Cycle Decomposition (ACD) as a generalisation of the Zielonka tree. The main motivation for the introduction of the ACD was to define optimal transformations of automata: given a Muller automaton $\mathcal{A}$, we can build using the ACD an equivalent parity automaton that is minimal amongst all parity automata obtained by duplicating states of $\mathcal{A}$ [10, Theorem 5.32]. Moreover, the ACD can be used to tell whether a Muller automaton can be relabelled with an acceptance condition of a simpler type [10, Section 6.1].

However, the works introducing the ACD [9, 10] are of theoretical nature, and no study of the cost of constructing it and performing the related transformations is presented.

The ACD: Practice. The transformations based on the ACD have been implemented in the tools Spot 2.10 [16] and Owl 21.0 [27], and are used in the LTL-synthesis tools `ltlsynt` [33] and `STRIX` [29, 32] (top-ranked in the SYNTCOMP competitions [24]). In the tool paper [12], these transformation are compared with the state-of-the-art methods to transform Emerson-Lei automata into parity ones. Surprisingly, the transformation based on the ACD does not only produce the smallest parity automata, but also outperforms all other existing paritizing methods in computation time.

In [12, Section 4], an algorithm computing the ACD is proposed. However, the focus is made in the handling of Boolean formulas to enhance the algorithm's performance in practice, but no theoretical analysis of its complexity is provided.

Simplification of acceptance conditions. As already mentioned, the complexity of the acceptance conditions play a crucial role in algorithms. One can simplify the acceptance condition of a Muller automaton by adding further states (and the optimal way of doing this is determined by the ACD [10]). However, in some cases this leads to an exponential blow-up in the number of states [28]. A natural question is therefore to try to simplify the acceptance condition while avoiding adding so many states. We consider two versions of this problem:

Typeness problem. Can we relabel the acceptance condition of a Muller automaton with one of a simpler type, such as Rabin, Streett or parity?

Minimisation of colours and Rabin pairs. Can we minimise the number of colours used by the acceptance condition (or, in the case of Rabin automata, the number of Rabin pairs)?

The ACD has proven fruitful for studying the typeness problem: just by inspecting the ACD of $\mathcal{A}$, we can tell whether we can relabel it with an equivalent Rabin, parity or Streett acceptance condition [10]. Also, it is a classical result that we can minimise in polynomial time the number of colours used by a parity automaton [7]. However, it was still unclear whether the ACD could help to minimise the number of colours of Muller conditions or the number of Rabin pairs of Rabin acceptance conditions, question that we tackle in this work.

The minimisation of colours in Muller automata has recently been studied by Schwarzová, Strejček and Major [39]. In their approach, they use heuristics to reduce the number of colours by applying QBF-solvers. The final acceptance condition is however not guaranteed to have a minimal number of colours. There have also been attempts to minimise the number of Rabin pairs of Rabin automata coming from the determinisation of Büchi automata [40]. Also, in their work about minimal history-deterministic Rabin automata, Casares, Colcombet and Lehtinen left open the question of the minimisation of Rabin pairs [11].

1.2 Contributions

1. **Computation of the ACD and the ACD-DAG.** We show that we can compute the ACD of a Muller automaton in polynomial time, provided that the Zielonka tree of its acceptance condition is given as input (Theorem 13). This shows that the computation of the ACD is not harder than that of the Zielonka tree, (partially) explaining the strikingly favourable experimental results from [12]. We also show that we can compute the DAG-version of the ACD in polynomial time if the acceptance condition of $\mathcal{A}$ is given colour-explicitly or by a Zielonka DAG (Theorem 15). The main technical challenge is to prove that the ACD has polynomial size in the size of the Zielonka tree.
2. **Deciding typeness in polynomial time.** Combining the previous contributions with the results from [10], we directly obtain that we can decide in polynomial time whether a Muller automaton can be relabelled with an equivalent parity, Rabin or Streett acceptance condition (Corollary 16). Moreover, we recover a result from Wilke and Yoo [41]: we can compute in polynomial time the parity index of the language of a Muller automaton.
3. **Minimisation of colours and Rabin pairs of acceptance conditions.** For a given Muller (resp. Rabin) language L, we show that we can minimise the number of colours (resp. Rabin pairs) needed to define L in polynomial time (Theorems 20 and 21). We also relate the minimisation of Rabin pairs to a subclass of interest of Boolean formulas, called generalised Horn formulas.
4. **Minimisation of colours and Rabin pairs over an automaton structure.** Given an automaton $\mathcal{A}$ using a Muller (resp. Rabin) acceptance condition, we show that the problem of minimising the number of colours (resp. Rabin pairs) to relabel $\mathcal{A}$ with an equivalent acceptance condition over its structure is NP-complete, and it remains NP-hard even

if the ACD is given as input (Theorems 26 and 27). This came as a surprise to us, as our first intuition was that the ACD would allow to lift the previous polynomial-time minimisation algorithms to ones which take into account the structure of the automaton.

We provide proof ideas for all the results, technical proofs can be found in the full version [13]. The full version also contains further contributions and discussions about the size of different representations of Muller conditions (summarised in Figure 3).

2 Preliminaries

2.1 Automata over infinite words and their acceptance conditions

Given a set Γ, we write 2^Γ_+ for the set of its non-empty subsets. For a word $w \in \Gamma^\omega$, we let $\mathsf{Inf}(w)$ be the set of letters appearing infinitely often in w.

Automata. A *(non-deterministic) automaton* is a tuple $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma, \mathsf{col}, W)$, where Q is a finite set of states, $q_{\mathsf{init}} \in Q$ is an initial state, Σ is an *input alphabet*, $\Delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, Γ is a finite set of *output colours*, $\mathsf{col}\colon \Delta \to \Gamma$ is a *colouring* of the transitions, and $W \subseteq \Gamma^\omega$ is a language over Γ. We call the tuple (col, W) the *acceptance condition* of $\mathcal{A}$. We write $q \xrightarrow{a} q'$ to denote a transition $e = (q, a, q') \in \Delta$, and $q \xrightarrow{a:c} q'$ to further indicate that $\mathsf{col}(e) = c$. We write $q \overset{w:u}{\rightsquigarrow} q'$ to represent the existence of a path from q to q' labelled with the input letters $w \in \Sigma^*$ and output colours $u \in \Gamma^*$.

Given an automaton $\mathcal{A}$ and a word $w \in \Sigma^\omega$, a *run over* w in $\mathcal{A}$ is a path $q_{\mathsf{init}} \xrightarrow{w_0:c_0} q_1 \xrightarrow{w_1:c_1} q_2 \xrightarrow{w_2:c_2} q_3 \xrightarrow{w_3:c_3} \cdots \in \Delta^\omega$. Such a run is *accepting* if $c_0c_1c_2 \cdots \in W$, and *rejecting* otherwise. A word $w \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if it admits an accepting run. The *language recognised* by an automaton $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^\omega \mid w \text{ is accepted by } \mathcal{A}\}$. Two automata over the same alphabet are *equivalent* if they recognise the same language. An automaton is *deterministic* (resp. *complete*) if for every $q \in Q$ and $a \in \Sigma$, there is at most (resp. at least) one transition $q \xrightarrow{a} q'$.

We underline that the colours of the acceptance of runs appear *over transitions*. For a discussion on the differences between transition and state-based automata, and arguments in favour of the former, we refer to [8, Chap. VI].

It is sometimes useful to let transitions carry multiple colours – for instance, this is the standard model in the HOA format [1]. For many results of this paper (those from Section 3), allowing or not multiple colours per edge does not make a difference; we could always take 2^Γ or Δ as new set of colours. This will however be relevant in Section 4.2. Also, the HOA format allows for multiple transitions between the same two states with the same input letter. These transitions can always be replaced by one carrying multiple colours (we refer to [11, Prop. 18] for details).

We let the *size of* $\mathcal{A}$ be $|\mathcal{A}| = |Q| + |\Sigma| + |\Delta| + |\Gamma|$. We note that this does not take into account the size of the representation of its acceptance condition, which can admit different forms (see page 7). When necessary, we will indicate the size of the representation of the acceptance condition separately.

Acceptance conditions. We now define the main classes of languages used by automata over infinite words as acceptance conditions. We let Γ stand for a finite set of colours.

Muller. We define the *Muller language* of a family $\mathcal{F} \subseteq 2^\Gamma_+$ of non-empty subsets of Γ as:

$$\mathsf{Muller}_\Gamma(\mathcal{F}) = \{w \in \Gamma^\omega \mid \mathsf{Inf}(w) \in \mathcal{F}\}.$$

We will often refer to sets in $\mathcal{F}$ as *accepting sets* and sets not in $\mathcal{F}$ as *rejecting sets*.

Rabin. A Rabin condition is represented by a family $\mathcal{R} = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ of *Rabin pairs*, where $\mathfrak{g}_j, \mathfrak{r}_j \subseteq \Gamma$. We define the *Rabin language* of a single Rabin pair $(\mathfrak{g}, \mathfrak{r})$ as

$$\mathsf{Rabin}_\Gamma((\mathfrak{g}, \mathfrak{r})) = \{w \in \Gamma^\omega \mid \mathsf{Inf}(w) \cap \mathfrak{g} \neq \emptyset \wedge \mathsf{Inf}(w) \cap \mathfrak{r} = \emptyset\},$$

and the Rabin language of a family of Rabin pairs $\mathcal{R}$ as: $\mathsf{Rabin}_\Gamma(\mathcal{R}) = \bigcup_{j=1}^{r} \mathsf{Rabin}_\Gamma((\mathfrak{g}_j, \mathfrak{r}_j))$.

Streett. The *Streett language* of a family $\mathcal{R} = \{(\mathfrak{g}_1, \mathfrak{r}_1), \dots, (\mathfrak{g}_r, \mathfrak{r}_r)\}$ of Rabin pairs is defined as the complement of its Rabin language:

$$\mathsf{Streett}_\Gamma(\mathcal{R}) = \Gamma^\omega \setminus \mathsf{Rabin}_\Gamma(\mathcal{R}).$$

Parity. We define the *parity language* over a finite alphabet $\Pi \subseteq \mathbb{N}$ as:

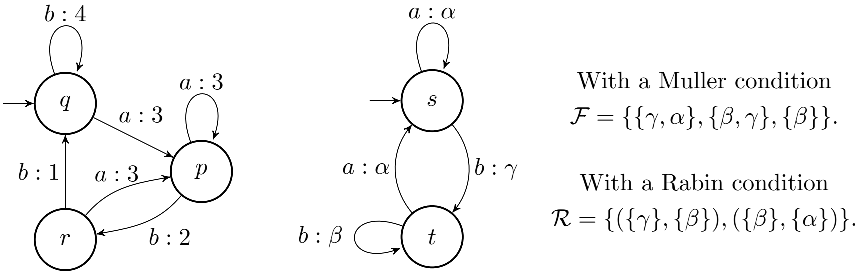
$$\mathsf{parity}_\Pi = \{w \in \Pi^\omega \mid \min \mathsf{Inf}(w) \text{ is even}\}.$$

We say that an automaton is a $\mathcal{C}$ *automaton*, for $\mathcal{C}$ one of the classes of languages above, if its acceptance condition uses a $\mathcal{C}$ language. We refer to the survey [3] for a more detailed account on different types of acceptance conditions.

▶ Remark 1. Muller languages are exactly the languages characterised by the set of letters seen infinitely often. They are also the languages recognised by deterministic Muller automata with one state.

We observe that parity languages are special cases of Rabin and Streett languages which are in turn special cases of Muller languages.

▶ **Example 2.** In Figure 1 we show different types of automata over the alphabet $\Sigma = \{a, b\}$ recognising the language of words that contain infinitely many bs and eventually do not encounter the factor abb.



With a Muller condition
$\mathcal{F} = \{\{\gamma, \alpha\}, \{\beta, \gamma\}, \{\beta\}\}$.

With a Rabin condition
$\mathcal{R} = \{(\{\gamma\}, \{\beta\}), (\{\beta\}, \{\alpha\})\}$.

Parity automaton $\mathcal{A}_1$.

Automaton $\mathcal{A}_2$ with equivalent Muller and Rabin conditions over it.

Figure 1 Different types of automata recognising the language $L = \Sigma^* b^\omega + \Sigma^*(a^+ b)^\omega$. (Note that the set of outputs that occur infinitely often in a run of $\mathcal{A}_2$ cannot be $\{\beta, \gamma\}$.)

The 8 classes of automata obtained by combining the 4 types of acceptance conditions above with deterministic and non-deterministic models are equally expressive [30, 34]. We call the class of languages that can be recognised by these automata *ω-regular languages*. The *parity index* of L is the minimal number k such that L can be recognised by a deterministic parity automaton using k output colours (which coincides with the minimal number of colours used by a Muller automaton recognising L [10, Proposition 6.14]).

Typeness. Let $\mathcal{A}_1 = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma_1, \mathsf{col}_1, W_1)$ be a deterministic automaton, and let $\mathcal{C}$ be a class of languages (potentially containing languages over different alphabets). We say that $\mathcal{A}_1$ can be *relabelled* with a $\mathcal{C}$-acceptance condition, or that $\mathcal{A}$ is $\mathcal{C}$*-type*, if there is $W_2 \subseteq \Gamma_2^\omega$, $W_2 \in \mathcal{C}$, and a colouring function $\mathsf{col}_2 \colon \Delta \to \Gamma_2$ such that $\mathcal{A}_1$ is equivalent to $\mathcal{A}_2 = (Q, q_{\mathsf{init}}, \Sigma, \Delta, \Gamma_2, \mathsf{col}_2, W_2)$. In this case, we say that (col_1, W_1) and (col_2, W_2) are *equivalent acceptance conditions over* $\mathcal{A}$.

Given a Muller automaton $\mathcal{A}$, we use the expression *deciding the typeness* of $\mathcal{A}$ for the problem of answering if $\mathcal{A}$ is Rabin type, Streett type, parity type, or none of those.[1]

▶ Remark 3. In this work, we only consider typeness for deterministic automata for simplicity. For non-deterministic models, typeness admits two non-equivalent definitions [26]: (1) the acceptance status of each individual infinite path coincide for both acceptance conditions, or (2) both automata recognise the same language.

▶ **Example 4.** The automaton $\mathcal{A}_2$ from Figure 1 is Rabin type, as we have labelled it with a Rabin acceptance condition that is equivalent over $\mathcal{A}$ to the Muller condition given by $\mathcal{F}$ (in this case, both conditions use the same set of colours $\Gamma = \{\alpha, \beta, \gamma\}$). However, we note that $\mathsf{Rabin}_\Gamma(\mathcal{R}) \neq \mathsf{Muller}_\Gamma(\mathcal{F})$, as $\gamma^\omega \in \mathsf{Rabin}_\Gamma(\mathcal{R})$, while $\gamma^\omega \notin \mathsf{Muller}_\Gamma(\mathcal{F})$. This is possible, as no infinite path in $\mathcal{A}_2$ is labelled by a word that differentiates both languages (such as γ^ω).

2.2 The Zielonka tree and the Zielonka DAG

We represent *trees* and *directed acyclic graphs* (DAGs) as pairs $T = (N, \preceq)$ with N a non-empty finite set of nodes and $\preceq$ the *ancestor relation* ($n \preceq n'$ meaning that n is above n'). An A*-labelled tree* (resp. A*-labelled DAG*) is a tree (resp. DAG) together with a labelling function $\nu \colon N \to A$. We write $|T|$ to denote the number of nodes of a tree T.

▶ **Definition 5** ([42]). *Let $\mathcal{F} \subseteq 2_+^\Gamma$ be a family of non-empty subsets of a finite set Γ. The* Zielonka tree *for $\mathcal{F}$ (over Γ),*[2] *denoted $\mathcal{Z}_\mathcal{F} = (N, \preceq, \nu : N \to 2_+^\Gamma)$ is a 2_+^Γ-labelled tree with nodes partitioned into* round nodes *and* square nodes*, $N = N_\bigcirc \sqcup N_\square$, such that:*

- *The root is labelled Γ.*
- *If a node is labelled $X \subseteq \Gamma$, with $X \in \mathcal{F}$, then it is a round node, and it has a child for each maximal non-empty subset $Y \subseteq X$ such that $Y \notin \mathcal{F}$, which is labelled Y.*
- *If a node is labelled $X \subseteq \Gamma$, with $X \notin \mathcal{F}$, then it is a square node, and it has a child for each maximal non-empty subset $Y \subseteq X$ such that $Y \in \mathcal{F}$, which is labelled Y.*

▶ **Example 6.** Let $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$ be the Muller condition of the automaton from Example 2, over the alphabet $\{\alpha, \beta, \gamma\}$. In Figure 2 we show the Zielonka tree of $\mathcal{F}$.

The *Zielonka DAG* of a family $\mathcal{F} \subseteq 2_+^\Gamma$ is the labelled directed acyclic graph $\mathcal{Z}\text{-}\mathsf{DAG}_\mathcal{F}$ obtained by merging the nodes of $\mathcal{Z}_\mathcal{F}$ with a common label. It inherits the partition into round and square nodes, with children of round nodes being square and vice-versa.

▶ **Lemma 7** (Implied by [21, Lemma 1]). *Given a 2_+^Γ-labelled tree T with a partition into round and square nodes, we can decide in polynomial time whether there is a family $\mathcal{F} \subseteq 2_+^\Gamma$ such that $T = \mathcal{Z}_\mathcal{F}$. In the affirmative case, this family is uniquely determined.*

[1] We could consider further classes of acceptance conditions such as Büchi, coBüchi, generalised Büchi, weak, etc... We refer to [10, Appendix A] for more details on these acceptance types.

[2] The definition of $\mathcal{Z}_\mathcal{F}$, as well as most subsequent definitions, do not only depend on $\mathcal{F}$ but also on the alphabet Γ. Although this dependence is important, we do not explicitly include it in the notations in order to lighten them, as most of the times the alphabet will be clear from the context.

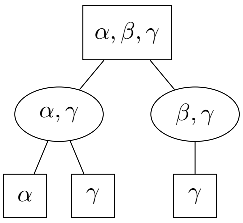


Figure 2 Zielonka tree $\mathcal{Z}_\mathcal{F}$ for $\mathcal{F} = \{\{\gamma, \alpha\}, \{\gamma, \beta\}, \{\beta\}\}$.

Representation of acceptance conditions. There is a wide variety of ways to represent a Muller language, including as its Zielonka tree, its Zielonka DAG, or *colour-explicitly*, that is, as a list of the subsets appearing in $\mathcal{F}$. In Figure 3 we summarise the relationship between these representations. We highlight that the Zielonka DAG can be built in polynomial time from both the Zielonka tree and from a colour-explicit representation of a Muller condition [23, Theorem 3.17]. The exponential-size separation between the Zielonka tree and colour-explicit representations, as well as explicit examples showing the gap between Zielonka trees and DAGs are original contributions.

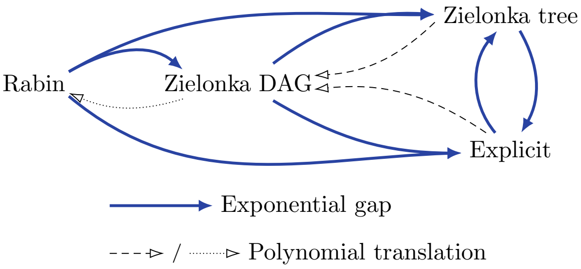


Figure 3 Comparison between the different representations of Muller conditions. A blue bold arrow from X to Y means that converting an X-representation into the form Y requires exponential time. A dashed arrow from X to Y means that a conversion can be computed in polynomial time. The dotted arrow indicates that the polynomial translation can only be applied on a fragment of X, as it is more expressive than Y.

2.3 The Alternating Cycle Decomposition

We now present the Alternating Cycle Decomposition and its DAG-version, following [10].

Let $\mathcal{A}$ be an automaton with Q and Δ as set of states and transitions, respectively. A *cycle* of $\mathcal{A}$ is a subset $\ell \subseteq \Delta$ such that there is a (not necessarily simple) path with the same starting and ending state such that the set of edges it visits is ℓ. The set of cycles of an automaton $\mathcal{A}$ is written $\mathcal{C}ycles(\mathcal{A})$. We will consider the set of cycles ordered by inclusion. If we see $\mathcal{A}$ as a graph, its cycles are the strongly connected subgraphs of that graph, and the maximal cycles are its strongly connected components (SCCs). Let $\mathcal{A}$ be a Muller automaton with acceptance condition $(\mathsf{col}, \mathsf{Muller}_\Gamma(\mathcal{F}))$. Given a cycle $\ell \in \mathcal{C}ycles(\mathcal{A})$, we say that ℓ is *accepting* (resp. *rejecting*) if $\mathsf{col}(\ell) \in \mathcal{F}$ (resp. $\mathsf{col}(\ell) \notin \mathcal{F}$).

▶ **Definition 8.** *Let $\ell_0 \in \mathcal{C}ycles(\mathcal{A})$ be a cycle. We define the* tree of alternating subcycles *of ℓ_0, denoted $\mathsf{AltTree}(\ell_0)$, as a $\mathcal{C}ycles(\mathcal{A})$-labelled tree with nodes partitioned into* round nodes *and* square nodes*, $N = N_\bigcirc \sqcup N_\square$, such that:*

- *The root is labelled* ℓ_0.
- *If a node is labelled* $\ell \in \mathcal{Cycles}(\mathcal{A})$*, and* ℓ *is an accepting cycle* ($\mathsf{col}(\ell) \in \mathcal{F}$)*, then it is a round node, and its children are labelled exactly with the maximal subcycles* $\ell' \subseteq \ell$ *such that* ℓ' *is rejecting* ($\mathsf{col}(\ell') \notin \mathcal{F}$).
- *If a node is labelled* $\ell \in \mathcal{Cycles}(\mathcal{A})$*, and* ℓ *is a rejecting cycle* ($\mathsf{col}(\ell) \notin \mathcal{F}$)*, then it is a square node, and its children are labelled exactly with the maximal subcycles* $\ell' \subseteq \ell$ *such that* ℓ' *is accepting* ($\mathsf{col}(\ell') \in \mathcal{F}$).

▶ **Definition 9** (Alternating cycle decomposition)**.** *Let* $\mathcal{A}$ *be a Muller automaton, and let* $\ell_1, \ell_2, \dots, \ell_k$ *be an enumeration of its maximal cycles. We define the* alternating cycle decomposition *of* $\mathcal{A}$ *to be the forest* $\mathcal{ACD}(\mathcal{A}) = \{\mathsf{AltTree}(\ell_1), \dots, \mathsf{AltTree}(\ell_k)\}$.

▶ Remark 10. The Zielonka tree can be seen as the special case of the alternating cycle decomposition for an automata with a single state.

As mentioned in the introduction, the ACD was introduced in order to build small parity automata from Muller ones: given the ACD of a Muller automaton $\mathcal{A}$, we can build in polynomial time an equivalent parity automaton $\mathcal{P}_{\mathcal{A}}^{\mathsf{ACD}}$ (called the *ACD-parity transform* of $\mathcal{A}$) of minimal size amongst all automata obtained from $\mathcal{A}$ by "duplication of states". See [10, Section 5.2] for a formal statement and further results.

▶ **Example 11.** Figure 4 contains the alternating cycle decomposition of the automata $\mathcal{A}_1$ and $\mathcal{A}_2$ from Figure 1. We represent their transitions as pairs $(q, a) \in Q \times \Sigma$. Since both automata are strongly connected, each ACD consists in a single tree, whose root is the whole set of transitions.

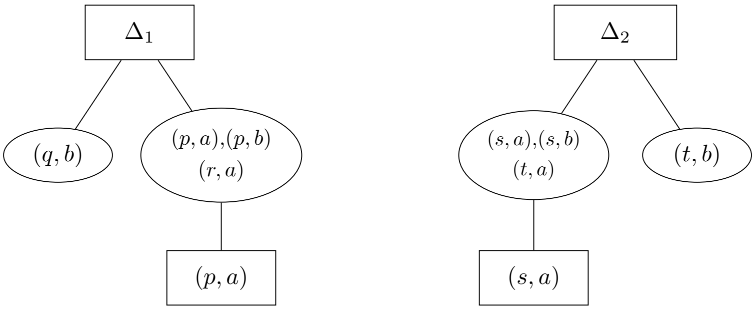


Figure 4 Alternating cycle decomposition of $\mathcal{A}_1$ (on the left) and $\mathcal{A}_2$ (on the right), from Figure 1.

The *ACD-DAG* of a Muller automaton $\mathcal{A}$, written $\mathcal{ACD}$-$\mathsf{DAG}(\mathcal{A})$, is the family of labelled DAGs obtained by merging nodes with the same label in the trees of $\mathcal{ACD}(\mathcal{A})$. It is useful for deciding the typeness and the parity index of $\mathcal{L}(\mathcal{A})$, as stated next.

▶ **Proposition 12** ([10, Section 6.1])**.** *Given a deterministic Muller automaton* $\mathcal{A}$ *and its ACD-DAG, we can decide the typeness of* $\mathcal{A}$ *and compute the parity index of* $\mathcal{L}(\mathcal{A})$ *in polynomial time.*

3 Computation of the Alternating Cycle Decomposition

We present in this section our central contribution: a polynomial-time algorithm to compute the alternating cycle decomposition of a Muller automaton with the acceptance condition given by a Zielonka tree (Theorem 13). This shows that the computation of the ACD is not harder than that of the Zielonka tree, (partially) explaining the strikingly performing experimental results from [12]. We also show that if the acceptance condition is represented as a Zielonka DAG, we can compute $\mathcal{ACD}$-DAG($\mathcal{A}$) in polynomial time (Theorem 15), from which we can derive decidability in polynomial time of typeness of Muller automata (Corollary 16).

3.1 Statements of the results

▶ **Theorem 13** (Computation of the ACD). *Given a Muller automaton $\mathcal{A}$ with acceptance condition represented by a Zielonka tree $\mathcal{Z}_{\mathcal{F}}$,*[3] *we can compute $\mathcal{ACD}(\mathcal{A})$ in polynomial time in $|\mathcal{A}| + |\mathcal{Z}_{\mathcal{F}}|$.*

As stated in the previous section, given the ACD of a Muller automaton $\mathcal{A}$, we can transform $\mathcal{A}$ in polynomial time into its ACD-parity-transform: a parity automaton equivalent to $\mathcal{A}$ that is minimal amongst parity automata obtained as a transformation of $\mathcal{A}$. The previous theorem implies that this can be done even if only the Zielonka tree of the acceptance condition of $\mathcal{A}$ is given as input, together with the automaton structure.[4]

▶ **Corollary 14.** *We can compute the ACD-parity-transform of a Muller automaton in polynomial time, if its acceptance condition is given by a Zielonka tree.*

▶ **Theorem 15** (Computation of the ACD-DAG). *Given a Muller automaton $\mathcal{A}$ with acceptance condition represented by a Zielonka DAG $\mathcal{Z}$-DAG$_{\mathcal{F}}$ (resp. colour-explicitly), we can compute $\mathcal{ACD}$-DAG($\mathcal{A}$) in polynomial time in $|\mathcal{A}| + |\mathcal{Z}\text{-DAG}_{\mathcal{F}}|$ (resp. $|\mathcal{A}| + |\mathcal{F}|$).*

Combining Theorem 15 with Propositions 12, we directly obtain that we can decide typeness of Muller automata and the parity index of their languages in polynomial time.

▶ **Corollary 16** (Polynomial-time decidability of typeness and parity index). *Given a deterministic Muller automaton $\mathcal{A}$ with its acceptance condition represented colour-explicitly, as a Zielonka tree, or as a Zielonka DAG, we can decide the typeness of $\mathcal{A}$, and determine the parity index of $\mathcal{L}(\mathcal{A})$, in polynomial time.*

The decidability of the parity index in polynomial time had already been obtained by Wilke and Yoo [41]. This result contrasts with the fact that deciding the parity index of a language represented by a deterministic Rabin or Streett automaton is NP-complete [25].

3.2 Main algorithm and complexity

Description of the algorithm. We describe an algorithm computing $\mathcal{ACD}$-DAG($\mathcal{A}$) from a Muller automaton $\mathcal{A}$. To obtain $\mathcal{ACD}(\mathcal{A})$, it suffices then to unfold this DAG. This algorithm builds the ACD-DAG in a top-down fashion: first, it computes the strongly connected components of $\mathcal{A}$ and initialises the root of each of the DAGs in $\mathcal{ACD}$-DAG($\mathcal{A}$). Then, it iteratively computes the children of the already found nodes as follows: Given a node n labelled ℓ (assume that ℓ is an accepting cycle), the algorithm goes through all square nodes m in the Zielonka DAG and for each of them computes the maximal sub-cycles of ℓ whose set of colours is included in the label of m, but not in those of any of its children. The algorithm then selects maximal cycles among those found, add them to $\mathcal{ACD}$-DAG($\mathcal{A}$) (if they do not already appear there) and sets them as children of n.

[3] Lemma 7 lets us check in polynomial time if a tree indeed is the Zielonka tree of a Muller condition.

[4] Also, we note that given $\mathcal{A}$ and its ACD, it is immediate to compute a Zielonka tree over the set of colours $\Gamma = \Delta$ defining an equivalent acceptance condition over $\mathcal{A}$.

Complexity analysis. We explain now how we obtain a polynomial upper bound on the complexity of the algorithms presented in the previous paragraph.

We first remark that we need to make at $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})|$ computations of the children of a node, as each node of the ACD-DAG is considered at most once by the algorithm. Therefore, to obtain Theorem 15 (computation of the ACD-DAG) we need to show that:

1. We can compute the children of a node in polynomial time in $|Q| + |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$, and
2. $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})|$ is polynomial in $|Q| + |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$.

To establish Theorem 13 (computation of the ACD), we remark that we can compute $\mathcal{ACD}(\mathcal{A})$ from $\mathcal{A}$ and $\mathcal{Z}_{\mathcal{F}}$ by simply folding $\mathcal{Z}_{\mathcal{F}}$ to obtain $\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}$, apply Theorem 15 to get $\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})$, and then unfold the latter to obtain $\mathcal{ACD}(\mathcal{A})$. The first two steps require a time polynomial in $|\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}| + |Q| \leq |\mathcal{Z}_{\mathcal{F}}| + |Q|$, while the third step takes a time polynomial in $|\mathcal{ACD}(\mathcal{A})|$. Thus, to obtain the theorem, it suffices to establish

3. $|\mathcal{ACD}(\mathcal{A})|$ is polynomial in $|Q| + |\mathcal{Z}_{\mathcal{F}}|$.

The most technical part lies in the proofs of items 2 and 3, stated below.

▶ **Proposition 17.** *Let $\mathcal{A}$ be a Muller automaton and $\mathcal{F}$ the family defining its acceptance condition. Then,*

a) $|\mathcal{ACD}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|$.
b) $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})| \leq |Q| \cdot |\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$.

We describe now the main ideas of the proof of Proposition 17. We use the notion of *local subtree at a state* of the ACD. If q is a state of $\mathcal{A}$ appearing in the SCC ℓ_i, we define the *local subtree at* q, noted $\mathcal{T}_q$, as the subtree of $\mathsf{AltTree}(\ell_i)$ containing the nodes $N_q = \{n \in \mathsf{AltTree}(\ell_i) \mid q \text{ is a state in the label of } n\}$. We define analogously the *local subDAG of $\mathcal{ACD}$-$\mathsf{DAG}(\mathcal{A})$ at q*, noted $\mathcal{D}_q$.

We remark that $|\mathcal{ACD}(\mathcal{A})| \leq \sum_{q\in Q} |\mathcal{T}_q|$ (resp. $|\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})| \leq \sum_{q\in Q} |\mathcal{D}_q|$), as each node of the ACD appears in some local subtree. Therefore, it suffices to bound the size of the local subtrees (resp. local subDAGs) to obtain a polynomial bound on the size of $\mathcal{ACD}(\mathcal{A})$ (resp. $\mathcal{ACD}\text{-}\mathsf{DAG}(\mathcal{A})$) to deduce Proposition 17. Quite surprisingly, the arguments to bound these objects are slightly different in each case.

▶ **Lemma 18.** *For every state q, the tree $\mathcal{T}_q$ has size at most $|\mathcal{Z}_{\mathcal{F}}|$.*

Proof sketch. We define in a top-down fashion an injective function $f \colon \mathcal{T}_q \to \mathcal{Z}_{\mathcal{F}}$. For the base case, we send the root of $\mathcal{T}_q$ to the root of $\mathcal{Z}_{\mathcal{F}}$. Let n be a node in $\mathcal{T}_q$ such that $f(n)$ has been defined, and let $n_1, \dots, n_k$ be its children. The key technical result is to show that there are k descendants of $f(n)$, containing the sets of labels of $n_1, \dots, n_k$, respectively, that are incomparable for the ancestor relation. Then, the subtrees rooted at these nodes are pairwise disjoint, which allows to define $f(n_i)$ for all i and carry out the induction. ◀

We conclude that the size of $\mathcal{ACD}(\mathcal{A})$ is polynomial in $|Q| + |\mathcal{Z}_{\mathcal{F}}|$, deriving the first item of Proposition 17:

$$|\mathcal{ACD}(\mathcal{A})| \leq \sum_{q\in Q} |\mathcal{T}_q| \leq |Q| \cdot |\mathcal{Z}_{\mathcal{F}}|.$$

▶ **Lemma 19.** *For every state q, the DAG $\mathcal{D}_q$ has size at most $|\mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}|$.*

Proof sketch. As before, we define an injective function $f \colon \mathcal{D}_q \to \mathcal{Z}\text{-}\mathsf{DAG}_{\mathcal{F}}$. However, now we cannot use the fact that the subDAGs rooted at k incomparable elements are disjoint.

To circumvent this difficulty, for each node n in $\mathcal{D}_q$ different from the root, we fix an arbitrary immediate ancestor of n, noted $\mathsf{pred}(n)$ (that is, n is a child of $\mathsf{pred}(n)$). For a node n in $\mathcal{D}_q$, we let C_n be the set of colours appearing in the label of n. We define f recursively:

For the root n_0 of $\mathcal{D}_q$, we let $f(n_0)$ be a maximal (deepest) node in $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$ containing C_{n_0} in its label. For n a node such that we have define f for all its ancestors, we let $f(n)$ be a maximal node in the subDAG rooted at $f(\mathsf{pred}(n))$ containing C_n in its label (we note that $f(n)$ is a round node if and only if n is a round node). The most technical part of the proof is to show injectivity of the obtained function. ◀

4 Minimisation of colours and Rabin pairs

We consider the problem of minimising the representation of the acceptance condition of automata. That is, given a deterministic automaton $\mathcal{A}$ using a Muller (resp. Rabin) acceptance condition, what is the minimal number of colours (resp. Rabin pairs) needed to define an equivalent acceptance condition over $\mathcal{A}$?

We first study the minimisation of colours for Muller languages, without taking into account the structure of the automaton. We show that given the Zielonka DAG of the condition (resp. set of Rabin pairs), we can minimise its number of colours (resp. number of Rabin pairs) in polynomial time (Theorems 20 and 21). We provide an alternative point of view over the minimisation of Rabin pairs, using so-called generalised Horn formulas (see Remark 23). Then, we tackle the same question taking into account the structure of the automaton. Surprisingly, we show that in this case both problems are NP-complete, even if the ACD is given as input (Theorems 26 and 27).

4.1 Minimisation of the representation of Muller languages in PTIME and generalised Horn formulas

Minimisation of colours for Muller languages. We say that a Muller language $\mathsf{Muller}_\Sigma(\mathcal{F})$ is *k-colour type* if there is a set of k colours Γ, a family of sets $\mathcal{F}' \subseteq 2^\Gamma_+$ and a mapping $\phi\colon \Sigma \to \Gamma$ such that for all $S \in 2^\Sigma_+$, $S \in \mathcal{F} \iff \phi(S) \in \mathcal{F}'$.

Note that this is equivalent to asking if all automata using $\mathsf{Muller}_\Sigma(\mathcal{F})$ as acceptance condition can be relabelled with an equivalent Muller condition using at most k colours. (However, it is *not* the same as having a Muller automaton recognising $\mathsf{Muller}_\Sigma(\mathcal{F})$ using at most k colours.)

Colour-Minimisation-ML is the problem of deciding whether a given Muller language (represented by its Zielonka DAG) is k-colour type. We chose to specify the input as a Zielonka DAG, as it is more succinct than the other representations we consider (c.f. Figure 3). We now prove that this problem can be solved in polynomial time, which implies that it can be equally solved in polynomial time if the Muller language is represented colour-explicitly, or as a Zielonka tree.

▶ **Theorem 20** (Tractability of minimisation of colours for Muller languages). *The problem* Colour-Minimisation-ML *can be solved in polynomial time.*

Proof sketch. We define two colours $a, b \in \Sigma$ as equivalent if for every node n of $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$, $a \in \nu(n) \iff b \in \nu(n)$. It is not difficult to see that we can merge equivalent colours, that is, we can define $\mathsf{Muller}(\mathcal{F})$ using as many colours as the number of classes for this equivalence relation. We prove that this is optimal: If $\mathsf{Muller}(\mathcal{F})$ can be defined using a mapping $\phi\colon \Sigma \to \Gamma$, then, for all $\alpha \in \Gamma$, the colours in $\phi^{-1}(\alpha)$ are equivalent. Therefore, it suffices to inspect $\mathcal{Z}$-$\mathsf{DAG}_{\mathcal{F}}$ to determine the number of equivalence classes. ◀

Minimisation of Rabin pairs for Rabin languages. In this section we tackle the minimisation of the number of Rabin pairs to represent Rabin languages. We provide a polynomial-time algorithm which turns a family of Rabin pairs into an equivalent one with a minimal number of pairs. The algorithm comes down to partially computing the Zielonka tree of the input Rabin language from top to bottom, and stopping when we can infer from the partial view of the tree a set of Rabin pairs equivalent to the input. We present the algorithm differently to clarify the proofs, in particular the proof that the resulting number of pairs is minimal.

We say that a Rabin language $L \subseteq \Sigma^\omega$ is *k-Rabin-pair type* if there is a family of k Rabin pairs $\mathcal{R}$ over some set of colours Γ and a mapping $\phi\colon \Sigma \to \Gamma$ such that for all $w \in \Sigma^\omega$, $w \in L \iff \phi(w) \in \mathsf{Rabin}_\Gamma(\mathcal{R})$.

Rabin-Pair-Minimisation-ML is the problem of deciding whether a language $\mathsf{Rabin}_\Sigma(\mathcal{R})$ (represented by the Rabin pairs R) is k-Rabin-pair type.

▶ **Theorem 21** (Tractability of minimisation of Rabin pairs for Rabin languages)**.** *The problem* Rabin-Pair-Minimisation-ML *can be solved in polynomial time.*

We obtain the minimal set of Rabin pairs iteratively. We start with an empty set of pairs. While our set of pairs is not equivalent to the input one, we compute a maximal set of colours S accepted by the input set of pairs and not by our current set of pairs. We then compute the maximal subset T of S that is rejected by the input set of pairs. We infer from them a new Rabin pair, which accepts the sets of colours contained in S but not in T. We add this pair to our set of pairs.

We prove that the resulting set is optimal by showing that at all times, we can define an injective function from our current set of pairs to any set of pairs equivalent to the input.

The minimisation of pairs for Streett conditions in polynomial time follows by symmetry.

Generalised Horn formulas. We discuss an alternative point of view on the minimisation of Rabin pairs, via a generalisation of Horn formulas.

Horn formulas are a popular fragment of propositional logic, as they enjoy some convenient complexity properties. It is well-known that the satisfiability problem for those formulas can be solved in linear time [15].

We consider a succinct representation of Horn formulas, called generalised Horn formula. They allow one to merge several Horn clauses with the same premises, e.g. $(x_1 \wedge x_2 \implies y_1)$ and $(x_1 \wedge x_2 \implies y_2)$, into a single clause $(x_1 \wedge x_2 \implies y_1 \wedge y_2)$. We can apply the classical linear-time algorithm for satisfiability on this generalised form, however, note that it is not linear in the size of the generalised formula, but in the size of the implicit Horn formula represented.

▶ **Definition 22.** *A* generalised Horn clause *(or GH clause) is a Boolean formula of the form either $(x_1 \wedge \cdots \wedge x_n) \implies (y_1 \wedge \cdots \wedge y_m)$ or $(x_1 \wedge \cdots \wedge x_n) \implies \bot$ (in the latter case, the clause is called* negative*). A* generalised Horn formula *(or GH formula) is a conjunction of GH clauses. It is* simple *if none of its GH clauses are negative.*

▶ Remark 23 (Correspondence simple GH formulas ↔ Streett conditions). We observe that there is a correspondence between simple GH formulas and Streett conditions. Define the function α that turns a GH clause $(x_1 \wedge \cdots \wedge x_n) \implies (y_1 \wedge \cdots \wedge y_m)$ into the Rabin pair $(\{y_1, \ldots, y_m\}, \{x_1, \ldots, x_n\})$. We extend it into a function turning simple GH formulas into families of Rabin pairs by defining $\alpha(\bigwedge_{i=1}^k \mathrm{GH}_i) = (\alpha(\mathrm{GH}_i))_{i=1}^k$. We can then observe that α is a bijection (we consider Boolean formulas up to commutation of the terms, for instance we consider that $\varphi \vee \psi$ and $\psi \vee \varphi$ are the same formula). We also note that the number of clauses of a simple GH formula is the number of pairs of its image by α.

Note that for all simple GH formula φ, the set of sets accepted by the Streett condition $\alpha(\varphi)$ is $\{\nu^{-1}(\bot) \mid \nu : \mathrm{Var} \to \{\top, \bot\}$ is a valuation satisfying $\varphi\}$. As a result, two simple GH formula are equivalent if and only if their images by α define the same Streett language.

As a consequence of this correspondence and Theorem 21, we obtain that we can minimise the number of clauses in a GH formula in polynomial-time. This result contrasts nicely with the NP-completeness of minimising the number of clauses in a Horn formula [4] (see also [14]). On the other hand, minimising the number of literals in a GH formula remains NP-complete, just like in the case of Horn formulas [20]. This can be showed by a slight adaptation of the reduction from [14] to GH formulas.

▶ **Proposition 24.** *There is a polynomial-time algorithm to minimise the number of clauses of a GH formula.*

Proof sketch. The polynomial-time minimisation of simple GH formulas follows from Theorem 21 and Remark 23. The extension to all Generalised Horn formulas is essentially a technicality, due to the fact that negative clauses cannot be directly translated into Rabin pairs as in the previously. We circumvent this problem by replacing them with some non-negative clauses and proving that minimising the initial formula comes down to minimising the resulting simple one. ◀

On the other hand, generalised Horn formulas are likely not a suitable representation for acceptance conditions on automata, as they yield an NP-complete emptiness problem (Proposition 25). This is an interesting example of a family of acceptance conditions whose satisfiability problem is in PTIME but which yields an NP-complete emptiness problem on automata.

▶ **Proposition 25.** *Checking emptiness of an automaton with an acceptance condition represented by a GH formula is* NP*-complete.*

Proof sketch. The NP upper bound follows from the one on Emerson-Lei conditions. For the hardness, we reduce from the Hamiltonian cycle problem. ◀

4.2 Minimisation of acceptance conditions on top of an automaton

We now consider the problem of minimising the number of colours or Rabin pairs used by a Muller or Rabin condition over a fixed automaton. We could expect that it is possible to generalise the previous polynomial time algorithms by using the ACD, instead of the Zielonka DAG. Quite surprisingly, we show that these problems become NP-complete when taking into account the structure of the automata.

Minimisation of colours on top of a Muller automaton. We say that a deterministic Muller automaton $\mathcal{A}$ is *k-colour type* if we can relabel it with a Muller condition using at most k output colours that is equivalent over $\mathcal{A}$ (and uses a single colour per edge). We also consider automata with multiple colours per edge (in this section, multiple labels may be relevant). We will nevertheless show that allowing them does not change the theoretical complexity of the problem. We say that $\mathcal{A}$ is *k-multi-colour type* if we can relabel it with an equivalent Muller condition using at most k colours, with possibly several colours per edge.

Colour-Minimisation-Aut (resp. Multi-Colour-Minimisation-Aut) is the problem of deciding whether a deterministic Muller automaton is k-colour type (resp. k-multi-colour type). These problems admit different variants according to the representation of the Muller condition. We will show that for the three representations we are concerned with (colour-explicit, Zielonka tree and Zielonka DAG), both problems are NP-complete. This implies that they are NP-hard even if the ACD is provided as input, by Theorem 13. Hugenroth showed[5] that, *for state-based automata*, the problem Colour-Minimisation-Aut is NP-hard when the acceptance condition of $\mathcal{A}$ is represented colour-explicitly or as a Zielonka tree [21]. However, it is not straightforward to generalise it to *transition-based automata*, since the classic translation between state-based and transition-based automata does not preserve minimality.

▶ **Theorem 26** (NP-completeness of minimisation of colours for Muller automata)**.** *The problems* Colour-Minimisation-Aut *and* Multi-Colour-Minimisation-Aut *are* NP*-complete, if the acceptance condition* $\mathsf{Muller}_\Gamma(\mathcal{F})$ *of* $\mathcal{A}$ *is represented colour-explicitly, as a Zielonka tree, Zielonka DAG or as the ACD of* $\mathcal{A}$*.*

To obtain the NP-hardness, we reduce from the chromatic number problem for graphs. We note that the fact that these problems lie in NP is not obvious: we could be tempted to guess an acceptance condition on the same automaton structure and check equivalence of the two automata. However, reducing the number of colours might blow up the size of the representation of the acceptance condition.

NP-upper bound: Proof sketch. We guess a colouring $\mathsf{col}' : \Delta \to [k]$ and check in polynomial time that there exists a family $\mathcal{F}'$ over $[k]$ defining an equivalent condition over $\mathcal{A}$. To do so, we remark that such $\mathcal{F}'$ exists if and only if there is no pair of words $w_+ \in \mathcal{L}(A)$ and $w_- \notin \mathcal{L}(\mathcal{A})$ such that the sets of colours produced infinitely often under col' by their runs are equal. The existence of such words reduces to emptiness of adequate Streett automata. ◀

Minimisation of Rabin pairs on top of a Rabin automaton. We say that a deterministic Muller automaton $\mathcal{A}$ is *k-Rabin-pair type* if we can relabel it with an equivalent Rabin condition using at most k Rabin pairs.

Rabin-Pair-Minimisation-Aut is the problem of deciding whether a given deterministic Rabin automaton is k-Rabin-pair type. As before, we can consider different representations of the acceptance condition of the automaton: using Rabin pairs, colour-explicitly, the Zielonka tree, the Zielonka DAG or by providing the ACD.

▶ **Theorem 27** (NP-completeness of minimisation of Rabin pairs for Rabin automata)**.** *The problem* Rabin-Pair-Minimisation-Aut *is* NP*-complete for all the previous representations of the acceptance condition.*

5 Conclusion

In this work we obtained several positive results concerning the complexity of simplifying the acceptance condition of an ω-automaton. Our first technical result is that the computation of the ACD (resp. ACD-DAG) of a Muller automaton is not harder than the computation of the Zielonka tree (resp. Zielonka DAG) of its acceptance condition (Theorems 13 and 15). This provides support for the assertion that the optimal transformation into parity automata based on the ACD is applicable in practical scenarios, backing the experimental evidence provided by the implementations of the ACD-transform [12].

Furthermore, this result has several implications for our simplification purpose: We can decide the typeness of Muller automata and compute the parity index of their languages in polynomial time (Corollary 16). In addition, we showed that we can minimise in polynomial

[5] As of today, the proof is not currently publicly available online, we got access to it by a personal communication. The theorem only expresses the NP-hardness for the colour-explicit representation, but a look into the reduction works unchanged if the condition is given as a Zielonka tree.

time the colours and Rabin pairs necessary to represent a Muller language. However, these problems become NP-hard when taking into account the structure of a particular automaton using this acceptance condition, even if the ACD of the automaton is provided as input. Nevertheless, we believe that the methods for the minimisation of colours in the case of Muller languages could be combined with the structure of the ACD to obtain heuristics reducing the number of colours used by Muller automata, which might lead to substantial (although not optimal) reductions.

In sum, our results clarify the potential of the ACD and complete our understanding of the complexity of simplifying the acceptance conditions of ω-automata.

References

1 Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi omega-automata format. In *CAV*, pages 479–486, 2015. doi:10.1007/978-3-319-21690-4_31.

2 Christel Baier, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, David Müller, and Jan Strejček. Generic emptiness check for fun and profit. In *ATVA*, pages 445–461, 2019. doi:10.1007/978-3-030-31784-3_26.

3 Udi Boker. Why these automata types? In *LPAR*, volume 57 of *EPiC Series in Computing*, pages 143–163, 2018. doi:10.29007/c3bj.

4 Endre Boros and Ondřej Čepek. On the complexity of Horn minimization. *Rutgers University. Rutgers Center for Operations Research [RUTCOR]*, 1994.

5 J. Richard Büchi. On a decision method in restricted second order arithmetic. *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

6 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022. doi:10.1137/17M1145288.

7 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO*, pages 495–506, 1999. doi:10.1051/ita:1999129.

8 Antonio Casares. *Structural properties of automata over infinite words and memory for games (Propriétés structurelles des automates sur les mots infinis et mémoire pour les jeux)*. Phd thesis, Université de Bordeaux, France, 2023. URL: https://theses.hal.science/tel-04314678.

9 Antonio Casares, Thomas Colcombet, and Nathanaël Fijalkow. Optimal transformations of games and automata using Muller conditions. In *ICALP*, volume 198, pages 123:1–123:14, 2021. doi:10.4230/LIPIcs.ICALP.2021.123.

10 Antonio Casares, Thomas Colcombet, Nathanaël Fijalkow, and Karoliina Lehtinen. From Muller to Parity and Rabin Automata: Optimal Transformations Preserving (History) Determinism. *TheoretiCS*, Volume 3, April 2024. doi:10.46298/theoretics.24.12.

11 Antonio Casares, Thomas Colcombet, and Karoliina Lehtinen. On the size of good-for-games Rabin automata and its link with the memory in Muller games. In *ICALP*, volume 229, pages 117:1–117:20, 2022. doi:10.4230/LIPIcs.ICALP.2022.117.

12 Antonio Casares, Alexandre Duret-Lutz, Klara J. Meyer, Florian Renkin, and Salomon Sickert. Practical applications of the Alternating Cycle Decomposition. In *TACAS*, volume 13244 of *Lecture Notes in Computer Science*, pages 99–117, 2022. doi:10.1007/978-3-030-99527-0_6.

13 Antonio Casares and Corto Mascle. The complexity of simplifying ω-automata through the alternating cycle decomposition. *CoRR*, abs/2401.03811, 2024. doi:10.48550/arXiv.2401.03811.

14 Tom Chang. Horn formula minimization. Master's thesis, Rochester Institute of Technology, 2006. Available at https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=7895&context=theses.

15 William F Dowling and Jean H Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

16 Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. From Spot 2.0 to Spot 2.10: What's new? In *CAV*, volume 13372 of *Lecture Notes in Computer Science*, pages 174–187, 2022. doi:10.1007/978-3-031-13188-2_9.

17 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997. doi:10.1109/LICS.1997.614939.

18 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In *CAV*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396, 1993. doi:10.1007/3-540-56922-7_32.

19 Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS*, pages 426–442, 2017. doi:10.1007/978-3-662-54577-5_25.

20 Peter L Hammer and Alexander Kogan. Optimal compression of propositional Horn knowledge bases: complexity and approximation. *Artificial Intelligence*, 64(1):131–145, 1993.

21 Christopher Hugenroth. Zielonka DAG acceptance and regular languages over infinite words. In *DLT*, volume 13911 of *Lecture Notes in Computer Science*, pages 143–155, 2023. doi:10.1007/978-3-031-33264-7_12.

22 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *MFCS*, pages 495–506, 2005. doi:10.1007/11549345_43.

23 Paul Hunter and Anuj Dawar. Complexity bounds for Muller games. *Theoretical Computer Science (TCS)*, 2008.

24 Swen Jacobs, Guillermo A. Perez, Remco Abraham, Veronique Bruyere, Michael Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaetan Staquet, Clement Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021, 2022. arXiv:2206.00251.

25 Sriram C. Krishnan, Anuj Puri, and Robert K. Brayton. Structural complexity of omega-automata. In *STACS*, pages 143–156, 1995. doi:10.1007/3-540-59042-0_69.

26 Orna Kupferman, Gila Morgenstern, and Aniello Murano. Typeness for omega-regular automata. *Int. J. Found. Comput. Sci.*, 17(4):869–884, 2006. doi:10.1142/S0129054106004157.

27 Jan Křetínský, Tobias Meggendorfer, and Salomon Sickert. Owl: A library for ω-words, automata, and LTL. In *ATVA*, volume 11138 of *Lecture Notes in Computer Science*, pages 543–550, 2018. doi:10.1007/978-3-030-01090-4_34.

28 Christof Löding. Optimal bounds for transformations of ω-automata. In *FSTTCS*, pages 97–109, 1999. doi:10.1007/3-540-46691-6_8.

29 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, pages 3–36, 2020. doi:10.1007/s00236-019-00349-3.

30 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and control*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.

31 Philipp Meyer and Salomon Sickert. On the optimal and practical conversion of Emerson-Lei automata into parity automata. Unpublished manuscript, obsoleted by the work of [9], 2021.

32 Philipp J. Meyer and Salomon Sickert. Modernising Strix. *SYNT Workshop*, 2021. URL: https://www7.in.tum.de/~sickert/publications/MeyerS21.pdf.

33 Thibaud Michaud and Maximilien Colange. Reactive synthesis from LTL specification with Spot. In *SYNT@CAV*, Electronic Proceedings in Theoretical Computer Science, 2018.

34 Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *SCT*, pages 157–168, 1984. doi:10.1007/3-540-16066-3_15.

35 David Müller and Salomon Sickert. LTL to deterministic Emerson-Lei automata. In *GandALF*, pages 180–194, 2017. doi:10.4204/EPTCS.256.13.

36 Nir Piterman and Amir Pnueli. Faster solutions of Rabin and Streett games. In *LICS*,, pages 275–284. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.23.

37 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.

38 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. URL: http://www.jstor.org/stable/1995086.

39 Tereza Schwarzová, Jan Strejček, and Juraj Major. Reducing acceptance marks in Emerson-Lei automata by QBF solving. In *SAT*, volume 271, pages 23:1–23:20, 2023. doi:10.4230/LIPIcs.SAT.2023.23.

40 Cong Tian and Zhenhua Duan. Büchi determinization made tighter. *CoRR*, abs/1404.1436, 2014. arXiv:1404.1436.

41 Thomas Wilke and Haiseung Yoo. Computing the Rabin index of a regular language of infinite words. *Inf. Comput.*, pages 61–70, 1996. doi:10.1006/inco.1996.0082.

42 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

Distance to Transitivity: New Parameters for Taming Reachability in Temporal Graphs

Arnaud Casteigts
Department of Computer Science, University of Geneva, Switzerland

Nils Morawietz
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Petra Wolf
LaBRI, CNRS, Université de Bordeaux, Bordeaux INP, France

Abstract

A temporal graph is a graph whose edges only appear at certain points in time. Reachability in these graphs is defined in terms of paths that traverse the edges in chronological order (temporal paths). This form of reachability is neither symmetric nor transitive, the latter having important consequences on the computational complexity of even basic questions, such as computing temporal connected components. In this paper, we introduce several parameters that capture how far a temporal graph $\mathcal{G}$ is from being transitive, namely, *vertex-deletion distance to transitivity* and *arc-modification distance to transitivity*, both being applied to the reachability graph of $\mathcal{G}$. We illustrate the impact of these parameters on the temporal connected component problem, obtaining several tractability results in terms of fixed-parameter tractability and polynomial kernels. Significantly, these results are obtained without restrictions of the underlying graph, the snapshots, or the lifetime of the input graph. As such, our results isolate the impact of non-transitivity and confirm the key role that it plays in the hardness of temporal graph problems.

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

Keywords and phrases Temporal graphs, Parameterized complexity, Reachability, Transitivity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.36

Related Version *Full Version*: https://arxiv.org/abs/2406.19514

Funding *Arnaud Casteigts*: supported by French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).
Petra Wolf: supported by French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).

1 Introduction

Temporal graphs have gained attention lately as appropriate tools to capture time-dependent phenomena in fields as various as transportation, social networks analysis, biology, robotics, scheduling, and distributed computing. On the theoretical side, these graphs generate interest mostly for their intriguing features. Indeed, many basic questions are still open, with a general feeling that existing techniques from graph theory typically fail on temporal graphs. In fact, most of the natural questions considered in static graphs turn out to be intractable when formulated in a temporal version, and likewise, most of the temporal analogs of classical structural properties are false.

One of the earliest examples is that the natural analog of Menger's theorem does not hold in temporal graphs [21]. Another early result is that deciding if a temporal connected component (set of vertices that can reach each other through temporal paths) of a certain size exists is NP-complete [6]. A more recent and striking result is that there exist temporally connected graphs on $\Theta(n^2)$ edges in which every edge is critical for connectivity; in other words, no temporal analog of sparse spanners exist unconditionally [5] (though they do,

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 36; pp. 36:1–36:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

probabilistically [11]). Moreover, minimizing the size of such spanners is APX-hard [2, 5]. Further hardness results for problems whose static versions are generally tractable include separators [19], connectivity mitigation [16], exploration [4, 17], flows [1], Eulerian paths [7], and even spanning trees [9].

Faced by these difficulties, the algorithmic community has focused on special cases, and tools from parameterized complexity were employed with moderate success. A natural approach here is to apply the range of classical graph parameters to restrict either the underlying graph of the temporal graph (i.e. which edges can exist at all) or its snapshots (i.e. which edges may exist simultaneously). For example, finding temporal paths with bounded waiting time at each node (which is NP-hard in general) turns out to be FPT when parameterized by treedepth or vertex cover number of the underlying graph. But the problem is already W[1]-hard for pathwidth (let alone treewidth) [10]. In fact, as observed in [18], most temporal graphs problems remain hard even when the underlying graph has bounded treewidth (sometimes, even a tree or a star [3, 4, 16]).

A possible explanation for these results is that temporal graph problems are *very* hard. Another one is that parameters based on static graph properties are not adequate. Some parameters whose definition is based on that of a temporal graph include timed feedback vertex sets (counting the cumulative distance to trees over all snapshots) [10] and the $p(\mathcal{G})$ parameter from [4], that measures in a certain way how dynamic the temporal graph is and enables polynomial kernels for the exploration problem. While these parameters represent some progress towards finer-grained restrictions, they remain somewhat structural in the sense that their definition is stable under re-shuffling of the snapshots.

A key aspect of temporal graphs is that the *ordering of events* matters. Arguably, a truly temporal parameter should be sensitive to that. An interesting step in this direction was recently made by Bumpus and Meeks [7], introducing interval-membership-width, a parameter that quantifies the extent to which the set of intervals defined by the first and last appearance of an edge at each vertex can overlap (with application to Eulerian paths). In a sense, this parameter measures how complex the interleaving of events could be. Another, perhaps even more fundamental feature of temporal graphs is that the reachability relation based on temporal paths is not guaranteed to be symmetric or transitive. While the former is a well-known limitation of directed graphs, the latter is specific to temporal graphs (directed or not), and it has been suspected to be one of the main sources of intractability since the onset of the theory. (Note that a temporal graph of bounded interval-membership-width may still be arbitrarily non-transitive.) In the present work, we explore new parameters that control how transitive a temporal graph is, thereby isolating, and confirming, the role that this aspect plays in the tractability of temporal reachability problems.

Our Contributions. We introduce and investigate two parameters that measure how far a temporal graph is from having transitive reachability. For a temporal graph $\mathcal{G}$, our parameters directly address the reachability features of $\mathcal{G}$, and as such, they are formulated in terms of its reachability graph $G_R = (V, \{(u, v) : u \rightsquigarrow v\})$, a directed graph whose arcs represent the existence of temporal paths in $\mathcal{G}$, whether $\mathcal{G}$ itself is directed or undirected. Indeed, the reachability of $\mathcal{G}$ is transitive if and only if the arc relation of G_R is transitive. Two natural ways of measuring this distance are in terms of *vertex deletion* and *arc modification*, namely:

- *Vertex-deletion distance to transitivity* (δ_{vd}) is the minimum number of vertices whose deletion from G_R makes the resulting graph transitive.
- *Arc-modification distance to transitivity* (δ_{am}) is the minimum number of arcs whose addition or deletion from G_R makes the resulting graph transitive.

As for the arc-modification distance, we may occasionally consider its restriction to arc-addition only (δ_{aa}).

Among the many problems that were shown intractable in temporal graphs, one of the first, and perhaps most iconic one, is the computation of temporal connected components [6] (see also [13, 23]). In order to benchmark our new parameters, we investigate their impact on the computational complexity of this problem. Informally, given a temporal graph $\mathcal{G}$ (defined later) on a set of vertices V, a temporal connected component is a subset $V' \subseteq V$ such that for all u and v in V', u can reach v by a temporal path. Interestingly, the non-transitive nature of reachability here makes it possible for such vertices to reach each other through temporal paths that travel outside the component, without absorbing the intermediate vertices into the component. This gives rise to two distinct notions of components: *open* temporal connected components (Open-TCC) and *closed* temporal connected components (Closed-TCC), the latter requiring that only internal vertices are used in the temporal paths, and both being NP-hard to compute.

The statement of our results requires a few more facts. Both algorithmic and structural results in temporal graphs are highly sensitive to subtle definitional variations, called *settings*. In the *non-strict* setting, the labels along a temporal path are only required to be non-decreasing, whereas in the *strict* setting, they must be increasing. It turns out that both settings are sometimes incomparable in difficulty, and the techniques developed for each may be different. Some temporal graphs, called *proper*, have the property that no two adjacent edges share a common time label, making it possible to ignore the distinction between strict and non-strict temporal paths. Whenever possible, hardness results should preferably be obtained for proper temporal graphs, so that they apply in both settings at once. Finally, with a few exceptions, our results hold for both directed and undirected temporal graphs.

Bearing these notions in mind, our results are the following. For Open-TCC, we obtain an FPT algorithm with parameter δ_{vd}, running in time $3^{\delta_{\text{vd}}} \cdot n^{\mathcal{O}(1)}$ (in all the settings). Unfortunately, δ_{vd} turns out to be too small for obtaining a kernel of polynomial size. In fact, we show that under reasonable computational complexity assumptions, no polynomial kernel in $\delta_{\text{vd}} + \text{vc} + k$ exists (except possibly for the non-strict undirected setting), where k denotes the size of the sought tcc and where vc denotes the vertex cover number of the underlying graph. Next, we obtain an FPT algorithm running in time $4^{\delta_{\text{am}}} \cdot n^{\mathcal{O}(1)}$ for the mostly larger parameter δ_{am}, and show that Open-TCC admits a polynomial kernel of size $|M|^3$, where M is a given arc set for which $(V, A(G_R)\Delta M)$ is transitive. It also admits a polynomial kernel of size δ_{aa}^2 when restricting modification to addition-only (again, all these results hold in all the settings). Closed-TCC, in comparison, seems to be a harder problem, at least with respect to our parameters. In particular, we show that it remains NP-hard even if $\delta_{\text{am}} = \delta_{\text{vd}} = 1$ in all the settings (through proper graphs). It is also W[1]-hard when parameterized by $\delta_{\text{vd}} + \delta_{\text{am}} + k$ in all the settings, except possibly in the non-strict undirected setting. In fact, these two results hold even for temporal graphs whose reachability graph misses a single arc towards being a bidirectional clique.

Put together, these results establish clearly that non-transitivity is a genuine source of hardness for Open-TCC. The case of Closed-TCC is less clear. On the one hand, the parameters do not suffice to make this particular version of the problem tractable. This is not so surprising, as the reachability graph itself does not encode which paths are responsible for reachability, in particular, whether these paths are internal or external in a component. On the other hand, this gives us a separation between both versions of the problem and provides some support for the fact that Closed-TCC may be harder than Open-TCC, which was not known before. Finally, the negative results for Closed-TCC can serve as a landmark result for guiding future efforts in defining transitivity parameters that exploit more sophisticated structures than the reachability graph.

Organization of the Work. The main definitions are given in Section 2. Then, we investigate each parameter in a dedicated section (δ_{vd} in Section 3 and δ_{am} in Section 4). The limitations of these parameters in the case of Closed-TCC are presented in Section 5. Finally, Section 6 concludes the paper with some remarks and open questions. Due to space limitations, the proofs of statements marked with ($\star$) are deferred to a full version.

2 Preliminaries

For concepts of parameterized complexity, like FPT, W[1]-hardness, and polynomial kernels, we refer to the standard monographs [14, 15]. A reduction g between two parameterized problems is called a *polynomial parameter transformation*, if the reduction can be computed in polynomial time and, if for every input instance (I, k), we have that $(I', k') = g(I, k)$ with $k' \in k^{\mathcal{O}(1)}$. We call a polynomial time reduction from a problem L to L itself a *self-reduction*.

Notation. Let j be a positive integer, we denote with $[j]$ the set $\{1, 2, \ldots, j\}$. Moreover, for $1 \leq i \leq j$, we define $[i, j] := [j] \setminus [i-1]$. For a decision problem L, we say that two instances I_1, I_2 of L are *equivalent* if I_1 is a yes-instance of L if and only if I_2 is a yes-instance of L. For two sets A and B, we denote with $A \Delta B$ the symmetric difference of A and B.

Graphs. We consider a graph $G = (V, E)$ to be a static graph. If not indicated otherwise, we assume G to be undirected. Given a (directed) graph G, we denote by $V(G)$ the set of vertices of G, by $E(G)$ (respectively, $A(G)$) the set of edges (arcs) of G. Let $G = (V, E)$ be a graph and let $X \subseteq V(G)$ be a set of vertices. We denote by $E_G(X) = \{\{u, v\} \in E \mid u \in X, v \in X\}$ the edges in G between the vertices of S. Moreover, we define the following operations on G: $G[X] = (X, E_G[X])$, $G - X = G[V \setminus X]$. We call a sequence $\rho = v_0, v_1, \ldots, v_r$ of vertices a *path* in graph G if $v_0, \ldots, v_r \in V(G)$ and for each $i \in [r]$, $\{v_{i-1}, v_i\} \in E(G)$. We denote with $N_G[v]$ the closed neighborhood of the vertex $v \in V(G)$. A vertex set $S \subseteq V$ is a *clique* in an undirected graph, if each pair of vertices in S is adjacent in G. For a directed graph $G = (V, A)$, we call a set $S \subseteq V$ a *bidirectional clique*, if for every pair of distinct vertices u, v in S, we have $(u, v) \in A$ and $(v, u) \in A$. Let $G = (V, A)$ be a directed graph. A *strongly connected component (scc)* in G is an inclusion maximal vertex set $S \subseteq V$ under the property that there is a directed path in G between any two vertices of S. For each directed graph G, there is a unique partition of the vertex set of G into sccs. Moreover, this partition can be computed in linear time [24].

Temporal graphs. A *temporal graph* $\mathcal{G}$ over a set of vertices V is a sequence $\mathcal{G} = (G_1, G_2, \ldots, G_L)$ of graphs such that for all $t \in [L], V(G_t) = V$. We call L the *lifetime* of $\mathcal{G}$ and for $t \in [L]$, we call $G_t = (V, E_t)$ the *snapshot graph* of $\mathcal{G}$ at *time step* t. We call $G = (V, E)$ with $E = \bigcup_{t \in [L]} E_t$ the *underlying graph* of $\mathcal{G}$. We denote by $V(\mathcal{G})$ the set of vertices of $\mathcal{G}$. We write V if the temporal graph is clear from context. We call an undirected temporal graph $\mathcal{G} = (G_1, G_2, \ldots, G_L)$ *proper*, if for each vertex $v \in V(\mathcal{G})$ the degree of v in G_t is one, for each $t \leq L$. We call a directed temporal graph $\mathcal{G} = (G_1, G_2, \ldots, G_L)$ *proper*, if for each vertex $v \in V(\mathcal{G})$ the out-degree or the in-degree of v in G_t is zero, for each $t \leq L$. We further call a (directed) temporal graph $\mathcal{G}$ *simple*, if each edge (arc) exists in exactly one snapshot. We call a sequence $v_0, v_1, \ldots, v_r$ of vertices that form a path in the underlying graph G of $\mathcal{G}$ a *strict (non-strict) temporal path* in $\mathcal{G}$ if for each $i \in [r]$, there exists an $j_i \in [L]$ such that $\{v_{i-1}, v_i\} \in E(G_{j_i})$ and the sequence of indices j_i is increasing (non-decreasing).

For a temporal graph $\mathcal{G}$, we say that a vertex $u \in V$ *strictly (non-strictly) reaches* a vertex $v \in V$ if there is a strict (non-strict) temporal path from u to v, i.e., with $v_0 = u$ and $v_r = v$. We define the *strict (non-strict) reachability relation* $R \subseteq V \times V$ as: for all $u, v \in V$, $(u, v) \in R$ if and only if u strictly (non-strictly) reaches v. We call the directed graph $G_R = (V, R)$ the *strict (non-strict) reachability graph* of $\mathcal{G}$. We say that G_R is transitive, if and only if R is transitive. More generally, we say that a directed graph G is *transitive*, if its set of arcs forms a transitive relation. For a directed graph $G = (V, A)$ we call a set of vertices $S \subseteq V$ a *transitivity modulator* if $G - S$ is transitive.

▶ **Observation 1.** *Let G be a transitive directed graph. Then, for each vertex $v \in V(G)$, $G[V \setminus \{v\}]$ is also transitive.*

Next we define our main problems of interest in this work: Finding open and closed temporal connected components.

> Open Temporal Connected Component (Open-TCC)
> **Input:** Temporal graph $\mathcal{G} = (G_1, G_2, \ldots, G_L)$ and integer k.
> **Question:** Does there exists an open temporal connected component of size at least k, i.e., a subset $C \subseteq V(\mathcal{G})$ with $|C| \geq k$, such that for each $u, v \in C$, u reaches v, and vice versa.

We differentiate between the strict vs. non-strict and directed vs. undirected version of Open-TCC depending on whether we consider strict vs. non-strict reachability and directed vs. undirected temporal graphs. We define the problem Closed Temporal Connected Component (Closed-TCC) similarly with the additional restriction that at least one temporal path over which u reaches v is fully contained in C. We abbreviate a temporal connected component as tcc.

Distance to transitivity. We introduce two parameters that measure how far the reachability graph $G_R = (V, A)$ of a temporal graph is from being transitive. The first parameter, *vertex-deletion distance to transitivity*, δ_{vd}, counts how many vertices need to be deleted from G_R in order to obtain a transitive reachability graph, i.e., the size of a minimum transitivity modulator. This parameter is especially suited for temporal graphs for which the reachability graph consists of cliques with small overlaps. The second parameter, *arc-modification distance to transitivity*, δ_{am}, counts how many arcs need to be added to or removed from G_R in order to obtain a transitive reachability graph and is especially suited for directed temporal graphs or temporal graphs for which the reachability graph consists of cliques with large overlaps. Formally, we define the parameters as follows.

$$\delta_{\text{vd}} = \min_{S \subseteq V}(|S|) \text{ for which } G'_R = G_R - S \text{ is transitive.}$$

$$\delta_{\text{am}} = \min_{M \subseteq V \times V}(|M|) \text{ for which } G'_R = (V, A \Delta M) \text{ is transitive.}$$

For δ_{am}, we call the set M an *arc-modification set*. Note that $\delta_{\text{vd}} \leq 2 \cdot \delta_{\text{am}}$, since the endpoints of an arc-modification set form a transitivity modulator.

2.1 Basic Observations

Next, we present basic observations that motivate the study of the considered parameters.

▶ **Lemma 2** ([6]). *Let $\mathcal{G}$ be a temporal graph with reachability graph G_R. Then a set $S \subseteq V(\mathcal{G})$ is a tcc in $\mathcal{G}$ if and only if S is a bidirectional clique in G_R.*

▶ **Lemma 3** (⋆). *Let G be a transitive directed graph. Then every vertex set $S \subseteq V(G)$ is a bidirectional clique in G if and only if each pair of vertices of S can reach each other.*

Note that this implies the following.

▶ **Corollary 4.** *Let G be a transitive directed graph. Then every scc in G is also a maximal bidirectional clique and vice versa.*

The previous observations thereby imply that both Open-TCC and Closed-TCC can be solved in polynomial time on temporal graphs with transitive reachability graphs.

3 Vertex-Deletion Distance to Transitivity

We first focus on the parameter δ_{vd}. Note that computing this parameter is NP-hard: In a strict temporal graph $\mathcal{G}$ with lifetime 1, the reachability graph G_R of $\mathcal{G}$ is exactly the directed graph obtained from orienting each edge of the underlying graph in both directions. Hence, on such a temporal graph, computing δ_{vd} is exactly the cluster vertex deletion number of the underlying graph, that is, the minimum size of any vertex set to remove, such that no induced path of length 2 remains. Since computing the latter parameter is NP-hard [22], this hardness also translates to computing the parameter δ_{vd}.

Moreover, note that computing this parameter can be done similarly to computing the cluster vertex deletion number of a graph: If a directed graph $G = (V, A)$ is not transitive, then there are vertices u, v, and w in V, such that (u, v) and (v, w) are arcs of A and (u, w) is not an arc of A. Hence, each transitivity modulator for G has to contain at least one of the vertices u, v, or w. This implies, that a standard branching algorithm that considers each of these three vertices to be removed from the graph to obtain a transitive graph, finds a minimum size transitivity modulator in $3^{\delta_{\text{vd}}} \cdot n^{\mathcal{O}(1)}$ time.

▶ **Proposition 5.** *Let $\mathcal{G}$ be a temporal graph with reachability graph G_R. Then, we can compute in time $3^{\delta_{\text{vd}}} \cdot n^{\mathcal{O}(1)}$ a minimal-size transitivity modulator of G_R.*

Based on this result, we now present an FPT-algorithm for Open-TCC when parameterized by δ_{vd}.

▶ **Lemma 6.** *Let $I := (\mathcal{G}, k)$ be an instance of Open-TCC with reachability graph G_R. Let S be a given transitivity modulator of G_R. Then, we can solve I in time $2^{|S|} \cdot n^{\mathcal{O}(1)}$.*

Proof. By Lemma 4, every scc in $G_R[V \setminus S]$ is a bidirectional clique, since S is a transitivity modulator for G_R. Lemma 2 then implies that each tcc C in $\mathcal{G}$ with $C \cap S = \emptyset$ is an scc in $G_R[V \setminus S]$ and vice versa.

The FPT-algorithm then works as follows: We iterate over all subsets S' of S with the idea to find a tcc that extends S'. If S' is not a bidirectional clique in G_R, we discard the current set and continue with the next subset of S, as no superset of S' is a bidirectional clique and thus also not a tcc. Hence, assume that S' is a bidirectional clique. If S' has size at least k, I is a trivial yes-instance of Open-TCC. Otherwise, we do the following: Let V' be the vertices of $V \setminus S$ that are bidirectional connected to every vertex in S'. As $G_R[V \setminus S]$ is transitive, Observation 1 implies that $G_R[V']$ is also transitive. Hence, the sccs in $G_R[V']$ correspond to tccs in $\mathcal{G}$ by Corollary 4 and Lemma 2. Since every vertex in S' is bidirectional connected to every other vertex in $S' \cup V'$ in G_R, for each bidirectional clique $C \subseteq V'$ in $G_R[V']$, $C \cup S'$ is a tcc in $\mathcal{G}$. Hence, it remains to check, whether any scc in $G_R[V']$ has size at least $k - |S'|$. Figure 1 illustrates the sets S, S', and V'.

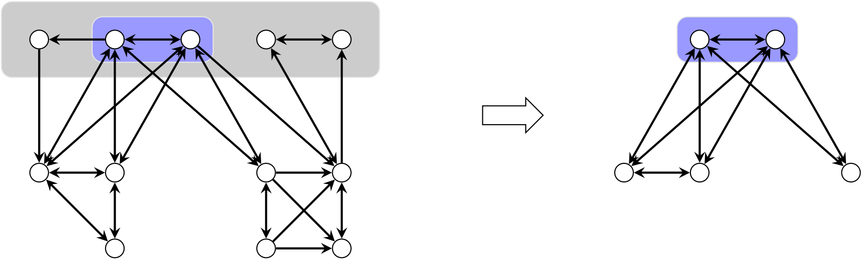


Figure 1 Illustration of the algorithm in Lemma 6. On the left: reachability graph G_R with transitivity modulator S in gray and the chosen subset $S' \subseteq S$ to extend in blue. On the right: The subset S' together with the vertices V' that are bidirectionally connected to all vertices in S'.

Finding the strongly connected components of a graph and identifying whether a set of vertices forms a bidirectional clique can be done in polynomial time. Hence, our algorithm runs in time $2^{\delta_{\mathrm{vd}}} \cdot n^{\mathcal{O}(1)}$, since we iterate over each subset S' of S. ◀

Based on Proposition 5 and Lemma 6, we thus derive our FPT-algorithm for Open-TCC when parameterized by δ_{vd}.

▶ **Theorem 7.** *Open-TCC can be solved in $3^{\delta_{\mathrm{vd}}} \cdot n^{\mathcal{O}(1)}$ time.*

Kernelization Lower Bounds

In this section, we show that a polynomial kernel for Open-TCC when parameterized by $\delta_{\mathrm{vd}} + \mathrm{vc} + k$ is unlikely, where vc is the vertex cover number of the underlying graph. Note that δ_{vd} and vc are incomparable: On the one hand, consider a temporal graph $\mathcal{G}$ where the underlying graph G is a star with leaf set $X \cup Y$ and center c, such that the edges from X to c exist in snapshots G_1 and G_3 and the edges from Y to c exist in snapshot G_2. Then, each vertex of X can reach each other vertex, but in the strict setting, no vertex of Y can reach any other vertex of Y. Hence, each minimum transitivity modulator has to contain all vertices of X or all but one vertex of Y, which implies that for $|X| = |Y|$, $\delta_{\mathrm{vd}} \in \Theta(|V(\mathcal{G})|)$, whereas the vertex cover number of G is only 1. On the other hand, consider a temporal graph $\mathcal{G}$ with only one snapshot G_1, such that G_1 is a clique. Then, the underlying graph of $\mathcal{G}$ is exactly G_1 and has a vertex cover number of $|V(\mathcal{G})| - 1$, but the strict reachability graph of $\mathcal{G}$ is a bidirectional clique, which is a transitive graph. Hence, $\delta_{\mathrm{vd}}(\mathcal{G}) = 0$.

We now present our kernelization lower bound for the strict undirected version of Open-TCC.

▶ **Theorem 8.** *The strict undirected version of Open-TCC does not admit a polynomial kernel when parameterized by $\mathrm{vc} + \delta_{\mathrm{vd}} + k$, unless NP ⊆ coNP/poly, where vc denotes the vertex cover number of the underlying graph.*

Proof. This result immediately follows from the known [21] reduction from Clique which, in fact, is as a polynomial parameter transformation.

Clique
Input: An undirected graph $G = (V, E)$ and integer k.
Question: Is there a clique of size k in G?

For the sake of completeness, we recall the reduction. Let $I := (G = (V, E), k)$ be an instance of Clique and let $\mathcal{G}$ be the temporal graph with lifetime 1, where G is the unique snapshot of $\mathcal{G}$.

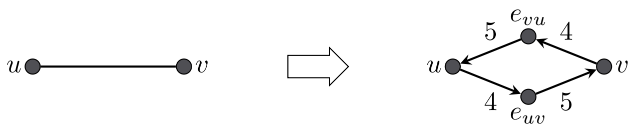


Figure 2 For two adjacent vertices u and v of S the vertices and arcs added to the temporal graph $\widetilde{\mathcal{G}}$ in the proof of Theorem 9.

Then, for each vertex set $X \subseteq V$, X is a clique in G if and only if X is a strict tcc in $\mathcal{G}$. Hence, I is a yes-instance of Clique if and only if $(\mathcal{G}, k)$ is a yes-instance of the strict undirected version of Open-TCC. It is known that Clique does not admit a polynomial kernel when parameterized by k plus the vertex cover number of G, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [12]. Let S be a minimum size vertex cover of G and let G_R be the strict reachability graph of $\mathcal{G}$. Note that G_R contains an arc (u, v) with $u \neq v$ if and only if $\{u, v\}$ is an edge of G. Hence, $V \setminus S$ is an independent set in G_R, which implies that S is a transitivity modulator of G_R. Consequently, $\delta_{\text{vd}} \leq |S|$. Recall that Clique does not admit a polynomial kernel when parameterized by $k + |S|$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [12]. This implies that the strict undirected version of Open-TCC does not admit a polynomial kernels when parameterized by $\text{vc} + \delta_{\text{vd}} + k$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. ◀

Next, we present the same lower bound for both directed versions of Open-TCC.

▶ **Theorem 9.** *The directed version of Open-TCC does not admit a polynomial kernel when parameterized by* $\text{vc} + \delta_{\text{vd}} + k$*, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$*, where* vc *denotes the vertex cover number of the underlying graph. This holds both for the strict and the non-strict version of the problem.*

Proof. Again, we present a polynomial parameter transformation from Clique.

Recall that Clique does not admit a polynomial kernel when parameterized by the size of a give minimum size vertex cover S of G plus k, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$ [12]. This holds even if $G[S]$ is $(k-1)$-partite [20], which implies that each clique of size k in G contains exactly $k-1$ vertices of S and exactly one vertex of $V \setminus S$, since $V \setminus S$ is an independent set.

Construction. Let $I := (G := (V, E), k)$ be an instance of Clique and let S be a given minimum size vertex cover S of G, such that $G[S]$ is $(k-1)$-partite. Assume that $k > 6$.

We obtain an equivalent instance of Open-TCC in two steps: First, we perform an adaptation of a known reduction [6] from the instance $(G[S], k-1)$ of Clique to an instance $(\widetilde{\mathcal{G}}, k-1)$ of the directed version of Open-TCC where each sufficiently large (of size at least 5) vertex set X of $\widetilde{\mathcal{G}}$ is a tcc in $\widetilde{\mathcal{G}}$ if and only if X is a clique in $G[S]$. Second, we extend $\widetilde{\mathcal{G}}$ by the vertices of $V \setminus S$ and some additional connectivity-gadgets, to ensure that the resulting temporal graph has a tcc of size k if and only if there is a vertex from $V \setminus S$ for which the neighborhood in G contains a clique of size $k-1$.

Let $(\widetilde{\mathcal{G}}, k-1)$ be the instance of Open-TCC constructed as follows: We initialize $\widetilde{\mathcal{G}}$ as an edgeless temporal graph of lifetime 5 with vertex set $S \cup \{e_{uv}, e_{vu} \mid \{u, v\} \in E_G(S)\}$. Next, for each edge $\{u, v\} \in E$, we add the arcs (u, e_{uv}) and (v, e_{vu}) to time step 4 and add the arcs (e_{uv}, v) and (e_{vu}, u) to time step 5. This completes the construction of $\widetilde{\mathcal{G}}$. An example of the arcs added to $\widetilde{\mathcal{G}}$ is shown in Figure 2. Note that the first three snapshots of $\widetilde{\mathcal{G}}$ are edgeless. This construction is an adaptation of the reduction presented by Bhadra and Ferreira [6]

Table 1 For each vertex $v \in V(\mathcal{G}')$ a lower bound for $\text{out}_v^{\min}$ and an upper bound for $\text{in}_v^{\max}$.

	$\text{out}_v^{\min}$	$\text{in}_v^{\max}$
$v \in S$	2	5
$v \in V(\widetilde{\mathcal{G}}) \setminus S$	4	5
$v \in V \setminus S$	1	2
$v \in \{u_{\text{in}} \mid u \in S\}$	3	1

to the case of directed temporal graphs. Note that the temporal graph $\widetilde{\mathcal{G}}$ has the following properties that we make use of in our reduction:

1) $\widetilde{\mathcal{G}}$ is a proper and simple directed temporal graph,
2) the vertex set $\mathcal{V}$ of $\widetilde{\mathcal{G}}$ has size $\mathcal{O}(|S|^2)$ and contains all vertices of S,
3) each tcc of size at least $k-1$ in $\widetilde{\mathcal{G}}$ contains only vertices of S, and
4) each vertex set $X \subseteq S$ of size at least $k-1$ is a tcc in $\widetilde{\mathcal{G}}$ if and only if X is a clique in $G[S]$.

Note that the two last properties imply that the largest tcc of $\widetilde{\mathcal{G}}$ has size at most $k-1$, since $G[S]$ is $(k-1)$-partite.

Next, we describe how to extend the temporal graph $\widetilde{\mathcal{G}}$ to obtain a temporal graph $\mathcal{G}'$ which has a tcc of size k if and only if I is a yes-instance of Clique. Let $n := |V|$. Moreover, let $\mathcal{G}'$ be a copy of $\widetilde{\mathcal{G}}$. We extend the vertex set of $\mathcal{G}'$ by all vertices of $V \setminus S$, and a vertex v_{in} for each vertex $v \in S$.

For each vertex $v \in S$, we add the arc (v_{in}, v) to time step 3. For each vertex $v \in S$ and each neighbor $w \in V \setminus S$ of v in G, we add the arc (v, w) to time step 2 and the arc (w, v_{in}) to time step 1. This completes the construction of $\mathcal{G}'$. Let V' denote the newly added vertices, that is, $V' := (V \setminus S) \cup \{v_{\text{in}} \mid v \in S\}$.

Next, we show that there is a clique of size k in G if and only if there is a tcc of size k in $\mathcal{G}'$.

($\Rightarrow$) Let $K \subseteq V$ be a clique of size k in G. We show that K is a tcc in $\mathcal{G}'$. As discussed above, K contains exactly $k-1$ vertices of S and exactly one vertex w^* of $V \setminus S$. By construction of $\widetilde{\mathcal{G}}$, $K \setminus \{w^*\}$ is a tcc in $\widetilde{\mathcal{G}}$ and thus also a tcc in $\mathcal{G}'$. It thus remains to show that each vertex $K \setminus \{w^*\}$ can reach vertex w^* in $\mathcal{G}'$ and vice versa. Since each vertex of $K \setminus \{w^*\}$ is adjacent to w^* in G, by construction, w^* is an out-neighbor of each vertex of $K \setminus \{w^*\}$ in $\mathcal{G}'$. Hence, it remains to show that w^* can reach each vertex of $K \setminus \{w^*\}$ in $\mathcal{G}'$. Let v be a vertex of $K \setminus \{w^*\}$. Since v is adjacent to w^* in G, there is an arc (w^*, v_{in}) in $\mathcal{G}'$ that exists at time step 1. Hence, there is a temporal path from w^* to v in $\mathcal{G}'$, since the arc (v_{in}, v) exists at time step 3. Concluding, K is a tcc in $\mathcal{G}'$.

($\Leftarrow$) Let X be a tcc of size k in $\mathcal{G}'$. We show that X is a clique of size k in G. To this end, we first show that X contains only vertices of V. Afterwards, we show that X is a clique in G.

To show that X contains only vertices of V, we first analyze the reachability of vertices of $V(\mathcal{G}')$. For a vertex $v \in V(\mathcal{G}')$, we denote

- by $\text{out}_v^{\min}$ the smallest time label of any arc exiting v and
- by $\text{in}_v^{\max}$ the largest time label of any arc entering v.

Note that a vertex v cannot reach a distinct vertex w in $\mathcal{G}'$ if $\text{in}_w^{\max} < \text{out}_v^{\min}$. Table 1 shows for each vertex $v \in V(\mathcal{G}')$ a lower bound for $\text{out}_v^{\min}$ and an upper bound for $\text{in}_v^{\max}$.

Based on Table 1, we can derive the following properties about reachability in $\mathcal{G}'$.

▷ Claim 10.

a) No vertex of $V(\widetilde{\mathcal{G}}) \setminus S$ can reach any vertex of V' in $\mathcal{G}'$.
b) No vertex of $\{v_{\text{in}} \mid v \in S\}$ can reach any other vertex of $\{v_{\text{in}} \mid v \in S\}$ in $\mathcal{G}'$.

c) No vertex of $\{v_{\text{in}} \mid v \in S\}$ can reach any vertex of $V \setminus S$ in $\mathcal{G}'$.
d) No vertex of S can reach any vertex of $\{v_{\text{in}} \mid v \in S\}$ in $\mathcal{G}'$.
e) No vertex of $V \setminus S$ can reach any other vertex of $V \setminus S$ in $\mathcal{G}'$.

Proof. Based on Table 1, we derive Items a) to d). It remains to show Item e). To this end, observe that each arc with a vertex of $V \setminus S$ as source has a vertex of $\{v_{\text{in}} \mid v \in S\}$ as sink. Due to Item c), no vertex of $\{v_{\text{in}} \mid v \in S\}$ can reach any vertex of $V \setminus S$ in $\mathcal{G}'$. Hence, no vertex $V \setminus S$ can reach any other vertex of $V \setminus S$ in $\mathcal{G}'$. This implies that Item e) holds. $\lhd$

Since X is a tcc in $\mathcal{G}'$, Claim 10 implies that X contains at most one vertex of $V \setminus S$ (due to Item e)) and at most one vertex of $\{v_{\text{in}} \mid v \in S\}$ (due to Item b)). In other words, X contains at most two vertices of V'. Since $k > 6$, this then implies that X contains at least one vertex of $V(\widetilde{\mathcal{G}})$. Claim 10 thus further implies that X contains no vertex of $\{v_{\text{in}} \mid v \in S\}$ (due to Items a) and d)). This then implies that X contains at least $k-1$ vertices of $V(\widetilde{\mathcal{G}})$.

To show that X contains only vertices of V and is a clique in G we now show that the reachability between any two vertices of $V(\widetilde{\mathcal{G}})$ in $\mathcal{G}'$ is the same as in $\widetilde{\mathcal{G}}$. Let P be a temporal path between two distinct vertices of $V(\widetilde{\mathcal{G}})$ in $\mathcal{G}'$. We show that P is also a temporal path in $\widetilde{\mathcal{G}}$. Assume towards a contradiction that this is not the case. Hence, P visits at least one vertex of V'. Since no vertex of $V(\widetilde{\mathcal{G}})$ can reach any vertex of $\{v_{\text{in}} \mid v \in S\}$ (due to Items a) and d)), P visits no vertex of $\{v_{\text{in}} \mid v \in S\}$. Moreover, since each vertex of $V \setminus S$ has only out-neighbors in $\{v_{\text{in}} \mid v \in S\}$, P visits no vertex of $V \setminus S$ either. Consequently, P contains no vertex of V'; a contradiction.

Hence, P is a temporal path in $\widetilde{\mathcal{G}}$, which implies that for each vertex set $Y \subseteq V(\widetilde{\mathcal{G}})$, Y is a tcc in $\widetilde{\mathcal{G}}$ if and only if Y is a tcc in $\mathcal{G}'$. Recall that X contains at least $k-1$ vertices of $V(\widetilde{\mathcal{G}})$. Since the largest tcc in $\widetilde{\mathcal{G}}$ has size at most $k-1$ and each tcc of size $k-1$ in $\widetilde{\mathcal{G}}$ is a clique in G, this implies that $X \cap V(\widetilde{\mathcal{G}})$ is a clique of size $k-1$ in $G[S]$. Since X contains no vertex of $\{v_{\text{in}} \mid v \in S\}$, this implies that X contains exactly one vertex w^* of $V \setminus S$. Hence, it remains to show that each vertex $v \in X \setminus \{w^*\}$ is adjacent to w^* in G. Since X is a tcc in $\mathcal{G}'$, v can reach w^* in $\mathcal{G}'$. By construction and illustrated in Table 1, $\text{out}_v^{\min} \geq 2 \geq \text{in}_{w^*}^{\max}$. Since v reaches w^* and $\mathcal{G}'$ is a proper temporal graph, the arc (v, w^*) is contained in $\mathcal{G}'$. By construction, this implies that v and w^* are adjacent in G. Consequently, X is a clique in G. This completes the correctness proof of the reduction.

Parameter bounds. It thus remains to show that $\delta_{\text{vd}}(\mathcal{G}')$ and the vertex cover of the underlying graph of $\mathcal{G}'$ are at most $|S|^{\mathcal{O}(1)}$ each. Let $V^* := V(\mathcal{G}') \setminus (V \setminus S)$. Note that V^* has size $|V(\widetilde{\mathcal{G}})| + |S| \in \mathcal{O}(|S|^2)$ and is a vertex cover of the underlying graph of $\mathcal{G}'$. Hence, the vertex cover number of the underlying graph of $\mathcal{G}'$ is $\mathcal{O}(|S|^2)$. To show the parameter bounds, it thus suffices to show that V^* is a transitivity modulator of the reachability graph G_R of $\mathcal{G}'$. Due to Claim 10, $G_R - V^* = G_R[V \setminus S]$ is an independent set. Consequently, V^* is a transitivity modulator of G_R. Hence, $\delta_{\text{vd}}(\mathcal{G}') \in \mathcal{O}(|S|^2)$. By the fact that CLIQUE does not admit a polynomial kernel when parameterized by $|S| + k$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$, OPEN-TCC does not admit a polynomial kernel when parameterized by $\delta_{\text{vd}}(\mathcal{G}')$ plus the vertex cover number of the underlying graph of $\mathcal{G}'$ plus k, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. $\blacktriangleleft$

Note that our kernelization lower bounds do not include the non-strict undirected version of OPEN-TCC. An modification of Theorem 9 seems difficult, unfortunately. This is due to the fact that undirected edges can be traversed in both direction, which makes it very difficult to limit the possible reachable vertices in the temporal graph, while preserving a small transitivity modulator.

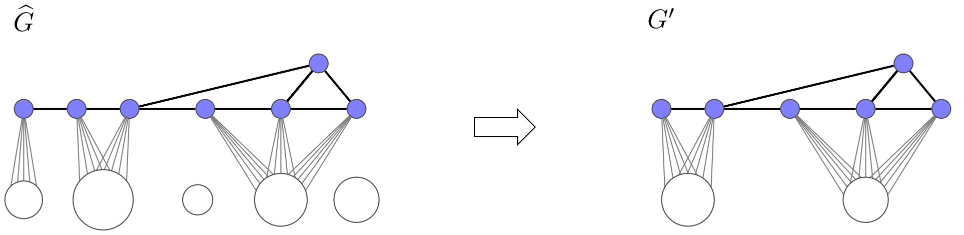


Figure 3 Left: the original instance of Clique from Theorem 12 constructed from the reachability graph of the considered temporal graph. Right: the obtained compressed instance of Clique after exhaustive application of all reduction rules. In both parts, the blue vertices are the vertices from the inherent transitivity modulator B and the cycles at the bottom indicate the white clusters. Note that in both graphs, each blue vertex has neighbors in at most one white cluster (see Claim 14). Intuitively, RR 1 ensures that small clusters are removed, RR 1 and RR 2 ensure that there are no isolated white clusters, and RR 3 reduces the size of each white cluster to at most $|B|+1$.

4 Arc-Modification Distance to Transitivity - A Polynomial Kernel

Next, we focus on the parameterized complexity of Open-TCC when parameterized by the size of a given arc-modification set towards a transitive reachability graph. As discussed earlier, for each arc-modification set M towards a transitive reachability graph, δ_{vd} does not exceed $2 \cdot |M|$, since removing the endpoints of all edges of M results in a transitivity modulator. This implies the following due to Theorem 7 and the fact that a minimum size arc-modification set towards a transitive graph can be computed in $2.57^{\delta_{\text{am}}} \cdot n^{\mathcal{O}(1)}$ time [25].

▶ **Corollary 11.** *Open-TCC can be solved in $4^{\delta_{\text{am}}} \cdot n^{\mathcal{O}(1)}$ time.*

In the remainder of this section, we thus consider this parameter with respect to kernelization algorithms. In contrast to parameterizations by δ_{vd}, we now show that a polynomial kernelization algorithm can be obtained for Open-TCC when parameterized by the size of a given arc-modification set towards a transitive reachability graph.

In fact, we show an even stronger result, since our kernelization algorithm does not need to know the actual arc-modification set but only its endpoints. To formulate this more general result, we need the following definition: Let $G=(V,A)$ be a directed graph. A transitivity modulator $S \subseteq V$ of G is called *inherent*, if there is an arc-modification set M with $M \subseteq S \times S$ for which $(V, A\Delta M)$ is a transitive graph. Note that the set of endpoint of an arc-modification set towards a transitive graph always forms an inherent transitivity modulator.

▶ **Theorem 12.** *Let $I=(\mathcal{G},k)$ be an instance of Open-TCC and let $G_R=(V,A)$ be the reachability graph of $\mathcal{G}$. Moreover, let $B \subseteq V$ be an inherent transitivity modulator of G_R. Then, for each version of Open-TCC, one can compute in polynomial time an equivalent instance of total size $\mathcal{O}(|B|^3)$.*

Proof. We first present a compression to Clique. Let $\widehat{G}=(V,E)$ be an undirected graph that contains an edge $\{u,v\}$ if and only if (u,v) and (v,u) are arcs of G_R. Due to Lemma 2, I is a yes-instance of Open-TCC if and only if $(\widehat{G},k)$ is a yes-instance of Clique. Let $W := V \setminus B$. We call the vertices of B *blue* and the vertices of W *white*. Note that $G_R[W]$ is a transitive graph, since B is a transitivity modulator of G_R. Moreover, there exists an arc set $M \subseteq B \times B$ such that $G'_R=(V, A\Delta M)$ is transitive, since B is an inherent transitivity modulator of G_R.

In the following, we present reduction rules to remove vertices from $\widehat{G}$ to obtain an equivalent instance (G', k') of Clique with $\mathcal{O}(|B|^2)$ vertices and where G' is an induced subgraph of $\widehat{G}$. The graphs $\widehat{G}$ and G' are conceptually depicted in Figure 3.

To obtain this smaller instance of Clique, we initialize G' as a copy of $\widehat{G}$ and k' as k and exhaustively applying three reduction rules. Our first two reduction rules are the following:

- RR 1: Remove a vertex v from G', if v has degree less than $k' - 1$ in G'.
- RR 2: If a white vertex has at least $k' - 1$ white neighbors in G', output a constant size yes-instance.

Note that the first reduction rule is safe, since no vertex of degree less than $k' - 1$ can be part of a clique of size at least k'. Moreover, each connected component in G' has size at least k' after this reduction rule is exhaustively applied. The safeness of the second reduction rule relies on the following observation.

▷ Claim 13. If two white vertices u and v are adjacent in G', then they are real twins in G'. That is, $N_{G'}[u] = N_{G'}[v]$.

Proof. Assume that u and v are adjacent in G' and assume towards a contradiction that there is a vertex w in G' which is adjacent to u in G' but not adjacent to v in G'. Since w and v are not adjacent in G', G_R contains at most one of the arcs (w, v) or (v, w). Assume without loss of generality that (w, v) is not an arc of G_R. Since u is adjacent to both v and w in G', G_R contains the arcs (v, u) and (u, w). Recall that both u and v are white vertices. This implies that the arc-modification set M contains no arc incident with any of these two vertices. Hence, M contains none of the arcs of $\{(v, u), (u, w), (v, w)\}$, which implies that $G'_R = (V, A\Delta M)$ is not a transitive graph; a contradiction. ◁

Note that this implies that each connected component in $G'[W]$ is a clique of real twins in G'. We call each such connected component in $G'[W]$ a *white cluster*.

Note that after exhaustive applications of the first two reduction rules, each white cluster has size at most $k' - 1$ and each connected component in G' has size at least k'. This implies that each connected component in G' contains at least one blue vertex. Since G' contains at most $|B|$ blue vertices, this implies that G' has at most $|B|$ connected components.

In the following, we show that no blue vertex has neighbors in more than one white cluster. This then implies that G' contains at most $|B| \cdot k'$ vertices. In a final step, we then show how to reduce the value of k'.

▷ Claim 14. No blue vertex has neighbors in more than one white cluster.

Proof. Assume towards a contradiction that there is a blue vertex w which is adjacent to two white vertices u and v in G', such that u and v are not part of the same white cluster. Since u and v are not part of the same white cluster, u and v are not adjacent in G' due to Claim 13. This implies that G_R contains at most one of the arcs (u, v) or (v, u). Assume without loss of generality that (u, v) is not an arc of G_R. Since w is adjacent to both u and v in G', G_R contains the arcs (u, w) and (w, v). By the fact that both u and v are white vertices, M contains no arc of $\{(u, w), (w, v), (u, v)\}$, which implies that $G'_R = (V, A\Delta M)$ is not a transitive graph; a contradiction. ◁

As mentioned above, this implies that G' contains at most $|B| \cdot k'$ vertices. Next, we show how to reduce the size of the white clusters if $k' > |B|$. To this end, we introduce our last reduction rule:

- RR 3: If $k' > |B| + 1$, remove an arbitrary white vertex from each white cluster and reduce k' by 1.

Note that RR 3 is safe: If $k' > |B| + 1$, a clique of size k' in G' has to contain at least two white vertices, since G' contains at most $|B|$ blue vertices. Since no clique in G' can contain vertices of different white clusters, we reduce the size of a maximal clique of size at least k' in G' by exactly one, when removing one vertex of each white cluster.

Hence, after all reduction rules are applied exhaustively, the resulting instance (G', k') of Clique contains at most $|B|$ blue vertices and at most $|B|$ white clusters. Each such white cluster hast size at most $|B| + 1$. This implies that the resulting graph G' contains $\mathcal{O}(|B|^2)$ vertices and $\mathcal{O}(|B|^3)$ edges, since each vertex has degree $\mathcal{O}(|B|)$.

Based on a known polynomial-time reduction [6], we can compute for an instance (G^*, k^*) of Clique, an equivalent instance $(\mathcal{G}^*, k^*)$ of Open-TCC, where $\mathcal{G}^*$ is a proper temporal graph and has $\mathcal{O}(n + m)$ vertices and edges, where n and m denote the number of vertices and the number of edges of G^*, respectively. Since G' has $\mathcal{O}(|B|^2)$ vertices and $\mathcal{O}(|B|^3)$ edges, this implies that we can obtain an equivalent instance of Open-TCC of total size $\mathcal{O}(|B|^3)$ in polynomial time. By the fact that $\mathcal{G}^*$ is a proper temporal graph, this works for all problem versions of Open-TCC. ◀

Based on the fact that the set B of endpoints of any arc-modification set M towards a transitive graph is an inherent transitivity modulator of size at most $2 \cdot |M|$, this implies the following for kernelization algorithms with respect to arc-modification sets towards a transitive graph.

▶ **Theorem 15.** *Let $I = (\mathcal{G}, k)$ be an instance of Open-TCC and let $G_R = (V, A)$ be the reachability graph of $\mathcal{G}$. Moreover, let $M \subseteq V \times V$ be a set of arcs such that $G'_R = (V, A \Delta M)$ is transitive. Then, for each version of Open-TCC, one can compute in polynomial time an equivalent instance of total size $\mathcal{O}(|M|^3)$.*

Moreover, if the arc-modification set M only adds arcs to the reachability graph, we can obtain the further even better kernelization result.

▶ **Lemma 16** (⋆). *Let $I = (\mathcal{G}, k)$ be an instance of Open-TCC and let $G_R = (V, A)$ be the reachability graph of $\mathcal{G}$. Moreover, let $M \subseteq V \times V$ be a set with $A \cap M = \emptyset$ of arcs such that $G'_R = (V, A \Delta M)$ is transitive. Then, for each version of Open-TCC, one can compute in polynomial time an equivalent instance of total size $\mathcal{O}(|M|^2)$.*

Hence, if we are given an arc-modification set M of size δ_{am}, we can compute a polynomial kernel. Unfortunately, finding a minimum-size arc-modification set of a given directed graph is NP-hard [25] and no polynomial-factor approximations are known that run in polynomial time. Hence, we cannot derive a polynomial kernel for the parameter δ_{am}. Positively, if we only consider arc-additions, we can compute the transitive closure of a given directed graph in polynomial time. This implies that we can find a minimum-size arc-modification set towards a transitive reachability graph in polynomial time among all such sets that only add arcs to to reachability graph. Consequently, we derive the following.

▶ **Corollary 17.** *Open-TCC admits a kernel of size $\mathcal{O}(\delta_{\text{aa}}^2)$, where δ_{aa} denotes the minimum number of necessary arc-additions to make the respective reachability graph transitive.*

5 Limits of these Parametrizations for Closed TCCs

So far, the temporal paths that realize the reachability between two vertices in a tcc could lie outside of the temporal connected component. If we impose the restriction that those temporal paths must be contained in the tcc, the problem of finding a large tcc becomes

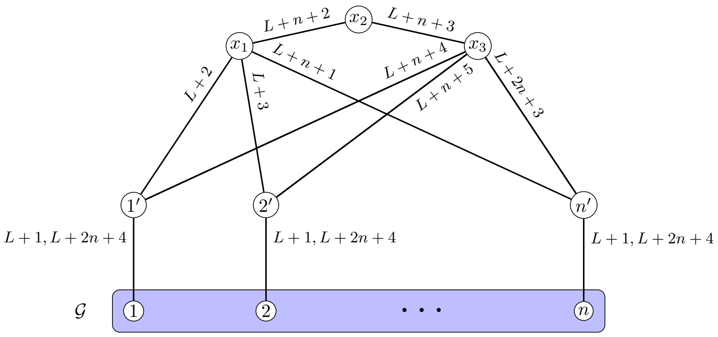


Figure 4 An illustration of the additional vertices and edges that are added to $\mathcal{G}$ in the reduction of Theorem 18. Here, L denotes the lifetime of $\mathcal{G}$ and the labels on the edges indicate in which snapshots the respective edges exist in the constructed temporal graph.

NP-hard even when the reachability graph is missing only a single arc to become a complete bidirectional clique: In other words, the problem becomes NP-hard even if $\delta_{\text{vd}} = \delta_{\text{am}} = 1$. On general temporal graphs, all versions of CLOSED-TCC are known to be NP-hard [8, 13].

▶ **Theorem 18.** *For each version of CLOSED-TCC, there is a polynomial time self-reduction that transforms an instance $(\mathcal{G}, k)$ with $k > 4$ of that version of CLOSED-TCC into an equivalent instance $(\mathcal{G}', k)$, such that the reachability graph of $\mathcal{G}'$ is missing only a single arc to be a complete bidirectional clique.*

Proof. Let $I := (\mathcal{G}, k)$ be an instance of CLOSED-TCC with underlying graph $G = (V, E)$ and let L be the lifetime of $\mathcal{G}$. Moreover, assume for simplicity that the vertices of V are exactly the natural numbers from $[1, n]$ with $n := |V|$. To obtain an equivalent instance $I' := (\mathcal{G}', k)$ of CLOSED-TCC, we extends $\mathcal{G}$ as follows: We initialize $\mathcal{G}'$ as a copy of $\mathcal{G}$ and for each vertex $v \in V$, we add a new vertex v' to $\mathcal{G}'$. Additionally, we add three vertices x_1, x_2, and x_3 to $\mathcal{G}'$. Furthermore, we append $2n + 4$ empty snapshots to the end of $\mathcal{G}'$ and add the following edges to $\mathcal{G}'$: For each vertex $v \in V$, we add the edge $\{v, v'\}$ to time steps $L + 1$ and $L + 2n + 4$, the edge $\{v', x_1\}$ to time step $L + 1 + v$, and the edge $\{v', x_3\}$ to time step $L + n + 3 + v$. Finally, we add the edge $\{x_1, x_2\}$ to time step $L + n + 2$ and the edge $\{x_2, x_3\}$ to time step $L + n + 3$. This completes the construction of $\mathcal{G}'$. An illustration of the additional vertices and edges is given in Figure 4.

Before we show the equivalence between the two instances of CLOSED-TCC, we first show that the reachability graph G'_R of $\mathcal{G}'$ only misses a single arc to be a bidirectional clique.

▷ Claim 19. The arc (x_3, x_1) is the only arc that is missing in G'_R.

Proof. First, we show that (x_3, x_1) is not an arc of G'_R. By construction of $\mathcal{G}'$, (i) no edge of $\mathcal{G}'$ that is incident with x_1 exists in any time step larger than $L + n + 2$, and (ii) no edge of $\mathcal{G}'$ that is incident with x_3 exists in any time step smaller than $L + n + 3$. This implies that no temporal path in $\mathcal{G}'$ that starts in x_3 can reach x_1. Hence, (x_3, x_1) is not an arc of G'_R.

Next, we show that G'_R contains all other possible arcs. To this end, we present strict temporal paths in $\mathcal{G}'$ that guarantee the existence of these arcs in G'_R. Let u and v be vertices of V. Consider the temporal path P that starts in vertex u and traverses

- the edge $\{u, u'\}$ in time step $L + 1$,
- the edge $\{u', x_1\}$ in time step $L + 1 + u$,
- the edge $\{x_1, x_2\}$ in time step $L + n + 2$,
- the edge $\{x_2, x_3\}$ in time step $L + n + 3$,

- the edge $\{x_3, v'\}$ in time step $L + n + 3 + v$, and
- the edge $\{v', v\}$ in time step $L + 2n + 4$.

Since each vertex of V is a natural number of $[1, n]$, the times steps in which these edges are traversed by P are strictly increasing, which implies that P is a strict temporal path of $\mathcal{G}'$. Note that each suffix and each prefix of P is also a strict temporal path in $\mathcal{G}'$. Hence, this implies that G'_R contains all the arcs $\{(u, v'), (u, v), (u', v'), (u', v)\} \cup \{(u', x_1), (u', x_1), (x_1, v'), (x_1, v) \mid i \in \{1, 2, 3\}\} \cup \{(x_1, x_3)\}$. Moreover, since $\{x_1, x_2\}$ and $\{x_2, x_3\}$ are edges in $\mathcal{G}'$, G'_R also contains the arcs $\{(x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2)\}$. This implies that (x_3, x_1) is the unique arc that is missing in G'_R. ◁

Next, we show that I is a yes-instance of Closed-TCC if and only if I' is a yes-instance of Closed-TCC.

(⇒) This direction follows directly by the fact that $\mathcal{G}'$ is obtained by extending $\mathcal{G}$. Hence, a closed tcc S of size k in $\mathcal{G}$ is also a closed tcc in $\mathcal{G}'$.

(⇐) Let S be a closed tcc of size k in $\mathcal{G}'$. Recall that $k > 4$. We show that S only contains vertices of V. To this end, we first show that S does not contain x_2.

▷ Claim 20 (⋆). The set S does not contain x_2.

Next, we show that the vertex x_2 is required to have pairwise temporal paths between distinct vertices of $\{w' \mid w \in V\}$ in $\mathcal{G}'$.

▷ Claim 21 (⋆). Let u and v be distinct vertices of V with $u < v$. Then, each temporal path from v' to u' in $\mathcal{G}'$ visits x_2.

As a consequence, S contains at most one vertex of $\{w' \mid w \in V\}$, since S is a closed tcc that does not contain vertex x_2. Based on the above two claims, we now show that S contains only vertices of V.

▷ Claim 22 (⋆). Only vertices of V are contained in S.

Since no edge between any two vertices of V was added while constructing $\mathcal{G}'$ from $\mathcal{G}$, each temporal path in $\mathcal{G}'$ that visits only vertices of V is also a temporal path in $\mathcal{G}$. Together with Claim 22, this implies that S is a closed tcc in $\mathcal{G}$, which implies that I is a yes-instance of Closed-TCC. ◀

Recall that all versions of Closed-TCC are NP-hard [8, 13]. Moreover the strict undirected version of Closed-TCC is W[1]-hard when parameterized by k [8] and both directed versions of Closed-TCC are W[1]-hard when parameterized by k [13]. Together with Theorem 18, this implies the following intractability results for Closed-TCC.

▶ **Theorem 23.** *All versions of Closed-TCC are NP-hard even if $\delta_{\text{vd}} = \delta_{\text{am}} = 1$. More precisely, this hardness holds on instances where the reachability graph is missing only a single arc to be a complete bidirectional clique. Excluding the undirected non-strict version of Closed-TCC, all versions of Closed-TCC are W[1]-hard when parameterized by k under these restrictions*

6 Conclusion

We introduced two new parameters δ_{vd} and δ_{am} that capture how far the reachability graph of a given temporal graph is from being transitive. We demonstrated their applicability when the goal is to find open tccs in a temporal graph, presenting FPT-algorithms for each parameter individually, and a polynomial kernel with respect to δ_{am}, assuming that the corresponding arc-modification set of size δ_{am} is given. Computing such a set is NP-hard in general directed graphs [25]. An interesting question, also formulated in that paper, is whether this parameter is at least approximable to within a polynomial factor of δ_{am}. If so, our result implies a polynomial kernel for Open-TCC when parameterized by δ_{am}. Alternatively, the existence of a proper polynomial kernel for Open-TCC when parameterized by δ_{am} could also be shown by finding an approximation for a minimum-size inherent transitivity modulator due to Theorem 12 and the fact that the size of a minimum-size inherent transitivity modulator never exceeds $2 \cdot \delta_{\text{am}}$.

Another natural question is to identify what are other (temporal) reachability problems for which our transitivity parameters could be useful. For instance, consider a variant of the Open-TCC problem where we search for *d-tccs*, that is, tccs such that the fastest temporal path between the vertices has duration at most d. It is plausible that our positive results carry over to this version when applied to the *d-reachability graph*, i.e., the graph whose arcs represent temporal paths of duration at most d.

Regarding Closed-TCC, our intractability results show that neither δ_{vd} nor δ_{am} suffice to make this problem tractable. However, our results do not preclude the existence of an FPT-algorithm in the case that the arc modification operations are restricted to deletion only, which remains to be investigated. Nonetheless, we still believe that transitivity is a key aspect of the problem. The problem with Closed-TCC is that the reachability graph itself does not encode whether the paths responsible for reachability travel through internal or external vertices.

We would like to initiate the idea of considering further transitivity parameters based on modifications of the temporal graph itself, not only of the reachability graph. In particular, could FPT-algorithms for such parameters be achieved for reachability problems such as Open-TCC with similar performance as our FPT-algorithms for δ_{vd} and δ_{am}, and could these parameters make Closed-TCC tractable as well? And if so, how difficult is the computation of such parameters?

References

1 Eleni C Akrida, Jurek Czyzowicz, Leszek Gasieniec, Łukasz Kuszner, and Paul G Spirakis. Temporal flows in temporal networks. *Journal of Computer and System Sciences*, 103:46–60, 2019.

2 Eleni C Akrida, Leszek Gąsieniec, George B Mertzios, and Paul G Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61:907–944, 2017.

3 Eleni C Akrida, George B Mertzios, Paul G Spirakis, and Christoforos Raptopoulos. The temporal explorer who returns to the base. *Journal of Computer and System Sciences*, 120:179–193, 2021.

4 Emmanuel Arrighi, Fedor V. Fomin, Petr A. Golovach, and Petra Wolf. Kernelizing temporal exploration problems. In Neeldhara Misra and Magnus Wahlström, editors, *18th International Symposium on Parameterized and Exact Computation, IPEC 2023, September 6-8, 2023, Amsterdam, The Netherlands*, volume 285 of *LIPIcs*, pages 1:1–1:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.1.

5 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

6 Sandeep Bhadra and Afonso Ferreira. Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *Ad-Hoc, Mobile, and Wireless Networks: Second International Conference, ADHOC-NOW2003, Montreal, Canada, October 8-10, 2003. Proceedings 2*, pages 259–270. Springer, 2003.

7 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. *Algorithmica*, 85(3):688–716, 2023.

8 Arnaud Casteigts. Finding structure in dynamic networks. *CoRR*, abs/1807.07801, 2018. arXiv:1807.07801.

9 Arnaud Casteigts and Timothée Corsini. In search of the lost tree: Hardness and relaxation of spanning trees in temporal graphs. In *31th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2024.

10 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.

11 Arnaud Casteigts, Michael Raskin, Malte Renken, and Viktor Zamaraev. Sharp thresholds in random simple temporal graphs. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–326. IEEE Computer Society, 2022.

12 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.

13 Isnard Lopes Costa, Raul Lopes, Andrea Marino, and Ana Silva. On Computing Large Temporal (Unilateral) Connected Components. In *Combinatorial Algorithms - 34th International Workshop, IWOCA 2023, Tainan, Taiwan, June 7-10, 2023, Proceedings*, volume 13889 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2023. doi:10.1007/978-3-031-34347-6_24.

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

15 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

16 Jessica Enright, Kitty Meeks, George B Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. In *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019).* Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

17 Thomas Erlebach and Jakob T Spooner. Parameterized temporal exploration problems. In *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022).* Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

18 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. As time goes by: reflections on treewidth for temporal graphs. In *Treewidth, Kernels, and Algorithms: Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, pages 49–77. Springer, 2020.

19 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020.

20 Niels Grüttemeier and Christian Komusiewicz. On the Relation of Strong Triadic Closure and Cluster Deletion. *Algorithmica*, 82(4):853–880, 2020.

21 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 504–513, 2000.

22 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

23 Jason Schoeters, Eric Sanlaville, Stefan Balev, and Yoann Pigné. Temporally connected components. *Available at SSRN 4590651*, 2024.

24 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

25 Mathias Weller, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. On making directed graphs transitive. *Journal of Computer and System Sciences*, 78(2):559–574, 2012. doi:10.1016/J.JCSS.2011.07.001.

Quasi-Isometric Reductions Between Infinite Strings

Karen Frilya Celine
School of Computing, National University of Singapore, Singapore

Ziyuan Gao
School of Science and Technology, Singapore University of Social Sciences, Singapore
Kaplan Higher Education Academy, Singapore

Sanjay Jain
School of Computing, National University of Singapore, Singapore

Ryan Lou
School of Computing, National University of Singapore, Singapore

Frank Stephan
Department of Mathematics, National University of Singapore, Singapore
School of Computing, National University of Singapore, Singapore

Guohua Wu
School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore

Abstract

This paper studies the recursion-theoretic aspects of large-scale geometries of infinite strings, a subject initiated by Khoussainov and Takisaka (2017). We investigate several notions of quasi-isometric reductions between recursive infinite strings and prove various results on the equivalence classes of such reductions. The main result is the construction of two infinite recursive strings α and β such that α is strictly quasi-isometrically reducible to β, but the reduction cannot be made recursive. This answers an open problem posed by Khoussainov and Takisaka.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases Quasi-isometry, recursion theory, infinite strings

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.37

Related Version *Full Version*: https://arxiv.org/abs/2407.14105

Funding S. Jain and F. Stephan were supported by Singapore Ministry of Education (MOE) AcRF Tier 2 grant MOE-000538-00. Additionally, S. Jain was supported by NUS grant E-252-00-0021-01 and F. Stephan was supported by the AcRF Tier 1 grants A-0008454-00-00 and A-0008494-00-00. G. Wu was partially supported by MOE Tier 1 grants RG111/19 (S) and RG102/23.

Acknowledgements We thank the anonymous referees for several helpful comments which improved the presentation of the paper.

1 Introduction

Quasi-isometry is an important concept in geometric group theory that has been used to solve problems in group theory. Loosely speaking, two metric spaces are said to be quasi-isometric iff there is a mapping (called a *quasi-isometry*) from one metric space to the other that preserves the distance between any two points in the first metric space up to some multiplicative and additive constants. Thus, for example, while the Euclidean plane is not isometric to $\mathbb{R}^2$ equipped with the taxicab distance, the two spaces are quasi-isometric to each other since the Euclidean distance between any two points does not differ from the

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 37; pp. 37:1–37:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

taxicab distance between them up to a multiplicative factor of $\sqrt{2}$. The study of group properties – where groups are represented by their Cayley graphs – that are invariant under quasi-isometries is quite a prominent theme in geometric group theory; examples of such group properties include hyperbolicity and growth rate [2].

This paper studies quasi-isometries of the ordered sets $(\mathbb{N}, <)$ with the objects being infinite strings, recursive functions from $\mathbb{N}$ to a finite alphabet or isomorphic copies of these structures defined with automatic functions in automata theory replacing recursive ones (the latter being delayed to the journal version of this paper). The notion of quasi-isometry for infinite strings was introduced by Khoussainov and Takisaka [6], enabling the study of global patterns on strings and linking the study of large-scale geometries with automata theory, computability theory, algorithmic randomness and model theory. Furthermore, quasi-isometries between hyperbolic metric spaces in general – an example of which is an infinite string when viewed as a colored metric space – are well-studied in geometric group theory. Isometries between computable metric spaces have also been studied by Melnikov [10].

Among the various questions investigated by Khoussainov and Takisaka was the computational complexity of the *quasi-isometry problem*: given any two infinite strings α and β, is there a quasi-isometry from α to β? They found that for any two quasi-isometric strings, a quasi-isometry that is recursive in the halting problem relative to α and β always exists between them, and that the quasi-isometry problem between any two recursive strings is Σ^0_2-complete [7][1]. In comparison, the corresponding problem for isometry with respect to recursive strings is Π^0_1-complete [10]. Khoussainov and Takisaka also had the following open problem which was mentioned in many talks and discussions: *if a quasi-isometric reduction from α to β exists, does there always exist a recursive quasi-isometric reduction?* This is a very natural question for computer science, specifically for computability theory, since it seeks to understand how complex such a reduction is. We answered this question in the negative, that is, there are cases where the reduction exists but cannot be made recursive. The fourth author's bachelor thesis [9] which contains this result was cited by Khoussainov and Takisaka in the journal version [7] of their paper [6].

To complete the picture, the present work examines, in more detail, the recursion-theoretic aspects of quasi-isometries between infinite strings. We study various natural restrictions on quasi-isometric reductions between strings: first, *many-one reductions*, where the quasi-isometric reduction is required to be *recursive* and *many-one*; second, *one-one reductions*, which are injective many-one reductions; third, *permutation quasi-isometric reductions*, which are surjective one-one reductions.

The main subjects of this work are the structural properties of the equivalence classes induced by the different types of reductions and the relationships between these reductions. In accordance with recursion-theoretic terminology, we call an equivalence class induced by a reduction type a *degree* of that reduction type. We show, for example, that within each many-one quasi-isometry degree, any pair of strings has a common upper bound as well as a common lower bound with respect to one-one reductions. Furthermore, there are two strings for which their many-one quasi-isometry degrees have a unique least common upper bound. The main result is the separation of quasi-isometry from *recursive* quasi-isometry, that is, we construct two recursive strings such that one is quasi-isometric reducible to the other but no recursive many-one quasi-isometry exists between them. This main result answers the above-mentioned open problem posed by Khoussainov and Takisaka.

[1] Note that [7] is the journal version of their paper [6], containing some corrections from the earlier paper.

2 Notation

Any unexplained recursion-theoretic notation may be found in [11, 13, 14]. The set of positive integers will be denoted by $\mathbb{N}$; $\mathbb{N} \cup \{0\}$ will be denoted by $\mathbb{N}_0$. The finite set Σ will denote the alphabet used. We assume knowledge of elementary computability theory over different size alphabets [1]. An infinite string $\alpha \in \Sigma^\omega$ can also be viewed as a Σ-valued function defined on $\mathbb{N}$. The length of an interval I is denoted by $|I|$. For $\alpha_i \in \Sigma^*$ and $i \in \mathbb{N}$, we write $(\alpha_i)_{i=1}^{\infty}$ to denote $\alpha_1\alpha_2\cdots$, a possibly infinite string.

3 Colored Metric Spaces and Infinite Strings

▶ **Definition 1** (Colored Metric Spaces, [6]). *A* colored metric space $(M; d_M, Cl)$ *consists of the* underlying metric space $(M; d_M)$ *with metric d_M and the color function $Cl : M \to \Sigma$, where Σ is a finite set of colors called an* alphabet*. We say that $m \in M$* has color $\sigma \in \Sigma$ *if $\sigma = Cl(m)$.*

▶ **Definition 2** (Quasi-isometries Between Colored Metric Spaces, [6]). *For any $A \geq 1$ and $B \geq 0$, an* (A, B)-quasi-isometry *from a metric space $\mathcal{M}_1 = (M_1; d_1)$ to a metric space $\mathcal{M}_2 = (M_2; d_2)$ is a function $f : M_1 \to M_2$ such that for all $x, y \in M_1$, $\frac{1}{A} \cdot d_1(x, y) - B \leq d_2(f(x), f(y)) \leq A \cdot d_1(x, y) + B$, and for all $y \in M_2$, there exists an $x \in M_1$ such that $d_2(f(x), y) \leq A$.*

Given two colored metric spaces $\mathcal{M}_1 = (M_1; d_1, Cl_1)$ and $\mathcal{M}_2 = (M_2; d_2, Cl_2)$, a function $f : M_1 \to M_2$ is a quasi-isometric reduction *from $\mathcal{M}_1$ to $\mathcal{M}_2$ iff for some $A \geq 1$ and $B \geq 0$, f is an (A, B)-quasi-isometry from $(M_1; d_1)$ to $(M_2; d_2)$ and f is* color-preserving*, that is, for all $x \in M_1$, $Cl_1(x) = Cl_2(f(x))$.*

An infinite string α can then be seen as a colored metric space $(\mathbb{N}; d, \alpha)$, where d is the metric on $\mathbb{N}$ defined by $d(i, j) = |i - j|$ and $\alpha : \mathbb{N} \to \Sigma$ is the color function. For any two infinite strings α and β, we write $\alpha \leq_{qi} \beta$ to mean that there is a quasi-isometric reduction from α to β. The relation $\leq_{qi}$ is a preorder on Σ^ω. For any pair of distinct letters $a_1, a_2 \in \Sigma$, a_1^ω and a_2^ω are incomparable with respect to $\leq_{qi}$, so this relation is not total.

The following proposition gives a useful simplification of the definition of quasi-isometry in the context of infinite strings.

▶ **Proposition 3.** *Given two infinite strings α and β, let $f : \mathbb{N} \to \mathbb{N}$ be a color-preserving function. Then f is a quasi-isometric reduction from α to β iff there exists a constant $C \geq 1$ such that for all x, y in the domain of α, the following conditions hold:*

(a) $d(f(x), f(x+1)) \leq C$;

(b) $x + C < y \Rightarrow f(x) < f(y)$.

Proof. First, suppose that $f : \mathbb{N} \to \mathbb{N}$ is a color-preserving quasi-isometric reduction from α to β. We show that there exists a constant $C \geq 1$ for which Conditions (a) and (b) hold for any $x, y \in \mathbb{N}$. By the definition of a quasi-isometric reduction, there exist constants $A \geq 1$ and $B \geq 0$ such that

$$\frac{1}{A} \cdot d(x, y) - B \leq d(f(x), f(y)) \leq A \cdot d(x, y) + B. \tag{1}$$

We first derive, for each of the two conditions, a choice of C satisfying it.

(i) Plugging $y = x + 1$ into the upper bound in (1) yields $d(f(x), f(x + 1)) \leq A + B$.

(ii) Assume for the sake of a contradiction that for all $C \geq 1$, there are $x \in \mathbb{N}$ and $C' > C$ such that $f(x + C') \leq f(x)$. We show that if C is chosen so that $A + B \leq \frac{1}{A} \cdot C - B$, then the existence of some $C' > C$ with $f(x + C') \leq f(x)$ would lead to a contradiction. Fix such a C, and suppose there were indeed some C' with $C' > C \geq 1$ and

$$f(x + C') \leq f(x)\,. \tag{2}$$

Then,

$$\begin{aligned} f(x + C' + 1) - f(x + C') &\leq d(f(x + C' + 1), f(x + C')) \\ &\leq A + B \quad \text{(by statement (i))} \\ &\leq \frac{1}{A} \cdot C - B \quad \text{(by the choice of } C\text{)} \\ &< \frac{1}{A} \cdot C' - B \quad \text{(since } C' > C\text{)} \\ &\leq f(x) - f(x + C') \quad \text{(by (1) and (2))}\,, \end{aligned}$$

giving $f(x + C' + 1) < f(x)$. One can repeat the preceding argument inductively, yielding the inequality $f(x + C' + k + 1) - f(x + C' + k) < f(x) - f(x + C' + k)$, or equivalently $f(x + C' + k + 1) < f(x)$, for each $k \geq 0$. But this is impossible since $f(x)$ is finite and $d(f(x + C' + k + 1), f(x + C' + k' + 1)) > 0$ whenever $|k - k'|$ is sufficiently large.

It follows from (i) and (ii) that Conditions (a) and (b) are satisfied for $C = A \cdot (A + 2B)$.

For a proof of the converse direction, fix a C satisfying Conditions (a) and (b). Suppose $x \in \mathbb{N}$. Then by Condition (a), $d(f(x), f(x + 1)) \leq C$. Inductively, assume that $d(f(x), f(x + n)) \leq n \cdot C$. Then by the inductive hypothesis and Condition (a), $d(f(x), f(x + n + 1)) \leq d(f(x), f(x+n)) + d(f(x+n), f(x+n+1)) \leq n \cdot C + C = (n+1) \cdot C$ where the first inequality follows from the triangle inequality. Consequently, for all $x, y \in \mathbb{N}$,

$$d(f(x), f(y)) \leq d(x, y) \cdot C\,. \tag{3}$$

Next, we establish a lower bound for $d(f(x), f(y))$. Without loss of generality, assume $x < y$. Write $y = x + i(C + 1) + j$ for some $i \in \mathbb{N}_0$ and $0 \leq j \leq C$. By a simple induction, one can show that $f(x + i(C + 1)) \geq f(x) + i$ and thus $d(f(x), f(x + i(C + 1)) \geq i$. Furthermore, $d(f(x + i(C + 1)), f(y)) \leq C^2$. Thus $d(f(x), f(y)) \geq i - C^2$ and $i \geq d(x, y)/(C + 1) - 1$. It follows that $d(f(x), f(y)) \geq d(x, y)/(C + 1) - 1 - C^2$. Thus one can select $A = (C + 1)$ and $B = C^2 + 1$ to establish the required bounds for the quasi-isometric mapping.

To establish that for all $y \in M_2$, there exists an $x \in M_1$ such that $d_2(f(x), y) \leq A$, one can choose any $A \geq max(C, f(1))$, as the distance between $f(x)$ and $f(x + 1)$ is bounded by C. ◀

By Proposition 3, we can now redefine quasi-isometric reduction in terms of one constant C, instead of two constants A and B as in Definition 2, reducing the number of constants by 1.

▶ **Definition 4.** *Suppose $C \geq 1$. Given infinite strings α and β, a C-*quasi-isometry *from α to β is a color-preserving function $f : \mathbb{N} \to \mathbb{N}$ such that for all x, y in the domain of α,*

(a) *$f(1) \leq C$ and $f(x) - C \leq f(x + 1) \leq f(x) + C$.*

(b) *$x + C < y \Rightarrow f(x) < f(y)$.*

For the rest of the paper, we shall use "Condition (a)" and "Condition (b)" to refer to the above conditions respectively, without necessarily mentioning the definition number.

A useful property of a C-quasi-isometry f from α to β is that any position of β has at most $C+1$ pre-images under f.

▶ **Lemma 5** ([6, Corollary II.4]). *Given two infinite strings α and β, suppose that f is a C-quasi-isometry from α to β. Then for all $y \in \mathbb{N}$, $|f^{-1}(y)| \leq C+1$.*

It was proven earlier that for any infinite strings α, β and any C-quasi-isometry f from α to β, there is a constant D such that each position of β is at most D positions away from some image of f. The next lemma states that each position of β in the range of f is at most C positions away from a different image of f. The proof is omitted due to space restrictions.

▶ **Lemma 6.** *Given two infinite strings α and β, suppose that f is a C-quasi-isometry from α to β. Then $\min\{f(x) : x \in \mathbb{N}\} \leq C$ and for each $y \in \mathbb{N}$, $min\{f(x) : x \in \mathbb{N}$ and $f(x) > f(y)\} \leq f(y) + C$. Hence for each $z \in \mathbb{N}$, there is some $x \in \mathbb{N}$ such that $d(f(x), z) \leq C$.*

▶ **Corollary 7.** *Let $\Sigma = \{a_1, \ldots, a_l\}$ and let α, β be two infinite strings. Let f be a C-quasi-isometry from α to β. Suppose that there is a positive integer K such that there is at least one occurrence of a_i in any interval of positions of α of length K. Then there is at least one occurrence of a_i in any interval of positions of β of length KC.*

A quasi-isometry f can fail to be order-preserving in that there are pairs $x, y \in \mathbb{N}$ with $x < y$ and $f(x) > f(y)$. Nonetheless, as Khoussainov and Takisaka noted [6, Lemma II.2], every quasi-isometry enforces a uniform upper bound on the size of a *cross-over* – the difference $f(x) - f(y)$ for such a pair $x, y \in \mathbb{N}$.

▶ **Lemma 8** (Small Cross-Over Lemma, [6, Lemma II.2]). *Given two infinite strings α and β, suppose that f is a C-quasi-isometry from α to β. Then for all $n, m \in \mathbb{N}$ with $n < m$, we have $f(n) - f(m) \leq C^2$.*

4 Recursive Quasi-Isometric Reductions

Khoussainov and Takisaka [6] investigated the structure of the partial-order Σ^{ω}_{qi} of the quasi-isometry degrees over an alphabet $\Sigma = \{a_1, \ldots, a_l\}$. They proved that Σ^{ω}_{qi} has a greatest element, namely the degree of $(a_1 \cdots a_n)^{\omega}$, and that Σ^{ω}_{qi} contains uncountably many minimal elements. Furthermore, they showed that Σ^{ω}_{qi} includes a chain of the type of the integers, and that it includes an antichain. In connection with computability theory, in particular with the arithmetical hierarchy, they established that the quasi-isometry relation on recursive infinite strings is Σ^0_2-complete [7]. In this section, we continue research into the recursion-theoretic aspects of quasi-isometries on infinite strings. We consider the notions of many-one and one-one recursive reducibilities first introduced by Post [12] as relations between recursive functions, and apply them to quasi-isometric reductions. We also define a third type of quasi-isometric reducibility – permutation reducibility – which is bijective. We then prove a variety of results on the degrees of such reductions.

▶ **Definition 9** (Many-One Reducibility). *A string α is* many-one reducible*, or* mqi-reducible*, to a string β iff there exists a quasi-isometric reduction f from α to β such that f is* recursive*. We call such an f a* many-one quasi-isometry (or mqi-reduction)*, and write $\alpha \leq_{mqi} \beta$ to mean that α is many-one reducible to β; if, in addition, f is a C-quasi-isometry, then we call f a* C-many-one quasi-isometry *(or* C-mqi-reduction*). We write $\alpha <_{mqi} \beta$ to mean that $\alpha \leq_{mqi} \beta$ and $\beta \not\leq_{mqi} \alpha$.*

▶ **Definition 10** (One-One Reducibility). *A string α is* one-one reducible*, or* 1qi-reducible*, to a string β iff there exists a many-one quasi-isometry f from α to β such that f is one-one. We call such an f a* one-one quasi-isometry (or 1qi-reduction)*, and write $\alpha \leq_{1qi} \beta$ to mean that α is one-one reducible to β; if, in addition, f is a C-quasi-isometry, then we call f a* C-one-one quasi-isometry *(or* C-1qi-reduction*). We write $\alpha <_{1qi} \beta$ to mean that $\alpha \leq_{1qi} \beta$ and $\beta \not\leq_{1qi} \alpha$.*

▶ **Definition 11** (Permutation Reducibility). *A string α is* permutation reducible*, or* pqi-reducible*, to a string β iff there exists a one-one quasi-isometry f from α to β such that f is surjective. We call such an f a* permutation quasi-isometry (or pqi-reduction)*, and write $\alpha \leq_{pqi} \beta$ to mean that α is permutation reducible to β; if, in addition, f is a C-quasi-isometry, then we call f a* C-permutation quasi-isometry *(or* C-pqi-reduction*). We write $\alpha <_{pqi} \beta$ to mean that $\alpha \leq_{pqi} \beta$ and $\beta \not\leq_{pqi} \alpha$. Here, note that it can be shown that $\alpha \leq_{pqi} \beta$ implies $\beta \leq_{pqi} \alpha$.*

Given an alphabet Σ, the relations $\leq_{mqi}$, $\leq_{1qi}$, $\leq_{pqi}$ and $\leq_{qi}$ are preorders on the class of infinite strings over Σ. Let $\equiv_{mqi}$ be the relation on Σ^ω such that $\alpha \equiv_{mqi} \beta$ iff $\alpha \leq_{mqi} \beta$ and $\beta \leq_{mqi} \alpha$. Then $\equiv_{mqi}$ is an equivalence relation on Σ^ω. We call an equivalence class on Σ^ω induced by $\equiv_{mqi}$ a *many-one quasi-isometry degree* (or *mqi-degree*), and denote the mqi-degree of an infinite string α by $[\alpha]_{mqi}$. Analogous definitions apply to $\equiv_{1qi}$, $[\alpha]_{1qi}$, $\equiv_{pqi}$, $[\alpha]_{pqi}$, $\equiv_{qi}$ and $[\alpha]_{qi}$.

We denote the partial orders induced by $\leq_{pqi}, \leq_{1qi}, \leq_{mqi}$ and $\leq_{qi}$ on the pqi-degrees, 1qi-degrees, mqi-degrees and qi-degrees by $\Sigma^\omega_{pqi}, \Sigma^\omega_{1qi}, \Sigma^\omega_{mqi}$ and Σ^ω_{qi} respectively.

By definition, Σ^ω_{pqi} is a refinement of Σ^ω_{1qi} in the sense that for all infinite strings α and β, $[\alpha]_{pqi} \leq_{pqi} [\beta]_{pqi} \Rightarrow [\alpha]_{1qi} \leq_{1qi} [\beta]_{1qi}$. In a similar manner, Σ^ω_{1qi} is a refinement of Σ^ω_{mqi}, which is in turn a refinement of Σ^ω_{qi}. The first subsection deals with the mqi-degrees, starting with the inner structure of each mqi-degree.

4.1 Structure of the mqi-Degrees

Fix any two distinct infinite strings β and γ belonging to $[\alpha]_{mqi}$. It can be shown that β and γ have a common upper bound as well as a common lower bound in $[\alpha]_{mqi}$ such that these bounds are witnessed by 1qi-reductions.

▶ **Proposition 12.** *For any two distinct infinite strings $\beta, \gamma \in [\alpha]_{mqi}$, there exists a $\delta \in [\alpha]_{mqi}$ such that $\beta \leq_{1qi} \delta$ and $\gamma \leq_{1qi} \delta$.*

Proof. Let f be a C-mqi-reduction from β to γ. Let δ be the infinite string obtained from γ by repeating $C+1$ times each letter of γ. Then $\gamma \leq_{1qi} \delta$ via a $(C+1)$-1qi-reduction g defined by $g(n) = (n-1)\cdot(C+1)+1$ for each $n \in \mathbb{N}$. Furthermore, $\delta \leq_{mqi} \gamma$ via a C-mqi-reduction g' defined by $g'(n) = \lceil \frac{n}{C+1} \rceil$. Thus $\delta \in [\alpha]_{mqi}$.

Next, one constructs a (C^2+2C)-1qi-reduction f' from β to δ using the function f. For each y in the range of f, map the pre-image of y under f, which by Lemma 5 has at most $C+1$ elements, to the set of positions of δ corresponding to the $C+1$ copies of the letter at position y. Formally, define

$$f'(n) = \begin{cases} g(f(n)), & \text{if } f(n) \neq f(n') \text{ for all } n' < n; \\ g(f(n)) + C', & \text{otherwise; where } 1 \leq C' < C+1 \text{ is minimum such that} \\ & g(f(n)) + C' \neq f'(n') \text{ for all } n' < n. \end{cases}$$

We verify that f' is an injective (C^2+2C)-quasi-isometry. Injectiveness follows from the definition of f': in the first case, the injectiveness of g ensures that $f'(x) \neq f'(x')$ for all $x' < x$; in the second case, it is directly enforced that $f'(x) \neq f(x')$ for all $x' < x$. Since f is a C-reduction, $x + C < y \Rightarrow f(x) < f(y) \Rightarrow g(f(x)) < g(f(y)) \Rightarrow f'(x) < f'(y)$, and so f' satisfies Condition (b) with constant C. Now we show that f' satisfies Condition (a) with constant C^2+2C. By Condition (a), $d(f(x), f(x+1)) \leq C$. Without loss of generality, assume that $f(x) \leq f(x+1)$. By the definition of f', $f'(x) \geq g(f(x))$ and $f'(x+1) \leq g(f(x+1)) + C$. Since $f(x) \leq f(x+1)$, it follows that $f'(x) \leq f'(x+1)$ and so

$$\begin{aligned} d(f'(x+1), f'(x)) &\leq g(f(x+1)) + C - g(f(x)) \\ &= (C+1)\cdot(f(x+1)-1) + 1 + C - (C+1)\cdot(f(x)-1) - 1 \\ &= (C+1)\cdot(f(x+1) - f(x)) + C \\ &\leq C\cdot(C+1) + C \\ &= C^2 + 2C\,. \end{aligned}$$

This completes the proof. ◀

Next, we prove a lower bound counterpart of Proposition 12.

▶ **Proposition 13.** *For any two distinct infinite strings $\beta, \gamma \in [\alpha]_{mqi}$, there exists a $\delta \in [\alpha]_{mqi}$ such that $\delta \leq_{1qi} \beta$ and $\delta \leq_{1qi} \gamma$.*

Proof. Suppose $\beta = \beta_1\beta_2\ldots$, where $\beta_i \in \Sigma$. Let $f : \mathbb{N} \to \mathbb{N}$ be a C-mqi-reduction from β to γ. Now define $\delta = \beta_{i_1}\beta_{i_2}\ldots$, where i_k is the minimum index such that $i_k \neq i_l$ for all $l < k$ and for all $j < i_k$, $f(j) \neq f(i_k)$. By Condition (b), the range of f is infinite and thus each i_k is well-defined. We verify that $\delta \leq_{1qi} \beta$ and $\delta \leq_{1qi} \gamma$.

Define $f'(n) = i_n$ for all $n \in \mathbb{N}$. We show that f' is a 1qi-reduction from δ to β. By the choice of the i_n's, $f'(n) > f'(m)$ whenever $n > m$; in particular, f' is injective and Condition (b) holds for f'. Furthermore, given any n, by applying Condition (b) to f and all $n' \leq n$, it follows that $f'(n+1) \leq f'(n) + C + 1$. Hence f' also satisfies Condition (a).

Next, define a 1qi-reduction f'' from δ to γ by $f''(n) = f(i_n)$. The injectiveness of f'' follows from the choice of the i_n's (though f'' is not necessarily strictly monotone increasing). Using the fact that $i_{n+1} \leq i_n + C + 1$, as well as applying Condition (a) $i_{n+1} - i_n$ times, $d(f''(n+1), f''(n)) = d(f(i_{n+1}), f(i_n)) \leq C \cdot d(i_{n+1}, i_n) \leq C \cdot (C+1)$. Hence f'' satisfies Condition (a) with constant $C \cdot (C+1)$. Since the i_n's are strictly increasing, $m + C < n \Rightarrow i_m + C < i_n \Rightarrow f(i_m) < f(i_n)$. Thus f'' is a $C \cdot (C+1)$-1qi-reduction.

Lastly, define a mqi-reduction g from β to δ by $g(n) = k$ where k is the minimum integer with $f(n) = f(i_k)$. As the i_n's cover the whole range of f, g is well-defined. For any given n, suppose $g(n) = k_1$ and $g(n+1) = k_2$, so that $f(n) = f(i_{k_1})$ and $f(n+1) = f(i_{k_2})$. By Condition (b), $d(n, i_{k_1}) \leq C$ and $d(n+1, i_{k_2}) \leq C$, and so

$$\begin{aligned} d(g(n), g(n+1)) &= d(k_1, k_2) \\ &\leq d(i_{k_1}, i_{k_2}) \\ &\leq d(n, i_{k_1}) + d(n, n+1) + d(n+1, i_{k_2}) \\ &\leq 2C + 1\,. \end{aligned}$$

Hence g satisfies Condition (a) with constant $2C+1$. To verify that g satisfies Condition (b) for some constant, fix any n and apply Condition (b) $C \cdot (C+1)$ times to f, giving $f(n) + C \cdot (C+1) \leq f(n + C \cdot (C+1)^2)$. Suppose $g(n) = i_{k_1}$ and $g(n + C \cdot (C+1)^2) = i_{k_2}$, so that $f(n) = f(i_{k_1})$ and $f(n + C \cdot (C+1)^2) = f(i_{k_2})$. Then $d(f(i_{k_1}), f(i_{k_2})) = d(f(n), f(n+

$C \cdot (C+1)^2)) \geq C \cdot (C+1)$. So by applying Condition (a) $d(i_{k_1}, i_{k_2})$ times to f, we have $C \cdot d(i_{k_1}, i_{k_2}) \geq d(f(i_{k_1}), f(i_{k_2})) \geq C \cdot (C+1)$. Dividing both sides of the inequality by C yields $d(i_{k_1}, i_{k_2}) \geq C+1$. Applying the contrapositive of Condition (b) to f then gives $f(i_{k_2}) \geq f(i_{k_1}) \Rightarrow i_{k_2} + C \geq i_{k_1}$. Since $d(i_{k_1}, i_{k_2}) \geq C+1$, this implies that $g(n + C \cdot (C+1)^2) = i_{k_2} > i_{k_1} = g(n)$. Thus g satisfies Condition (b) with constant $C \cdot (C+1)^2 - 1$. ◀

4.2 1qi-Degrees Within mqi-Degrees

We now investigate the structural properties of 1qi-degrees within individual mqi-degrees. As will be seen shortly, these properties can vary quite a bit depending on the choice of the mqi-degree.

▶ **Proposition 14.** *There exists an infinite string α such that $[\alpha]_{mqi}$ is the union of an infinite ascending chain of 1qi-degrees.*

Proof. Let $\Sigma = \{0, 1\}$ and let $\alpha = 10^\omega$. Then $[\alpha]_{mqi}$ consists of all infinite strings with a finite, positive number of occurrences of 1. Given any infinite string β with $k \geq 1$ occurrences of 1, β is 1qi-equivalent to a string γ in $[\alpha]_{mqi}$ iff γ has exactly k occurrences of 1. If $1 \leq k < k'$, then each string $\beta \in [\alpha]_{mqi}$ with exactly k occurrences of 1 is 1qi-reducible to any string $\beta' \in [\alpha]_{mqi}$ with exactly k' occurrences of 1. Thus $[\alpha]_{mqi}$ is the union of an ascending chain $[\alpha]_{1qi} < [110^\omega]_{1qi} < [1110^\omega]_{1qi} < \ldots$, where the i-th term of this chain is $1^i 0^\omega$. ◀

▶ **Proposition 15.** *There exists an infinite string α such that the poset of 1qi-degrees within $[\alpha]_{mqi}$ is isomorphic to $\mathbb{N}^2$ with the componentwise ordering. That is, $[\alpha]_{mqi}$ is the union of infinitely many disjoint infinite ascending chains of 1qi-degrees such that every pair of these ascending chains has incomparable elements. Also, $[\alpha]_{mqi}$ does not contain infinite anti-chains of 1qi-degrees.*

Proof. Let $\Sigma = \{0, 1, 2\}$ and let $\alpha = 120^\omega$. Then $[\alpha]_{mqi}$ consists of all infinite strings with a finite, positive number of 1's and a finite, positive number of 2's. Furthermore, $[\alpha]_{1qi}$ consists of all infinite strings with exactly one occurrence of 1 and exactly one occurrence of 2.

Based on the proof of Proposition 14, $[\alpha]_{mqi}$ is the union, over all $k \geq 1$, of chains of the form $[12^k 0^\omega]_{1qi} < [1^2 2^k 0^\omega]_{1qi} < \ldots$, where the i-th term of each chain is $[1^i 2^k 0^\omega]_{1qi}$. Given any two chains $\Gamma_j = \{[1^i 2^j 0^\omega]_{1qi} : i \in \mathbb{N}\}$ and $\Gamma_k = \{[1^i 2^k 0^\omega]_{1qi} : i \in \mathbb{N}\}$, where $j < k$, the classes $[1^2 2^j 0^\omega]_{1qi} \in \Gamma_j$ and $[12^k 0^\omega]_{1qi} \in \Gamma_k$ are incomparable with respect to $\leq_{1qi}$.

It remains to show that any anti-chain of 1qi-degrees contained in $[\alpha]_{mqi}$ must be finite. Consider any anti-chain of 1qi-degrees containing the class $[1^i 2^j 0^\omega]_{1qi} \subsetneq [\alpha]_{mqi}$. Every element of this anti-chain that is different from $[1^i 2^j 0^\omega]_{1qi}$ is of the form $[1^{i'} 2^{j'} 0^\omega]_{1qi}$, where either $i < i'$ and $j > j'$, or $i > i'$ and $j < j'$. Thus, if the anti-chain were infinite, then it would contain at least 2 1qi-degrees, $[\beta]_{1qi}$ and $[\gamma]_{1qi}$, such that either β has the same number of occurrences of 1 as γ, or β has the same number of occurrences of 2 as γ. This is a contradiction as it would imply that either $\beta \leq_{1qi} \gamma$ or $\gamma \leq_{1qi} \beta$. ◀

4.3 pqi-Reductions

We now discuss pqi-reductions, which are the most stringent kind of quasi-isometric reductions considered in the present work. Pqi-reductions are 1qi-reductions that are surjective; an example of such a reduction is the mapping $2m-1 \mapsto 2m$, $2m \mapsto 2m-1$ from $(01)^\omega$ to $(10)^\omega$. We record a few elementary properties of pqi-reductions.

▶ **Lemma 16.** *If f is a pqi-reduction and if $x + D = f(x)$ for some $D \geq 1$ and some $x \in \mathbb{N}$, then there are at least D positions $y > x$ such that $f(y) < f(x)$.*

Proof. If $x + D = f(x)$ for some $D \geq 1$, then $\{1, \ldots, x + D - 1\} \setminus \{f(1), \ldots, f(x-1)\}$ must contain at least D elements as the former set contains D more elements than the latter. Thus, for f to be a bijection, there must exist at least D positions $y > x$ that are mapped by f into $\{1, \ldots, x + D - 1\} \setminus \{f(1), \ldots, f(x-1)\}$. ◀

We next observe that for any pqi-reduction f, there is a uniform upper bound on the difference $x - f(x)$.

▶ **Proposition 17.** *If f is a C-pqi-reduction, then for all $x \in \mathbb{N}$, $x - f(x) < 2C^2 + 1$.*

Proof. Assume, by way of contradiction, that there is some $x \in \mathbb{N}$ such that $x - f(x) \geq 2C^2 + 1$. First, suppose that there are at least $C^2 + 1$ numbers z such that $z > x$ and $f(z) \in \{f(x)+1, f(x)+2, \ldots, x-1\}$. Then there are at least $C^2 + 1$ numbers z' such that $z' < x$ and $f(z') > x > f(x)$, among which there is at least one z'_0 with $f(z'_0) \geq x + C^2 + 1$. This would contradict the fact that by the Small Cross-Over Lemma (Lemma 8), $z'_0 < x \Rightarrow f(z'_0) \leq f(x) + C^2 < x + C^2$.

Second, suppose that f maps at most C^2 numbers greater than x into $\{f(x)+1, f(x)+2, \ldots, x-1\}$. Then there are at least $C^2 + 1$ numbers less than x that are mapped into $\{f(x)+1, f(x)+2, \ldots, x-1\}$ and in particular, there is at least one number $y < x$ such that $f(y) \geq f(x) + C^2 + 1$, contradicting the Small Cross-over Lemma. Thus for all $x \in \mathbb{N}$, $x - f(x) < 2C^2 + 1$. ◀

Lemma 16 and Proposition 17 together give a uniform upper bound on the absolute difference between any position number and its image under a C-pqi-reduction.

▶ **Corollary 18.** *If f is a C-pqi-reduction, then for all $x \in \mathbb{N}$, $|x - f(x)| < 2C^2 + 1$.*

Proof. By Condition (b), there cannot be more than C numbers y such that $y > x$ and $f(y) < f(x)$. Lemma 16 thus implies that there cannot exist any $D > C$ such that $x + D = f(x)$, and so $f(x) - x \leq C$. Combining the latter inequality with that in Proposition 17 yields $|x - f(x)| < \max\{C+1, 2C^2+1\} = 2C^2 + 1$. ◀

Given any infinite string α, it was observed earlier that by the definitions of pqi, 1qi and mqi-reductions, $[\alpha]_{pqi} \subseteq [\alpha]_{1qi} \subseteq [\alpha]_{mqi}$. In the following example, we give instances of strings α where each of the two subset relations is proper or can be replaced with the equals relation.

▶ **Example 19.**

(a) $[\alpha]_{pqi} = [\alpha]_{1qi} = [\alpha]_{mqi}$. Set $\alpha = 0^\omega$. For any infinite string γ such that $\gamma \leq_{mqi} 0^\omega$, γ can only contain occurrences of 0, and therefore $[0^\omega]_{pqi} = [0^\omega]_{1qi} = [0^\omega]_{mqi} = \{0^\omega\}$.

(b) $[\alpha]_{1qi} = [\alpha]_{mqi}$ and $[\alpha]_{pqi} \subset [\alpha]_{1qi}$. Set $\alpha = (01)^\omega$. First, $(001)^\omega \leq_{1qi} (01)^\omega$, as witnessed by the 1qi-reduction $3n-2 \mapsto 4n-3, 3n-1 \mapsto 4n-1, 3n \mapsto 4n$ for $n \in \mathbb{N}$. We also have $(01)^\omega \leq_{1qi} (001)^\omega$ via the 1qi-reduction $2n-1 \mapsto 3n-2, 2n \mapsto 3n$ for $n \in \mathbb{N}$. However, $(001)^\omega \notin [(01)^\omega]_{pqi}$ because the density of 0's and 1's in the two strings are different, making it impossible to construct a permutation reduction between them. More formally, if there were a pqi-reduction from $(001)^\omega$ to $(01)^\omega$, then by Corollary 18, there would be a constant D such that for each n, the first $3n$ positions of $(001)^\omega$ are mapped into the first $3n + D$ positions of $(01)^\omega$. But the first $3n$ positions of $(001)^\omega$ contain $2n$ occurrences of 0 while the first $3n + D$ positions of $(01)^\omega$ contain at most $\lceil 1.5n + \frac{D}{2} \rceil$ occurrences of 0, and for large enough n, one has $2n > \lceil 1.5n + \frac{D}{2} \rceil$. Hence no pqi-reduction from $(001)^\omega$ to $(01)^\omega$ can exist, and so $[\alpha]_{pqi} \subset [\alpha]_{1qi}$.

To see that $[(01)^\omega]_{mqi} \subseteq [(01)^\omega]_{1qi}$, we first note that any string that is mqi-reducible to $(01)^\omega$ (or to any other recursive string) must be recursive. Thus if $\beta \leq_{mqi} (01)^\omega$, then a 1qi-reduction from β to $(01)^\omega$ can be constructed by mapping the n-th position of β to the position of the matching letter in the n-th occurrence of 01 in $(01)^\omega$. Next, suppose that f is a C-mqi-reduction from $(01)^\omega$ to β. By Corollary 7, f maps the positions of $(01)^\omega$ to a sequence of positions of β that contains 0 and 1 every $2C$ positions. Thus a 1qi-reduction can be constructed from $(01)^\omega$ to β by mapping, for each n, the $(2n-1)$-st and $(2n)$-th positions of $(01)^\omega$ to the positions of the first occurrence of 0 and first occurrence of 1 respectively in the interval $[2C(n-1)+1, 2Cn]$ of positions of β. Therefore $\beta \in [(01)^\omega]_{1qi}$.

(c) $[\alpha]_{1qi} \subset [\alpha]_{mqi}$ and $[\alpha]_{pqi} = [\alpha]_{1qi}$. Set $\alpha = 10^\omega$. We recall from the proof of Proposition 14 that $[10^\omega]_{pqi}$ and $[10^\omega]_{1qi}$ consist of all binary strings with a single occurrence of 1, while $[10^\omega]_{mqi}$ consists of all binary strings with a finite, positive number of occurrences of 1. Thus $[10^\omega]_{pqi} = [10^\omega]_{1qi}$ and $[10^\omega]_{pqi} \neq [10^\omega]_{mqi}$.

(d) $[\alpha]_{pqi} \subset [\alpha]_{1qi} \subset [\alpha]_{mqi}$. Set $\alpha = (0^n1)_{n=1}^\infty$, the concatenation of all strings 0^n1 where $n \in \mathbb{N}$. Then $\beta = (0^n11)_{n=1}^\infty \in [\alpha]_{mqi}$; however, $\beta \notin [\alpha]_{1qi}$ as each pair of adjacent positions of 1's in β must be mapped to distinct positions of 1's in α, but the distance between the n-th and $(n+1)$-st occurrences of 1 in α increases linearly with n, meaning that Condition (a) cannot be satisfied.

To construct an mqi-reduction from β to α, map the positions of the substring 0^n11 of β to the positions of the substring 0^n1 of α as follows: for $k \in \{1, \ldots, n\}$, the position of the k-th occurrence of 0 in 0^n11 is mapped to that of the k-th occurrence of 0 in 0^n1, while the two positions of 1's in 0^n11 are mapped to the position of the single 1 in 0^n1. For an mqi-reduction from α to β, for each substring 0^n1 of α and each substring 0^n11 of β, the positions of 0^n in 0^n1 are mapped to the corresponding positions of 0^n in 0^n11, while the position of 1 in 0^n1 is mapped to the position of the first occurrence of 1 in 0^n11. Thus $\beta \in [\alpha]_{mqi}$.

Furthermore, $\gamma = 1(0^n1)_{n=1}^\infty \in [\alpha]_{1qi}$ but $\gamma \notin [\alpha]_{pqi}$. The reason for γ not being pqi-reducible to α is similar to that given in Example (b). If such a pqi-reduction did exist, then by Corollary 18, there would exist a constant D such that for all n, the first $1 + \sum_{k=1}^n (k+1) = 1 + \frac{n(n+3)}{2}$ positions of γ are mapped into the first $1 + \frac{n(n+3)}{2} + D$ positions of α. But the first $1 + \frac{n(n+3)}{2}$ positions of γ contain $n+1$ occurrences of 1 and for large enough n, the first $1 + \frac{n(n+3)}{2} + D$ positions of α contain at most n occurrences of 1. Hence no pqi-reduction from γ to α is possible.

For a 1qi-reduction from γ to α, map the starting position of γ, where the letter 1 occurs, to the first occurrence of 1 in α. For subsequent positions of γ, for each $n \geq 1$, the set of positions of γ where the substring 0^n1 occurs can be mapped in a one-to-one fashion into the set of positions of α where the substring $0^{n+1}1$ occurs. To see that α is 1qi-reducible to γ, it suffices to observe that α is a suffix of γ, so one can map the positions of α in a one-to-one fashion to the positions of the suffix of γ corresponding to α. The 1qi-reduction from α to γ is trivial.

Proposition 21 extends the first example in Example 19 by characterising all recursive strings whose pqi, 1qi and mqi-degrees all coincide. In fact, there are only $|\Sigma|$ many such strings: those of the form a_i^ω, where $a_i \in \Sigma$. We call the pqi, 1qi and mqi-degrees of such strings *trivial*. Due to space constraint, the proof of Proposition 21 is omitted.

▶ **Definition 20.** *The pqi, 1qi and mqi-degrees of each string a_i^ω, where $a_i \in \Sigma$, will be called* trivial *pqi, 1qi and mqi-degrees respectively.*

▶ **Proposition 21.** *If, for some recursive string α, $[\alpha]_{pqi} = [\alpha]_{1qi} = [\alpha]_{mqi}$, then all three degree classes are trivial.*

We observe next that every non-trivial pqi degree must be infinite.

▶ **Proposition 22.** *All non-trivial pqi-degrees are infinite.*

Proof. Suppose that at least two distinct letters occur in α. Fix a letter, say a_1, that occurs infinitely often in α. Let a_2 be a letter different from a_1 that occurs in α. For each $n \in \mathbb{N}$, let $\beta_n = a_1^n a_2 \alpha^{(n+1)}$, where $\alpha^{(n+1)}$ is obtained from α by removing the first occurrence of a_2 as well as the first n occurrences of a_1. Since β_n is built from α by permuting the letters occurring at a finite set of positions of α, $\beta_n \in [\alpha]_{pqi}$. As the β_n's are all distinct, it follows that $[\alpha]_{pqi}$ is indeed infinite. ◀

We close this subsection by illustrating an application of Proposition 22, showing that if the mqi-degree of α contains at least two distinct strings such that one is 1qi-reducible to the other, then the first string is 1qi-reducible to infinitely many strings in $[\alpha]_{mqi}$.

▶ **Proposition 23.** *If there exist distinct $\beta \in [\alpha]_{mqi}$ and $\gamma \in [\alpha]_{mqi}$ such that $\beta \leq_{1qi} \gamma$, then β is 1qi-reducible to infinitely many strings in $[\alpha]_{mqi}$.*

Proof. Suppose that $\beta \leq_{1qi} \gamma$ and $\beta \neq \gamma$ for some $\beta \in [\alpha]_{mqi}$ and $\gamma \in [\alpha]_{mqi}$. Then $[\alpha]_{mqi}$ is non-trivial, so by Proposition 22, $[\gamma]_{pqi}$ is infinite. Since $[\gamma]_{pqi} \subseteq [\gamma]_{1qi}$, $[\gamma]_{1qi}$ is also infinite. Thus β is 1qi-reducible to each of the infinitely many strings in $[\gamma]_{1qi}$. ◀

4.4 The Partial Order of All mqi-Degrees

As discussed earlier, Khoussainov and Takisaka [6] observed that for any alphabet $\Sigma = \{a_1, \ldots, a_l\}$, the partial order Σ^ω_{qi} has a greatest element equal to $[(a_1 \cdots a_l)^\omega]_{qi}$. Their proof also extends to the partial order of all recursive mqi-degrees, showing that for each recursive string α, $[\alpha]_{mqi} \leq_{mqi} [(a_1 \cdots a_l)^\omega]_{mqi}$. We next prove that there is a pair of recursive mqi-degrees whose join is precisely the maximum recursive mqi-degree $[(a_1 \cdots a_l)^\omega]_{mqi}$.

▶ **Proposition 24.** *Suppose that $\Sigma = \{a_1, \ldots, a_l\}$. Then there exist two distinct infinite strings α and β such that $[(a_1 \cdots a_l)^\omega]_{mqi}$ is the unique recursive common upper bound of $[\alpha]_{mqi}$ and of $[\beta]_{mqi}$ under $\leq_{mqi}$.*

Proof. Let $\alpha = (a_1)^\omega$ and $\beta = (a_2 a_3 \cdots a_l)^\omega$. Suppose that for some recursive string γ, $\alpha \leq_{mqi} \gamma$ via a C-mqi-reduction. Since a_1 is the only letter occurring in α, Condition (a) implies that there must be at least one occurrence of a_1 in γ every C positions. Similarly, if $\beta \leq_{mqi} \gamma$ via a C'-mqi-reduction, then for each a_i with $i \geq 2$, since a_i occurs every $l-1$ positions, it must also occur in γ every $C' \cdot (l-1)$ positions. Hence there exists a constant C'' such that every substring of γ of length C'' contains at least one occurrence of a_i for every $i \in \{1, \ldots, l\}$, and therefore $(a_1 \cdots a_l)^\omega \leq_{mqi} \gamma$. Since $\gamma \leq_{mqi} (a_1 \cdots a_l)^\omega$ follows from the proof of [6, Proposition II.1], one has $\gamma \in [(a_1 \cdots a_l)^\omega]_{mqi}$, as required. ◀

Khoussainov and Takisaka [6] showed that the partial order Σ^ω_{qi} is not dense. In particular, given any distinct $a_i, a_j \in \Sigma$, there is no element $[\beta]_{qi}$ that is strictly between the minimal element $[(a_j)^\omega]_{qi}$ and the "atom" $[a_i(a_j)^\omega]_{qi}$ [6, Proposition II.1]. The next theorem shows similarly that the partial order Σ^ω_{mqi} is non-dense with respect to pairs of mqi-degrees. The high-level proof of the theorem is given below; the proofs of claims used are not shown due to space limitations.

▶ **Theorem 25.** *There exist two pairs (α, β) and (γ, δ) of recursive strings such that both α and β are mqi-reducible to γ as well as mqi-reducible to δ, but there is no string ξ such that $\alpha \leq_{mqi} \xi, \beta \leq_{mqi} \xi, \xi \leq_{mqi} \gamma$ and $\xi \leq_{mqi} \delta$.*

Proof. Let $\Sigma = \{0, 1\}$. Define the strings

$$\alpha = \sigma_1\sigma_2 \ldots, \quad \text{where } \sigma_n = (01)^{2^{2^n}} 0^n 1^n\,;$$

$$\beta = \tau_1\tau_2 \ldots, \quad \text{where } \tau_n = (01)^{2^{2^n}} 1^n 0^n\,;$$

$$\gamma = \mu_1\mu_2 \ldots, \quad \text{where } \mu_n = (01)^{2^{2^n}} 0^n\,;$$

$$\delta = \nu_1\nu_2 \ldots, \quad \text{where } \nu_n = (01)^{2^{2^n}} 1^n\,.$$

We first show that $\alpha \leq_{mqi} \gamma$. For each $i \in \mathbb{N}$, define the following intervals of positions.

- $K_i = [k_i, k_i + 2^{2^i+1} - 1]$ is the interval of positions of the substring $(01)^{2^{2^i}}$ of σ_i in α.
- $R_i = [r_i, r_i + 2i - 1]$ is the interval of positions of the substring 0^i1^i of σ_i in α.
- $L_i = [l_i, l_i + 2^{2^i+1} - 1]$ is the interval of positions of the substring $(01)^{2^{2^i}}$ of μ_i in γ.
- $L'_i = [l_i + 2^{2^i+1}, l_i + 2^{2^i+1} + i - 1]$ is the interval of positions of the substring 0^i of μ_i in γ.

Define an mqi-reduction g from α to γ as follows. For $i \in \mathbb{N}$,

$$g(k_i + 4w + 2u + x) = l_i + 2(i-1) + 2w + x\,, \quad 0 \leq w \leq i-2, u, x \in \{0, 1\}\,;$$

$$g(k_i + m) = l_i + m\,, \quad 4i - 4 \leq m \leq 2^{2^i+1} - 1\,;$$

$$g(r_i + m) = l_i + 2^{2^i+1} + m\,, \quad 0 \leq m \leq i - 1\,;$$

$$g(r_i + i + m) = l_{i+1} + 2m + 1\,, \quad 0 \leq m \leq i - 1.$$

The mqi-reduction g maps the interval K_i to the suffix of the interval L_i starting at its $(2i-1)$-st position such that each of the first $i-1$ pairs of positions of this suffix is the image of two consecutive pairs of positions of K_i, while the remaining $|K_i| - 2(i-1)$ positions of K_i is mapped by g to the remaining positions of the suffix of L_i in a one-to-one fashion. Thus g is a 4-mqi-reduction from α to γ. A similar mqi-reduction can be constructed from β to γ, from α to δ, as well from β to δ.

Assume, by way of contradiction, that there is a string ξ and there are mqi-reductions f_1 from α to ξ, f_2 from β to ξ, f_3 from ξ to γ and f_4 from ξ to δ with constants C_1, C_2, C_3 and C_4 respectively. Set $C = \max\{C_1, C_2, C_3, C_4\}$ and fix some $n > 2C^7 + 1$.

For $i \in \mathbb{N}$, let $K'_i = [k_i + C^2 + 2, k_i + 2^{2^i+1} - 3 - C^2]$ be the interval obtained from K_i by removing the first and last $C^2 + 2$ positions. We make the following observation. The proof is omitted due to space constraint.

▷ Claim 26. For all positions $m \in K'_n$, for $i \in \{1, 2\}$ and $j \in \{3, 4\}$, $f_j(f_i(m)) \in L_n$.

Define the sets $H_i = f_1(K'_i) \cup f_2(K'_i)$ for $i \in \mathbb{N}$. We show that the sets H_n and H_{n+1} are non-overlapping by proving $\max(H_n) < \min(H_{n+1})$. By Claim 26, for all $m \in H_n$ and $j \in \{3, 4\}$ we have $f_j(m) \in L_n$. Then, we have $f(\min(H_{n+1})) - f(\max(H_n)) \geq \min(L_{n+1}) - \max(L_n) = n + 1 > 2C^7 + 2 > C^2$. So by the Small Cross-Over Lemma, $\min(H_{n+1}) > \max(H_n)$.

Consider the interval $[\min(H_n), \max(H_n)]$ in the domain of ξ. By Claim 26, f_3 (resp. f_4) maps each element of H_n into L_n. Fix any other position z in the interval. Then f_3 cannot map z into L'_n, which is the set of positions in γ of the string 0^n. To see this, we note that if ℓ and $\ell + 1$ are the two largest values of K_n, then ℓ is at least $C^2 + 1$ more than the value x such that $f_i(x) = \max(H_n)$ for some $i \in \{1, 2\}$, and so by Condition (b), $z + C < \max(H_n) + C < f_k(\ell + 1)$ for $k \in \{1, 2\}$. Thus $f_3(z) < f_3(f_k(\ell + 1))$ for

$k \in \{1,2\}$. Furthermore, by applying Condition (a) repeatedly to f_3 and then to f_k, we have $d(f_3(f_k(\ell+1)), f_3(f_k(\max(K'_n)))) \leq C \cdot d(f_k(\ell+1), f_k(\max(K'_n))) \leq C^2 \cdot (C^2+2) = C^4+2C^2$. Since $f_3(f_k(\max(K'_n))) \in L_n$ and we fixed $n > 2C^7+1$, then $f_3(f_k(\ell+1)) \notin L_{n+1}$. Furthermore, the letter at position $f_3(f_k(\ell+1))$ of γ is 1. Thus $f_3(z)$ cannot lie in L'_n as there is no occurrence of 1 in L'_n. A similar argument, using position ℓ rather than position $\ell+1$, shows that $f_4(z)$ cannot lie in L'_n. One can also prove similarly that none of the positions in the interval $[\min(H_{n+1}), \max(H_{n+1})]$ is mapped by f_3 or f_4 into the interval L'_n.

Next, we consider the positions of ξ between $\max(H_n)$ and $\min(H_{n+1})$. Since none of the positions of ξ in the union $[\min(H_n), \max(H_n)] \cup [\min(H_{n+1}), \max(H_{n+1})]$ is mapped by f_3 into L'_n and L'_n is an interval of length $n > 2C^7$, Lemma 6 implies that there are at least $\lfloor \frac{n}{C_3} \rfloor$ positions of ξ between H_n and H_{n+1} which are mapped into L'_n. Then, we can make the following observations – the proofs of which are omitted due to space constraint.

▷ Claim 27. The string ξ contains a substring of 0's (resp. 1's) of length $\Omega(C^4)$ between H_n and H_{n+1} such that all positions of this substring are mapped by f_3 (resp. f_4) into L'_n.

▷ Claim 28. There cannot exist between H_n and H_{n+1} two $\Omega(C^4)$-long substrings of 0's (resp. 1's) such that an $\Omega(C^4)$-long substring of 1's (resp. 0's) lies between them.

Based on these two claims, there are exactly two maximal intervals J_1 and J_2, each of length $\Omega(C^4)$, such that the substrings of ξ occupied by J_1 and J_2 belong to $\{0\}^*$ and $\{1\}^*$ respectively. Then f_1 maps $\Omega(C^3)$ positions of $[r_n, r_n+n-1]$ into J_1 and $\Omega(C^3)$ positions of $[r_n+n, r_n+2n-1]$ into J_2; further, there are two positions that are $\Omega(C^3)$ positions apart, one in $[r_n, r_n+n-1]$ and the other in $[r_n+n, r_n+2n-1]$, such that f_1 maps the first position into J_1 and the second position into J_2. This implies that J_1 must precede J_2, for otherwise Condition (b) would be violated. Arguing similarly with f_2 in place of f_1 (that is, the mapping from β to ξ), it follows that J_2 must precede J_1, a contradiction. We conclude that the string ξ cannot exist. ◀

Example 19 established separations between various notions of recursive quasi-reducibility: pqi, 1qi and mqi-reducibilities. It remains to separate general quasi-isometry from its recursive counterpart. Due to space constraint, we only give a proof sketch of Theorem 29.

▶ **Theorem 29.** *There exist two recursive strings α and β such that $\alpha \leq_{qi} \beta$ but $\alpha \not\leq_{mqi} \beta$.*

Proof sketch. We begin with an overview of the construction of α and β. To ensure that only non-recursive quasi-isometries between α and β exist, we use a tool from computability theory, which is a Kleene tree [8] – an infinite uniformly recursive binary tree with no infinite recursive branches (see, for example, [11, §V.5]). The idea of the proof is to encode a fixed Kleene tree into β, and construct α such that for any quasi-isometry f from α to β, an infinite branch of the encoded Kleene tree can be computed recursively from f. Hence, f cannot be recursive, as otherwise the chosen infinite branch of the Kleene tree must be recursive, contradicting the definition of a Kleene tree.

We now describe the construction of α and β based on some fixed Kleene tree $T \subseteq \{0,1\}^*$. The building blocks for α and β are called *n-blocks*, which are strings of the following form for some $n \in \mathbb{N}$:

- $\lambda_{(n,0)} = 0^n 1^n$,
- $\lambda_{(n,i)} = 0^{\lfloor \frac{n+1}{2} \rfloor} 1^i 0^{\lceil \frac{n+1}{2} \rceil} 1^n$, for $1 \leq i \leq n-1$,
- $\lambda'_n = (01)^n 1^n$,

We may also call an n-block as simply a *block*.

To construct α and β we concatenate the blocks in stages, where in stage n we arrange n-blocks to form θ_n and ζ_n. Then, θ_n and ζ_n for $n \in \mathbb{N}$ are concatenated to form α and β respectively. That is, $\alpha = \theta_1\theta_2\dots$ and $\beta = \zeta_1\zeta_2\dots$ where θ_n and ζ_n are defined as follows:

- $\theta_1 = \zeta_1 = \lambda_{(1,0)}$.
- For $n \geq 2$, $\theta_n = v_{n,1}\ s_{n,1}\ v_{n,2}\ s_{n,2}\ v_{n,3}\ t_n\ v_{n,4}\ u_n$.
- For $n \geq 2$, $\zeta_n = v'_{n,1}\ s'_{n,1}\ v'_{n,2}\ s'_{n,2}\ v'_{n,3}\ t'_n\ v'_{n,4}\ u'_n$.

We now state the definitions of each variables $v_{n,1}$ and so on, and explain their purpose later. For $n \geq 2$, we define the following:

- Let $B^n_1, B^n_2, B^n_3, B^n_4, B^n_5, B^n_6, B^n_7, B^n_8$ be the number of blocks in α before the start of $v_{n,1},\ s_{n,1},\ v_{n,2},\ s_{n,2},\ v_{n,3},\ t_n,\ v_{n,4},\ u_n$ respectively.
- *Join segments* $v_{n,i} = v'_{n,i} = (\lambda_{(n,0)})^{3nB^n_{2i-1}}$.
- *Scaling segments* $s_{n,1}v_{n,2}s_{n,2}$ and $s'_{n,1}v'_{n,2}s'_{n,2}$, each containing two of the *scaling parts* $s_{n,i} = s'_{n,i} = (\lambda_{(n,1)})^{nB}(\lambda_{(n,2)})^{nB}\dots(\lambda_{(n,n-1)})^{nB}(\lambda_{(n,0)})^{2nB}$ where B means B^n_{2i}.
- *Branching segments* $t_n = (\lambda_{(n,0)})^{2nB^n_6+1}$ and $t'_n = (\lambda_{(n,0)})^{2nB^n_6}\lambda'_n$.
- *Selection segments* u_n and u'_n defined as follows. Let

$$S_n = \left\{ \sum_{m=1}^{n-1} b_m 4^{n-1-m} : b_1 \cdots b_{n-1} \in T \cap \{0,1\}^{n-1} \right\}.$$

  Then, each element $\sum_{m=1}^{n-1} b_m 4^{n-1-m}$ of S_n corresponds to the binary string $b_1 \cdots b_{n-1} \in T$. Define $u_n = \lambda_{(n,1)}(\lambda_{(n,0)})^{\max(S_n)}$. For $1 \leq i \leq \max(S_n)+1$, let the i-th block of u'_n be $\lambda_{(n,0)}$ if $i-1 \notin S_n$ and be $\lambda_{(n,1)}$ if $i-1 \in S_n$.

Note that the number of blocks in the respective segments of α and β are the same. So, for example, the number of blocks in α before $v_{n,i}$ is the same as that of β before $v'_{n,i}$.

Before we explain the purpose of each segment, we first describe some useful properties of the n-blocks. The proofs are omitted due to space restrictions. Let f be any C-quasi-isometric reduction from α to β. For all large enough n:

- No position of the i-th occurrence of an n-block in α is mapped by f to the position of an m-block in β with $m < n$.
- No position of a block $\lambda_{(n,1)}$ is mapped by f to a position of a block $\lambda_{(n,0)}$ occurring in β.
- For $i \leq n-2$, f maps the sequence of positions of each $\lambda_{(n,i)}$ block in a scaling segment of θ_n into either the sequence of positions of a $\lambda_{(n,i)}$ block in a scaling segment of ζ_n or the sequence of positions of a $\lambda_{(n,i+1)}$ block in a scaling segment of ζ_n.
- f can map the sequence of positions of a $\lambda_{(n,n-1)}$ block into the sequence of positions of exactly k blocks $\lambda_{(n,0)}$ iff $k = 2$.
- Up to $C+1$ blocks $\lambda_{(n,0)}$ can be mapped to a single λ'_n block.
- A single $\lambda_{(n,0)}$ block can be mapped across $\lambda_{(n,0)}\lambda'_n$.

We can now describe the purpose of each segment. As above, the proofs are omitted due to space restrictions. Let f be a C-quasi-isometric reduction from α to β. For sufficiently large $n \geq 2$:

- Each join segment $v_{n,i}$ has a non-negative *lead* ℓ such that for all $nB^n_{2i-1}+1 \leq j \leq 2nB^n_{2i-1}$, f maps the j-th $\lambda_{(n,0)}$ block of $v_{n,i}$ to the $(j+\ell)$-th $\lambda_{(n,0)}$ block of $v'_{(n,i)}$.
- The scaling part doubles the lead in the previous join segment. Since a scaling segment contains two scaling parts, the scaling segment multiplies the lead by four. Hence, if the lead of $v_{n,1}$ is ℓ, then the lead of $v_{n,3}$ is 4ℓ.
- The branching segment ensures that the lead is decreased by at most C or increased by 1. So, if the lead of $v_{n,3}$ is 4ℓ, then the lead of $v_{n,4}$ is between $4\ell - C$ and $4\ell + 1$ inclusive.
- The selection segment ensures that the lead in the previous join segment $v_{n,4}$ is in S_n.

Then, one can show that there is some constant c such that for large enough n, the leads ℓ_n and ℓ_{n+1} of the join segments $v_{n,4}$ and $v_{n+1,4}$ are contained in S_n and S_{n+1} respectively, and the binary strings $\sigma_n \in T \cap \{0,1\}^{n-1}$ and $\sigma_{n+1} \in T \cap \{0,1\}^n$, corresponding to ℓ_n and ℓ_{n+1} respectively have a common prefix of length $n-c$. Let τ_n be the prefix of σ_n of length $n-c$. Then, for some large enough n, $\tau_n \prec \tau_{n+1} \prec \tau_{n+2} \prec \ldots$ give an infinite branch of the Kleene tree T, which is non-recursive by definition of Kleene trees. Moreover, this infinite branch can be computed recursively from f. Hence, the quasi-isometric reduction f from α to β must be non-recursive. Hence, $\alpha \not\leq_{mqi} \beta$.

We now describe how to construct a quasi-isometric reduction f from α to β, using some fixed infinite branch $\mathcal{B}(1)\mathcal{B}(2)\cdots$ of the Kleene tree. Since $\theta_1 = \zeta_1 = \lambda_{(n,0)}$, we can map θ_1 to ζ_1 in a strictly increasing manner and the lead in the next segment is 0. We can now describe the mappings for each segment in θ_n for $n \geq 2$. Observe that each block $\lambda_{(n,i)}$ can be mapped to a block $\lambda_{(n,j)}$ in a strictly increasing manner if $j = i$ or $i+1$. For each join segment $v_{(n,i)}$ with lead ℓ_1 and $3nB^n_{2i-1}$ blocks, map the first $3nB^n_{2i-1} - \ell_1$ blocks of $v_{n,i}$ to the last $3nB^n_{2i-1} - \ell_1$ blocks of $v'_{(n,i)}$. Then, map the last ℓ_1 blocks of $v_{n,i}$ to the first ℓ_1 blocks of the following segment in ζ_n. The lead in the next segment is ℓ_1.

Next we describe the mapping for scaling part $s_{n,i}$ with lead ℓ_2. For each $1 \leq j \leq (n-1)nB^n_{2i} - \ell_2$, map the j-th block of $s_{n,i}$ to the $(j+\ell_2)$-th block of $s'_{n,i}$. Map each of the last ℓ_2 blocks $\lambda_{(n,n-1)}$ of $s_{n,i}$ to 2 blocks $\lambda_{(n,0)}$ of $s'_{n,i}$. Map the first $2nB^n_{2i} - 2\ell_2$ blocks $\lambda_{(n,0)}$ of $s_{n,i}$ to the remaining $2nB^n_{2i} - 2\ell_2$ blocks $\lambda_{(n,0)}$ of $s'_{n,i}$. Map the last $2\ell_2$ blocks $\lambda_{(n,0)}$ of $s_{n,i}$ to the first $2\ell_2$ blocks of the following join segment in ζ_n. The lead of the next join segment is $2\ell_2$.

For the branching segment, suppose that the current lead is ℓ_3. If $\mathcal{B}(n-1) = 1$, map the $(2nB^n_6 - \ell_3)$-th $\lambda_{(n,0)}$ block to the concatenation $\lambda_{(n,0)}\lambda'_n$ of two blocks in $t'_n v_{n,4}$. Otherwise, map the $(2nB^n_6 - \ell_3 + 1)$-st $\lambda_{(n,0)}$ block to the λ'_n block in t'_n. Map the rest of the $\lambda_{(n,0)}$ blocks such that f is strictly increasing. Then, the lead of the next join segment is $\ell_3 + \mathcal{B}(n-1)$.

For the selection segment, suppose that the current lead is ℓ_4. Map the $\lambda_{(n,1)}$ block to the (ℓ_4+1)-st block. By induction, $\ell_4 \in S_n$ and so the (ℓ_4+1)-st block in the selection segment of ζ_n is $\lambda_{(n,1)}$. Map the $\lambda_{(n,0)}$ blocks in a strictly increasing manner. ◀

5 Conclusions and Future Investigations

The present paper introduced finer-grained notions of quasi-isometries between infinite strings, in particular requiring the reductions to be recursive. We showed that permutation quasi-isometric reductions are provably more restrictive than one-one quasi-isometric reductions, which are in turn provably more restrictive than many-one quasi-isometric reductions. One result was that general many-one quasi-isometries are strictly more powerful than recursive many-one quasi-isometries, which answers Khoussainov and Takisaka's open problem.

This work also presented some results on the structures of the permutation, one-one and many-one quasi-isometric degrees. It was shown, for example, that there are two infinite strings whose many-one quasi-isometric degrees have a unique common upper bound. It was also proven that the partial order Σ^ω_{mqi} is non-dense with respect to pairs of mqi-degrees. We conclude with the simple observation that the class of mqi-degrees does not form a lattice; in particular, the mqi-degrees $[0^\omega]_{mqi}$ and $[1^\omega]_{mqi}$ do not have a common lower bound.

For future work, we consider an automata-theoretic version of quasi-isometric reduction. Note that Definition 4 defines quasi-isometric reduction based only on the ordering of the natural numbers, as well as adding and subtracting constants. Then we can define an automatic version of Definition 4 for any automatically-ordered set $(A, \leq_A)$ which is order-isomorphic to $(\mathbb{N}, \leq)$: we replace natural number i with the i-th smallest element of A, and study automatic functions $\mu, \nu : A \to \Sigma$ instead of infinite recursive strings $\alpha, \beta : \mathbb{N} \to \Sigma$. (The complete definitions of automatic functions and relations can be found in [3, 4, 5].) Note that the successor function is first-order definable in $(A, \leq_A)$ as follows: $succ(x) = x + 1 \coloneqq \min\{y : x <_A y\}$. So, adding and subtracting constants are automatic as well. Then this definition corresponds to the quasi-isometric reduction between colored metric spaces $(A; d_A, \mu)$ and $(A; d_A, \nu)$ where the metric function d_A is defined in terms of the order-isomorphism g from $(A, \leq_A)$ to $(\mathbb{N}, \leq)$; that is, $d_A(x, y) = |g(x) - g(y)|$.

Now one can ask, for which automatic functions $\mu, \nu : A \to \Sigma$, is there a quasi-isometric reduction from μ to ν? And can every quasi-isometric reduction between some given μ, ν be replaced by an automatic quasi-isometric reduction? The journal version of this paper will also study this automatic setting and show that the answer to the second question depends on $(A, \leq_A)$. Moreover, the expressibility, that is, which mappings from $\mathbb{N}$ to Σ correspond to automatic functions from A to Σ in $(A, \leq_A)$, also depends on the choice of $(A, \leq_A)$.

References

1 Cristian S. Calude. *Information and Randomness – An Algorithmic Perspective*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04978-5.

2 Michael Gromov. Groups of polynomial growth and expanding maps. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 53:53–78, 1981. URL: http://eudml.org/doc/103974.

3 Bernard R. Hodgson. *Théories décidables par automate fini*. PhD thesis, University of Montréal, 1976.

4 Bernard R. Hodgson. Décidabilité par automate fini. *Annales des sciences mathématiques du Québec*, 7(1):39–57, 1983.

5 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Proceedings of the International Workshop on Logic and Computational Complexity (LCC)*, pages 367–392, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. doi:10.1007/3-540-60178-3_93.

6 Bakhadyr Khoussainov and Toru Takisaka. Large scale geometries of infinite strings. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, Reykjavik, Iceland, june 20-23 2017. IEEE Computer Society. doi:10.1109/LICS.2017.8005078.

7 Bakhadyr Khoussainov and Toru Takisaka. Infinite strings and their large scale properties. *The Journal of Symbolic Logic*, 87(2):585–625, 2022. doi:10.1017/JSL.2020.70.

8 Stephen C. Kleene. Recursive functions and intuitionistic mathematics. In *Proceedings of the International Congress of Mathematicians*, volume I, pages 679–685, Cambridge, Massachusetts, U.S.A., 1952. American Mathematical Society.

9 Ryan Lou. *Quasi-isometric Reductions Between K-ary Sequences*. Bachelor thesis, National University of Singapore, 2019.

10 Alexander G. Melnikov. Computably isometric spaces. *Journal of Symbolic Logic*, 78(4):1055–1085, 2013. doi:10.2178/jsl.7804030.

11 Piergiorgio Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. Number 125 in Studies in Logic and the Foundations of Mathematics. Elsevier, North-Holland, Amsterdam, 1989.

12 Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50:284–316, 1944.

13 Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MacGraw-Hill, New York, 1967.

14 Robert I. Soare. *Recursively Enumerable Sets and Degrees – A Study of Computable Functions and Computably Generated Sets*. Perspectives in mathematical logic. Springer, 1987.

Algorithms and Complexity for Path Covers of Temporal DAGs

Dibyayan Chakraborty
School of Computing, University of Leeds, UK

Antoine Dailly
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Florent Foucaud
Université Clermont Auvergne, CNRS, Mines Saint-Étienne, Clermont Auvergne INP, LIMOS, 63000 Clermont-Ferrand, France

Ralf Klasing
Université de Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR 5800, Talence, France

Abstract

A *path cover* of a digraph is a collection of paths collectively containing its vertex set. A path cover with minimum cardinality for a *directed acyclic graph* can be found in polynomial time [Fulkerson, AMS'56; Cáceres et al., SODA'22]. Moreover, Dilworth's celebrated theorem on chain coverings of partially ordered sets equivalently states that the minimum size of a path cover of a DAG is equal to the maximum size of a set of mutually unreachable vertices. In this paper, we examine how far these classic results can be extended to a dynamic setting.

A temporal digraph has an arc set that changes over discrete time-steps; if the underlying digraph is acyclic, then it is a *temporal DAG*. A temporal path is a directed path in the underlying digraph, such that the time-steps of arcs are strictly increasing along the path. Two temporal paths are temporally disjoint if they do not occupy any vertex at the same time. A *temporal path cover* is a collection $\mathcal{C}$ of temporal paths that covers all vertices, and $\mathcal{C}$ is *temporally disjoint* if all its temporal paths are pairwise temporally disjoint. We study the computational complexities of the problems of finding a minimum-size temporal (disjoint) path cover (denoted as Temporal Path Cover and Temporally Disjoint Path Cover).

On the negative side, we show that both Temporal Path Cover and Temporally Disjoint Path Cover are NP-hard even when the underlying DAG is planar, bipartite, subcubic, and there are only two arc-disjoint time-steps. Moreover, Temporally Disjoint Path Cover remains NP-hard even on temporal oriented trees. We also observe that natural temporal analogues of Dilworth's theorem on these classes of temporal DAGs do not hold.

In contrast, we show that Temporal Path Cover is polynomial-time solvable on temporal oriented trees by a reduction to Clique Cover for (static undirected) *weakly chordal* graphs (a subclass of perfect graphs for which Clique Cover admits an efficient algorithm). This highlights an interesting algorithmic difference between the two problems. Although it is NP-hard on temporal oriented trees, Temporally Disjoint Path Cover becomes polynomial-time solvable on temporal oriented *lines* and temporal *rooted directed* trees.

Motivated by the hardness result on trees, we show that, in contrast, Temporal Path Cover admits an XP time algorithm with respect to parameter $t_{\max} + tw$, where $t_{\max}$ is the maximum time-step and tw is the treewidth of the underlying static undirected graph; moreover, Temporally Disjoint Path Cover admits an FPT algorithm with respect to the same parameterization.

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

Keywords and phrases Temporal Graphs, Dilworth's Theorem, DAGs, Path Cover, Temporally Disjoint Paths, Algorithms, Oriented Trees, Treewidth

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.38

Related Version *Full Version*: `https://arxiv.org/abs/2403.04589` [9]

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 38; pp. 38:1–38:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This research was partially supported by the ANR projects GRALMECO (ANR-21-CE48-0004) and TEMPOGRAL (ANR-22-CE48-0001) and by the International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25.

1 Introduction

A classic theorem of Dilworth from 1950 [14] states that in any partially ordered set (poset), the minimum number of chains required to cover all the elements is equal to the maximum size of an antichain. Dilworth's theorem is fundamental from the mathematical point of view; furthermore, an algorithmic proof (that enables to construct a chain cover and an antichain in polynomial time) was published by Fulkerson in 1956 [17]. This theorem and its algorithmic form have many applications, not only in combinatorics, but also in various fields such as bioinformatics [6], scheduling [32], databases [22], program testing [36], etc.

A collection $\mathcal{P}$ of (resp. pairwise vertex-disjoint) directed paths of a digraph D is a *path cover* (resp. *path partition*) of D if all vertices of D are contained in some path of $\mathcal{P}$. Dilworth's theorem can be restated in an equivalent form, equating the minimum cardinality of path covers on directed acyclic graphs (DAGs) and the maximum size of a set of pairwise "unreachable" vertices, or *antichain* vertices [4, 5, 16].

Fulkerson [17] showed that finding a minimum-size path cover of a DAG can be done in polynomial time. Moreover, by using similar methods, one can also find a minimum-size path *partition* in polynomial time for arbitrary DAGs (see [11, Probl. 26-2] or [15, Chapter 11.5]). Improving the best known algorithms for path cover and partitions of DAGs is still an active field of research, see for example [4, 5, 10, 28, 31] for some recent results.

The notions of directed paths and path covers naturally extends to *temporal (di)graphs*. Informally, the arc set of a temporal digraph changes over discrete time-steps and *labels* of an arc are the time-steps where the arc appears. Temporal (di)graphs have been extensively studied in the two last decades, with contributions from and applications to various fields, see [7, 21, 23, 34, 35, 37]. A *temporal path* of a digraph is a path that traverses edges appearing at strictly increasing time-steps. The asymmetric nature of temporal paths has motivated many recent algorithmic works on related reachability or path problems on temporal graphs, such as [1, 2, 3, 8, 24, 33].

Two temporal paths are *temporally disjoint* if they do not occupy a vertex at the same time-step. Motivated by applications in artificial intelligence, this definition was introduced by Klobas et al. [25] and has since then garnered some attention from the graph algorithmic community [29]. Even though the above notion was introduced in the context of temporal undirected graphs, it naturally extends to temporal digraphs and motivates the corresponding covering problems. The objective of Temporal Path Cover (resp. Temporally Disjoint Path Cover) is to cover an input temporal digraph by a minimum number of temporal paths (resp. temporally disjoint paths).

Main objectives. In this paper, we initiate the algorithmic study of Temporal Path Cover and Temporally Disjoint Path Cover and focus on *temporal directed acyclic graphs* (or simply, temporal DAGs). A temporal digraph is a *temporal DAG* if the union of all arcs across all time-steps induces a (static) DAG. We say that a temporal digraph satisfies the *Dilworth property* (resp. *temporally disjoint Dilworth property*, or *TD-Dilworth property* for short) if the largest size of a *temporal antichain* (understood as a set of pairwise temporally unreachable vertices) is equal to the smallest size of a temporal path cover (resp. temporally disjoint path cover). The main goals of this paper are the following:

(a) Determine classes of temporal DAGs satisfying the (TD-)Dilworth property.
(b) Study the computational complexities of Temporal Path Cover and Temporally Disjoint Path Cover on temporal digraphs.

Practical motivation. Similar to the problems studied by Klobas et al. [25], one natural application is *Multi-Agent Path Finding* (MAPF) [13, 40], which can be applied for example to crime prevention in transportation networks [44]. In this setting, k agents are assigned the task of surveying a location: collecting data, moving objects around, looking out for hazards, etc. When the changes over time are predictable (train network, irrigation periods in a field, departure and arrival of vehicles in a logistics area, blockades at given times in a post-disaster area, naturally occurring blockades such as tides, ...), the location is modeled as a temporal digraph. If the location digraph does not contain directed cycles, it is modeled by a temporal DAG (for example, if it is inherently directed from a start area towards a target area). The exploration path of an agent can be modeled by a temporal path. Now, Temporal Path Cover corresponds to the situation where the agents need to explore the whole location, while for Temporally Disjoint Path Cover, the agents also cannot be simultaneously at the same place, a scenario described as *vertex-conflicts* in the literature [39]. In both cases, we want to minimize the number k of agents.

Spatio-temporal security games [42, 43] are a natural example of applicability of MAPF. In these games, defenders want to protect spatially-located resources from attackers by covering them, and have to take time into account whenever they are moving (to be sure that the edges are available when they need to use them). In these games, the temporal dimension is generally represented by adding one layer of space per time-step. This representation induces a DAG, with as many vertices as the size of the initial graph multiplied by the number of time-steps. It seems natural to encode time through the more compact model of temporal graphs, which allows these games to be modeled by Temporal Path Cover and Temporally Disjoint Path Cover.

Our results. We begin by formally defining the problems studied in this paper.

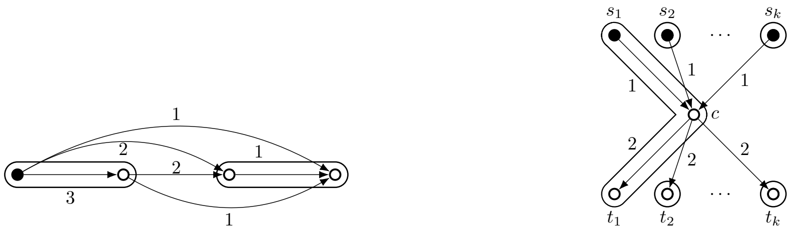
Temporal Path Cover (TPC)
Input: A temporal digraph D, an integer k.
Problem: Does there exist a set $\mathcal{C}$ of k temporal paths in D such that every vertex of D is covered by some path of $\mathcal{C}$?

Temporally Disjoint Path Cover (TD-PC)
Input: A temporal digraph D, an integer k.
Problem: Does there exist a set $\mathcal{C}$ of k temporally disjoint temporal paths in D such that every vertex of D is covered by some path of $\mathcal{C}$?

We observe that in general, temporal DAGs do not have the Dilworth property (see Figure 1a). Then, we prove the following negative result.

▶ **Theorem 1.** *Temporal Path Cover and Temporally Disjoint Path Cover are NP-hard on temporal DAGs, even if the input is planar, bipartite, subcubic, of girth 10, has only one time label per arc, and every label is either* 1 *or* 2*.*

A temporal DAG D is a *temporal oriented tree* if the underlying directed graph of D is a tree. On the positive side, we prove the following.



(a) A temporal DAG not having the Dilworth property.

(b) A temporal oriented tree not having the TD-Dilworth property.

Figure 1 Each encircled area is a path of a minimum-size (temporally disjoint) temporal path cover, vertices in a maximum-size temporal antichain are in black.

▶ **Theorem 2.** *There is an $\mathcal{O}(\ell n^2 + n^3)$-time algorithm for* Temporal Path Cover *on temporal oriented trees with n vertices and at most ℓ many labels per arc. Furthermore, temporal oriented trees satisfy the Dilworth property.*

We briefly describe the technique we use for proving Theorem 2. Two vertices of a temporal digraph are *temporally connected* if there exists a temporal path from one to the other. The *connectivity graph* of a temporal digraph D is an undirected (static) graph whose vertex set is the same as that of D, and whose edge set consists of all pairs of temporally connected vertices. To prove the above theorem, we show that the connectivity graph of a temporal oriented tree is a *weakly chordal graph* [19] (a subclass of *perfect graphs*). We show Temporal Path Cover can be reduced to Clique Cover on weakly chordal graphs. The above observation, combined with the Weak Perfect Graph Theorem (proved by Lovász [30]), proves that temporal oriented trees satisfy the Dilworth property. Moreover, the existing $\mathcal{O}(nm)$-time algorithm [20] to compute a minimum clique cover of a weakly chordal graph (having n vertices and m edges) completes the proof of Theorem 2. Our proof gives interesting structural information on the interaction between temporal paths in temporal oriented trees. Interestingly, another important class of perfect graphs plays an important role in connection with Dilworth's theorem and its translation to the setting of static DAGs: the class of comparability graphs, see [18, Chapter 5.7]. In our case, there does not appear to be any connection to comparability graphs.

On the other hand, temporal oriented trees do not satisfy the TD-Dilworth property (see Figure 1b for an example). Then, we prove the following negative result.

▶ **Theorem 3.** Temporally Disjoint Path Cover *is NP-hard on temporal oriented trees.*

To find classes that satisfy the TD-Dilworth property, we study *temporal oriented lines* (that is, where the underlying digraph is an oriented path) and *temporal rooted directed trees.* A tree is a *rooted directed tree* if it is an oriented tree with a single source vertex called the *root.* We prove the following result.

▶ **Theorem 4.** Temporal Path Cover *and* Temporally Disjoint Path Cover *can be solved in time:*
(a) *$\mathcal{O}(\ell n)$ on temporal oriented lines;*
(b) *$\mathcal{O}(\ell n^2)$ on temporal rooted directed trees;*
where ℓ is the maximum number of labels per arc and n is the number of vertices. Furthermore, both classes satisfy the TD-Dilworth property.

Note that some related problems remain NP-hard for temporal lines, such as Temporally Disjoint Walks [26]. Theorem 4(a) shows that this is not the case here. To prove

Table 1 Summary of our algorithmic results. For all polynomial-time solvable classes of temporal DAGs, we also show that the Dilworth property (or TD-Dilworth property for TD-PC) holds.

temporal graph class	TPC	TD-PC
temporal DAGs (planar bipartite subcubic, girth 10, two arc-disjoint time-steps)	NP-c.	NP-c.
temporal oriented trees	poly $\mathcal{O}(\ell n^2 + n^3)$	NP-c.
temporal rooted directed trees	poly $\mathcal{O}(\ell n^2)$	poly $\mathcal{O}(\ell n^2)$
temporal oriented lines	poly $\mathcal{O}(\ell n)$	poly $\mathcal{O}(\ell n)$
general temporal digraphs with bounded treewidth tw and number of time-steps $t_{\max}$	poly (XP w.r.t. tw and $t_{\max}$)	poly (FPT w.r.t. tw and $t_{\max}$)

Theorem 4(b), we begin by constructing a temporal path cover before transforming it into a temporally disjoint one of the same size. This is in contrast with general temporal oriented trees, for which, by Theorem 3, such an approach is not possible.

As Temporally Disjoint Path Cover is NP-hard even on temporal oriented trees and on temporal DAGs with two time-steps, a natural question is what happens when the number of time-steps is small *and* the underlying digraph is a tree. Motivated by this question, we study the case where both the number of time-steps and the treewidth of the underlying digraph are bounded (where we define the *treewidth* of a temporal digraph as the treewidth of the underlying static undirected graph). We show that both problems become tractable in this setting. More precisely, we give a fixed-parameter tractable (FPT) algorithm for Temporally Disjoint Path Cover with treewidth and number of time-steps as parameters. The same technique gives an XP algorithm for Temporal Path Cover.

▶ **Theorem 5.** *There is an algorithm for Temporally Disjoint Path Cover on general temporal digraphs that is FPT with respect to the treewidth of the underlying undirected graph and the maximum number of labels per arc. For Temporal Path Cover on general temporal digraphs, there is an XP algorithm for the same parameter.*

See Table 1 for a summary of our algorithmic results.

Further related work. Algorithms for solving several types of path and distance problems in temporal graphs have been developed, see for example [3, 24, 41]. Recently, the problem Temporally Disjoint Paths was introduced in [25], as a generalization of the notorious Disjoint Paths problem (also known as Linkage). In Temporally Disjoint Paths, one is given a temporal graph with k pairs of vertices called *terminals*, and the goal is to find a set of k pairwise temporally disjoint paths, each of them connecting one pair of terminals. Temporally Disjoint Paths is NP-hard, even for temporal lines and two paths [25] or temporal stars [29], but becomes FPT for trees when parameterized by the number of paths [25]. Algorithms that are FPT for certain structural parameters are given in [29].

Structure of the paper. We start with the hardness result for temporal DAGs (Theorem 1) in Section 3. We then prove our results for temporal oriented trees (Theorem 2 and Theorem 3) in Sections 4 and 5. We prove Theorem 4, the polynomial-time algorithms for special temporal oriented trees (temporal rooted directed trees and temporal oriented lines), in Section 6. We then prove our results for temporal digraphs of bounded treewidth and number of time-steps

(Theorem 5) in Section 7. We conclude in Section 8. Due to space constraints, proofs of propositions and lemmas marked with **(*)** are omitted here and can be found in the full version of the paper [9].

2 Preliminaries

A *temporal digraph* $\mathcal{D} = (V, A_1, \dots, A_{t_{\max}})$ is given by a sequence of arc-sets representing $t_{\max}$ discrete *time-steps* $\{1, \dots, t_{\max}\}$, where an arc in A_i is *active* at time-step i [25]. We denote by $D = (V, A)$, where $A = \cup_{i=1}^{t_{\max}} A_i$, the *underlying digraph* of $\mathcal{D}$ (sometimes called the *footprint (di)graph* [7]). Equivalently, one can view the time-steps as an arc-labelling function $\lambda : A(D) \to 2^{[t_{\max}]}$, where $\lambda(\overrightarrow{xy}) \subseteq [1, t_{\max}]$ is the set of time-steps where $\overrightarrow{xy}$ is active [24]. In that case, we denote the temporal digraph as $\mathcal{D} = (D, \lambda)$. We say that a temporal digraph has a given property $\mathcal{P}$ (planarity, given girth, ...) if the undirected graph obtained by forgetting the orientation of the arcs of its underlying digraph has property $\mathcal{P}$. Similarly, we call a temporal digraph a temporal DAG (resp. temporal oriented tree, temporal line, ...) if its underlying digraph is a DAG (resp. oriented tree, path, ...). For a given temporal digraph, we denote by ℓ the maximum number of labels per arc and by n the number of vertices in the underlying digraph.

For a (temporal) (di)graph $\mathcal{D}$ and subset S of its vertices (resp. edges), $\mathcal{D} \setminus S$ denotes the (temporal) (di)graph obtained by removing the vertices (resp. edges) in S from $\mathcal{D}$.

In a temporal digraph, a *temporal (directed) path* is a sequence $(v_1, v_2, t_1), (v_2, v_3, t_2), \dots,$ (v_{k-1}, v_k, t_{k-1}) such that for any i, j with $1 \leq i < j \leq k$, $v_i \neq v_j$ and for any i with $1 \leq i \leq k-1$, $t_i < t_{i+1}$ and there is an arc $\overrightarrow{v_i v_{i+1}}$ at time-step t_i. These paths are sometimes called *strict* in the literature.[1] For a temporal path $P = (v_1, v_2, t_1), \dots, (v_{k-1}, v_k, t_{k-1})$, we denote by $V(P)$ the set $\cup_{i=1}^{k} \{v_i\}$ and by $A(P)$ the set $\cup_{i=1}^{k-1} \{\overrightarrow{v_i v_{i+1}}\}$. Note that we allow a temporal path to contain exactly one vertex and no arc.

The *length* of a temporal path is the number of arcs it uses. We say that a temporal path $P = (v_1, v_2, t_1), \dots, (v_{k-1}, v_k, t_{k-1})$ *occupies* vertex v_i during the time interval $[t_{i-1}, t_i]$. Two temporal paths P_1, P_2 *temporally intersect* if there is a vertex $v \in V(P_1) \cap V(P_2)$ and two time intervals $[a_1, b_1], [a_2, b_2]$ where $[a_1, b_1] \cap [a_2, b_2] \neq \emptyset$ such that P_1 (resp. P_2) occupies v during $[a_1, b_1]$ (resp. $[a_2, b_2]$). Two temporal paths are *temporally disjoint* if they do not temporally intersect. In other words, they do not occupy the same vertex at the same time. A *temporal path cover* (resp. *temporally disjoint path cover*) of a temporal digraph D is a collection of temporal paths (resp. temporally disjoint paths) that cover all vertices of D. Two vertices are *temporally connected* in D if there exists a temporal path between them. A *temporal antichain* is a set of vertices that are pairwise not temporally connected.

▶ **Definition 6.** *A class $\mathcal{C}$ has the* Dilworth property *(resp.* TD-Dilworth property*) if, for every digraph $D \in \mathcal{C}$, the cardinality of a minimum temporal path cover (resp. temporally disjoint path cover) of D is equal to the maximum cardinality of a temporal anti-chain in D.*

A *hole* of a static undirected graph is an induced cycle of length at least 5, and an *anti-hole* is the complement of a hole. A hole or anti-hole is *even* (resp. *odd*) if it has an even (resp. odd) number of vertices. A graph G is *weakly chordal* if it has neither a hole nor an anti-hole. A *(minimum) clique cover* of a graph G is a (minimum cardinality) set of complete subgraphs of G that covers all vertices. A *(maximum) independent set* of a graph G is a (maximum cardinality) set of pairwise non-adjacent vertices. We shall use the following results for weakly chordal graphs.

[1] For non-strict paths, the condition $t_i < t_{i+1}$ is replaced with $t_i \leq t_{i+1}$; argued in [29], the strict definition is more natural for applications where an agent cannot traverse any number of arcs at once.

▶ **Theorem 7** ([20, 30, 38]). *Let H be a weakly chordal graph with n vertices and m edges. Then, a minimum clique cover of H can be found in $\mathcal{O}(nm)$-time. Furthermore, the maximum size of an independent set of H equals the minimum size of a clique cover of H.*

3 Temporal DAGs

We provide a reduction from a restricted variant of 3-Dimensional Matching to prove the following (proof deferred to the full version [9] due to space constraints).

▶ **Theorem 1.** *Temporal Path Cover and Temporally Disjoint Path Cover are NP-hard on temporal DAGs, even if the input is planar, bipartite, subcubic, of girth 10, has only one time label per arc, and every label is either 1 or 2.*

We also show the following.

▶ **Proposition 8** (*). *There are temporal DAGs (whose underlying digraph is a transitive tournament) that satisfy neither the Dilworth nor the TD-Dilworth property. Moreover, the ratio between the minimum-size temporal path cover and the maximum-size temporal antichain can be arbitrarily large.*

4 Temporal Path Cover on temporal oriented trees

In this section we prove the following theorem.

▶ **Theorem 2.** *There is an $\mathcal{O}(\ell n^2 + n^3)$-time algorithm for Temporal Path Cover on temporal oriented trees with n vertices and at most ℓ many labels per arc. Furthermore, temporal oriented trees satisfy the Dilworth property.*

For the rest of this section, $\mathcal{T} = (T, \lambda)$ shall denote a temporal oriented tree with n vertices and at most ℓ-many labels per edge. We construct the *connectivity graph of* $\mathcal{T}$, denoted by G, as follows: $V(G) = V(T)$ and $E(G) = \{uv \mid u \neq v \text{ and } u \text{ and } v \text{ are temporally connected}\}$. In other words, the connectivity graph of a temporal oriented tree connects vertices that are temporally connected. Observe that G can be constructed in $\mathcal{O}(\ell n^2)$-time. The next observation follows immediately from the definition.

▶ **Observation 9.** *A set S of vertices of $\mathcal{T}$ is a temporal antichain if and only if S induces an independent set in G.*

We have the following relationship between temporal paths in $\mathcal{T}$ and cliques in G.

▶ **Lemma 10.** *Let S be a set of vertices of $\mathcal{T}$. Then S is contained in a temporal path in $\mathcal{T}$ if and only if S is contained in a clique of G.*

Proof. Let S be contained in temporal path P in $\mathcal{T}$. Let $u_1, u_2, \ldots, u_k$ where $k = |S|$, be the ordering of the vertices in S as they are encountered while traversing P from the source to the sink. Notice that, for each $1 \leq i < j \leq k$, there is a temporal path from u_i to u_j. Therefore, u_i is adjacent to u_j in G. Hence, S is contained in a clique of G.

Let S be contained in a clique of G and S' be a maximal complete subgraph of G such that $S \subseteq V(S')$. Now, we orient the edges of S' to create a digraph $\overrightarrow{S'}$ as follows. For an edge $uv \in E(S')$, we introduce an arc $\overrightarrow{uv} \in A(\overrightarrow{S'})$ if there is a temporal path from u to v in $\mathcal{T}$. Since $\mathcal{T}$ is acyclic, $\overrightarrow{S'}$ is a transitive tournament. Hence, there is an ordering $u_1, u_2, \ldots, u_k$ of the vertices of S' where $k = |V(S')|$ such that for $1 \leq i < j \leq k$, there is a temporal path from u_i to u_j in $\mathcal{T}$. Now, consider any temporal path P from u_1 to u_k in $\mathcal{T}$. (P exists as $\overrightarrow{u_1u_k} \in A(\overrightarrow{S'})$. Since $\mathcal{T}$ is a temporal oriented tree, P will contain all vertices of S' and therefore of S. ◀

Following is an immediate corollary of the above.

▶ **Corollary 11.** *The minimum cardinality of a temporal path cover of $\mathcal{T}$ is equal to the minimum cardinality of a clique cover of G.*

We will often use the following lemma.

▶ **Lemma 12.** *Let $\{u, v, w, x\} \subseteq V(T)$ be four vertices such that there is a temporal path P_1 from u to v and a temporal path P_2 from w to x. If P_1 and P_2 have a vertex in common, then, there is a temporal path from u to x, or a temporal path from w to v, or both.*

Proof. Assume that there is no temporal path from u to x. Let y be the vertex of a temporal path from w to x that is closest to u in T. Let t be the smallest integer such that there is a temporal path from u to v that reaches y at time-step t. Observe that no temporal path from y to x can start at time-step $t' > t$ since, otherwise, there would be a temporal path from u to x. This implies that all temporal paths between w and x reach y at time-step $t'' \leq t$. Let P_1 be a temporal path from w to y which is also a subpath of a temporal path from w to x. Let P_2 be a temporal path from y to v which is also a subpath of a temporal path from u to v. The above arguments imply that the arc incident with y in P_1 has time-step at most t. Similarly, the arc incident with y in P_2 has time-step strictly greater than t. Hence, the concatenation of P_1 and P_2 is a temporal path in $\mathcal{T}$ from w to v. ◀

4.1 The case of holes

In this subsection, we will show that the connectivity graph G does not contain any holes. We use the following lemma.

▶ **Lemma 13.** *Let H be an induced cycle of length at least 4 in G. Then, for every vertex $v \in V(H)$ and every arc $\overrightarrow{a}$ of T incident with v, the vertices of $H \setminus \{v\}$ lie in the same weakly connected component of $T \setminus \{\overrightarrow{a}\}$.*

Proof. For the sake of contradiction, let there exist vertices $\{u, v, w\} \subseteq V(H)$ and an arc $\overrightarrow{a}$ of T incident with v such that u and w lie in two different connected components of $T' = T \setminus \{\overrightarrow{a}\}$. Let C_u and C_w be the sets of vertices of $H \setminus \{v\}$ contained in the same connected component as u and w, respectively. Since $H \setminus \{v\}$ is connected, there exist $u' \in C_u$ and $w' \in C_w$ such that $u'w' \in E(H)$ *i.e.* $u'w' \in E(G)$. Hence, there is a temporal path P from u' to w' or w' to u' in $\mathcal{T}$. Since T is a tree, P must contain v. Lemma 10 implies that $\{u', v, w'\}$ forms a subset of a clique in G, and therefore $\{u', v, w'\}$ forms a triangle. But this contradicts that H is a hole. ◀

Going forward, we need the following notations. For an edge $e = uv \in E(G)$, let Q_e denote a temporal path from u to v or v to u in $\mathcal{T}$. For an induced cycle H of length at least 4 in G, let T_H denote the smallest connected subtree of T containing all vertices of H. Lemma 13 implies that every vertex of H must be a leaf in T_H (that is, a vertex with degree 1 in the underlying undirected tree). For a vertex $v \in V(H)$, let $\overrightarrow{a}(v)$ be the arc incident with v in T_H. Let H be an induced cycle of length at least 4 in G. We can partition the vertex set of H into two sets $IN(H)$ and $OUT(H)$ as follows: a vertex $v \in V(H)$ is in

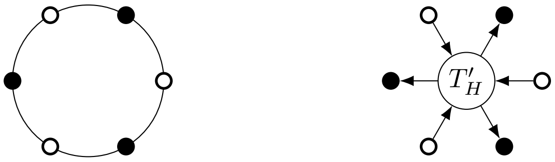


Figure 2 On the left, a hole H in the connectivity graph. On the right, its corresponding vertices in the oriented subtree T_H, with $T'_H = T_H \setminus V(H)$. Vertices in $IN(H)$ are in black.

$IN(H)$ if $\overrightarrow{a}(v)$ is directed towards v, and otherwise v is in $OUT(H)$ (see Figure 2 for an illustration of these definitions).

For a vertex $v \in IN(H)$, notice that both neighbors of v in H must lie in $OUT(H)$, and vice versa, since they must be connected by a directed path in T. Hence, H is bipartite, and therefore G does not contain any odd hole:

▶ **Lemma 14.** *The connectivity graph G does not contain any odd hole.*

Without loss of generality, we assume in the following that $OUT(H)$ (resp. $IN(H)$) contains every odd-indexed (resp. even-indexed) vertex of H. For an even hole H whose vertices are cyclically ordered as $u_1, u_2, \ldots, u_k$, we use a cyclic definition of addition, so $u_{k+1} = u_1$. We first prove the following lemmas.

▶ **Lemma 15.** *Let H be an even hole in the connectivity graph G. Then, for every i, $Q_{u_iu_{i+1}}$ and $Q_{u_{i+2}u_{i+3}}$ share a common vertex in T.*

Proof. Assume by contradiction that $Q_{u_iu_{i+1}}$ and $Q_{u_{i+2}u_{i+3}}$ are vertex-disjoint. Assume without loss of generality that $Q_{u_iu_{i+1}}$ goes from u_i to u_{i+1}. Note that, since each vertex of the hole is a leaf of T_H as a consequence of Lemma 13, the two paths $Q_{u_iu_{i+1}}$ and $Q_{u_{i+1}u_{i+2}}$ have to share a common vertex other than u_{i+1} (its neighbour in T_H). By the same reasoning, $Q_{u_{i+1}u_{i+2}}$ and $Q_{u_{i+2}u_{i+3}}$ share a common vertex other than u_{i+2}. Hence, since the three paths $Q_{u_iu_{i+1}}$, $Q_{u_{i+1}u_{i+2}}$ and $Q_{u_{i+2}u_{i+3}}$ are in T_H, and $Q_{u_iu_{i+1}}$ and $Q_{u_{i+2}u_{i+3}}$ are vertex-disjoint, there is an arc $\overrightarrow{a}$ contained in $Q_{u_{i+1}u_{i+2}}$ that separates $Q_{u_iu_{i+1}}$ and $Q_{u_{i+2}u_{i+3}}$.

Removing $\overrightarrow{a}$ from T partitions the vertices of H into two sets H_1 and H_2: H_1 (resp. H_2) contains the vertices of H that are in the same part of $T \setminus \overrightarrow{a}$ as u_{i+1} (resp. u_{i+2}). Now, since H is a cycle, there is an edge u_ju_{j+1} such that (without loss of generality) $u_j \in H_1$, $u_{j+1} \in H_2$ and $(j, j+1) \neq (i+1, i+2)$. This implies that the path $Q_{u_ju_{j+1}}$ has to use $\overrightarrow{a}$ in $\mathcal{T}$, and thus $Q_{u_{i+1}u_{i+2}}$ and $Q_{u_ju_{j+1}}$ share a common vertex. Hence, Lemma 12 implies that there is a temporal path from u_{j+1} to u_{i+1} or from u_{i+2} to u_j. However, since $j \neq i+3$ ($u_j \in H_1$ and $u_{i+3} \in H_2$) and $j+1 \neq i$ ($u_{j+1} \in H_2$ and $u_i \in H_1$), both temporal paths would induce a chord in H, a contradiction. ◀

▶ **Lemma 16.** *The connectivity graph G does not contain any hole of size 6.*

Proof. Assume by contradiction that there is a hole on six vertices $u_1, \ldots, u_6$. We know that $Q_{u_1u_2}$ and $Q_{u_4u_5}$ are vertex-disjoint (since otherwise, by Lemma 12, at least one of the chords u_1u_4 or u_2u_5 would exist). The u_i's are leaves of T_H, so $Q_{u_1u_2}$ and $Q_{u_1u_6}$, being paths with a common leaf in the same subtree, share at least one common vertex other than u_1 (its neighbour in T_H), let v be the last (with respect to the orientation of T) vertex in their common subpath. Now, $Q_{u_5u_6}$ has a common vertex with both $Q_{u_1u_2}$ (by Lemma 15) and $Q_{u_1u_6}$ (the neighbour of u_6 in T_H), so it has to contain v by the Helly property of subtrees of a tree. By the same reasoning, $Q_{u_4u_5}$ and $Q_{u_5u_6}$ share at least one common vertex other than u_5 (its neighbour in T_H), let w be the last vertex in their common subpath. The Helly property of subtrees of a tree again implies that both $Q_{u_2u_3}$ and $Q_{u_3u_4}$ have to contain w, since they pairwise intersect with $Q_{u_4u_5}$. But this means that $Q_{u_2u_3}$ and $Q_{u_5u_6}$ share both v and w as common vertices, and so by Lemma 12 there is at least one of the two chords u_2u_5 or u_3u_6, a contradiction. ◀

We can now prove that there is no even hole in G:

▶ **Lemma 17.** *The connectivity graph G does not contain any even hole.*

Proof. Assume by contradiction that G contains an even hole H on $k \geq 8$ vertices ($k = 6$ is impossible by Lemma 16). We know by Lemma 15 that both $Q_{u_3u_4}$ and $Q_{u_{k-1}u_k}$ intersect $Q_{u_1u_2}$, but do not intersect each other (otherwise, by Lemma 12, at least one of the edges u_3u_k or u_4u_{k-1} would exist, and both would be chords since $k \geq 8$), so there is an arc $\overrightarrow{a}$ in T that separates them. Removing $\overrightarrow{a}$ from T partitions the vertices of H into two sets H_1 and H_2: H_1 (resp. H_2) contains the vertices of H that are in the same part of $T \setminus \overrightarrow{a}$ as u_3 (resp. u_k). Now, since H is a cycle, there is an edge u_ju_{j+1} such that (without loss of generality) $u_j \in H_1$ and $u_{j+1} \in H_2$. This implies that the path $Q_{u_ju_{j+1}}$ has to use $\overrightarrow{a}$ in $\mathcal{T}$, and thus $Q_{u_1u_2}$ and $Q_{u_ju_{j+1}}$, both containing $\overrightarrow{a}$, share a common vertex. Hence, Lemma 12 implies that there is a temporal path from u_{j+1} to u_2 or from u_1 to u_j. However, since $j \neq k$ ($u_j \in H_1$ and $u_k \in H_2$) and $j + 1 \neq 3$ ($u_{j+1} \in H_2$ and $u_3 \in H_1$), by Lemma 12 both temporal paths would induce a chord in H, a contradiction. ◀

4.2 The case of anti-holes

In this subsection, we will show that the connectivity graph G does not contain any anti-hole. For an anti-hole H, let its vertices be circularly ordered as $u_1, u_2, \ldots, u_k$ as they are encountered while traversing the complement of H (which is a hole). Let $ODD(H)$ (resp. $EVEN(H)$) denote the set of vertices with odd (resp. even) indices.

▶ **Lemma 18.** *The connectivity graph G does not contain any anti-hole.*

Proof. Throughout this proof, recall that T is a tree, in particular, if two vertices are temporally connected, then there is a unique temporal path from one to the other. Assume by contradiction that G contains an anti-hole H with k vertices. If $k = 5$, then H is a hole, which contradicts Lemma 14; hence, assume $k \geq 6$.

When k is odd, let $F_1 = ODD(H) \setminus \{u_k\}, F_2 = EVEN(H)$. When k is even, let $F_1 = ODD(H), F_2 = EVEN(H)$. Observe that $|F_1| = |F_2| \geq 3$ and both sets induce (vertex-disjoint) cliques in G. By Lemma 10, there are temporal paths P_1 and P_2 in $\mathcal{T}$ containing F_1 and F_2, respectively, which we can assume are minimal vertex-inclusion-wise (so that, for each $i \in \{1, 2\}$, both end-vertices of P_i lie in F_i). For $i \in \{1, 2\}$, let v_i and w_i denote the source and sink of P_i, respectively. We have two cases.

Case 1: $V(P_1) \cap V(P_2) = \emptyset$. Let Q be the shortest temporal path that contains vertices from both P_1 and P_2 (note that, since every vertex in F_1 is temporally connected to at least one vertex in F_2, Q necessarily exists). Let p_1, p_2 be the end-vertices of Q that lie on P_1 and P_2, respectively. Since for each $i \in \{1, 2\}$ and $Z \in \{F_1, F_2\}$, $N_G(w_i) \cap Z \neq \emptyset$, Q is oriented from p_1 to p_2, or vice versa. Without loss of generality, assume that Q is oriented from p_2 to p_1. Then, necessarily $p_2 = w_2$, since otherwise w_2 is not temporally connected with any vertex of F_1, a contradiction. By a similar argument, we have $p_1 = v_1$. Now, consider the clique induced by $N_G(v_2) \cap F_1$. Due to Lemma 10, all vertices of $N_G(v_2) \cap F_1$ and v_2 itself are contained in a temporal path, which also necessarily contains w_2. Hence all of F_2 (P_2, even) is in a temporal path containing v_1, since the path has to go through v_1 to reach other vertices of F_1, and so $F_2 \cup \{v_1\}$ forms a clique. This is a contradiction as v_1 necessarily has at least one non-neighbor in F_2.

Case 2: $V(P_1) \cap V(P_2) \neq \emptyset$. Let Q denote the maximal vertex-inclusion-wise path that is common to both P_1 and P_2, *i.e.*, the path induced by the set $V(P_1) \cap V(P_2)$. Note that Q does not contain any vertex from H, since a vertex of H in Q would be temporally connected to every other vertex of $F_1 \cup F_2$, a contradiction. Let p denote source of Q and for each $i \in \{1, 2\}$ let Q_i (resp. Q'_i) be the subpath of P_i between p and w_i (resp. p and v_i).

Note that no vertex of $Q'_1 \setminus \{p\}$ can be in a directed path with any vertex of $Q'_2 \setminus \{p\}$. Similarly, no vertex of $Q_1 \setminus Q$ can be in a directed path with any vertex of $Q_2 \setminus Q$. Thus, the two subgraphs of the connectivity graph G induced by the vertices of $(V(Q_1) \cup V(Q_2)) \setminus V(Q)$ and $(V(Q'_1) \cup V(Q'_2)) \setminus \{p\}$ each induce the complement of a complete bipartite graph. As H does not contain any complement of a 4-cycle as an induced subgraph, this implies that there are exactly three vertices of H in each of these two subsets of vertices (since Q does not contain any vertex of H). In particular, H has size either 6 or 7.

Without loss of generality, we assume that Q'_1 contains only one vertex of H, which must be v_1. Thus, there are two vertices of H in Q'_2: v_2 and another vertex, say, v'_2. Since F_1 and F_2 both have size 3, the vertices of H in Q_1 are w_1 and (say) w'_1, and the only vertex of H in Q_2 is w_2. Now, observe that if v_2 is contained in a temporal path with w_1, then v_2, v'_2, w'_1 and w_1 are in a common temporal path. This is not possible, since in H, there is either one or two non-edges among these four vertices (depending on whether H has size 7 or 6). Thus, w_1 and v_2 are in no common temporal path. Since v_2 has no non-neighbour in H other than v_1 and w_1, v_2 and w'_1 are in a common temporal path, that also contains v'_2. Thus, $\{v_2, v'_2, w'_1\}$ form a clique in H. Similarly, $\{v'_2, w'_1, w_1\}$ also form a clique in H. If H had size 6, v'_2 and w'_1 would need to be non-neighbours in H (since w_1 already has two non-neighbours in H), a contradiction. Thus, H has size 7, and the two non-neighbours in H of u_7 (the vertex of H not in $F_1 \cup F_2$) are v'_2 and w'_1 (since they are the only ones without two non-neighbours in H). But u_7 has to be temporally connected to all of v_1, v_2, w_1 and w_2, so u_7 has to be in Q. But any temporal path from v_2 to a vertex of Q has to contain v'_2, and so u_7 and v'_2 are temporally connected, a contradiction. This completes the proof. ◀

4.3 Completion of the proof of Theorem 2

Lemmas 14, 17, and 18 imply that the connectivity graph of a temporal oriented tree is weakly chordal. Note that this cannot be strengthened to chordal, as there are temporal oriented trees whose connectivity graphs contain induced 4-cycles: let $\lambda(\overrightarrow{s_1c}) = \lambda(\overrightarrow{s_2c}) = 1$ and $\lambda(\overrightarrow{ct_1}) = \lambda(\overrightarrow{ct_2}) = 2$, the vertices s_1, t_1, s_2 and t_2 induce a C_4 in the connectivity graph. Corollary 11 implies the correspondence between a minimum temporal path cover of a temporal oriented tree and a minimum clique cover of the corresponding connectivity graph. We then conclude using Theorem 7 for the algorithm. Observation 9, Corollary 11 and Theorem 7 together give the Dilworth property.

5 Temporally Disjoint Path Cover on temporal oriented trees

We provide a reduction from Unary Bin Packing to prove the following (proof deferred to the full version [9] due to space constraints).

▶ **Theorem 3.** *Temporally Disjoint Path Cover is NP-hard on temporal oriented trees.*

We also show the following.

▶ **Proposition 19 (*).** *There are temporal oriented trees (whose underlying digraph is a star) that do not satisfy the TD-Dilworth property.*

6 Subclasses of temporal oriented trees

▶ **Theorem 4.** *Temporal Path Cover and Temporally Disjoint Path Cover can be solved in time:*

(a) *$\mathcal{O}(\ell n)$ on temporal oriented lines;*

(b) *$\mathcal{O}(\ell n^2)$ on temporal rooted directed trees;*

where ℓ is the maximum number of labels per arc and n is the number of vertices. Furthermore, both classes satisfy the TD-Dilworth property.

Proof. (a) Let $\mathcal{P} = (P, \lambda)$ be a temporal oriented line, and let v be a leaf of P. We construct $\mathcal{C}$ as follows. Assume that v is incident with an in-arc $\overrightarrow{uv}$. We construct a maximum-length temporal path that covers v. Set $(b, c) = (u, v)$, $\ell = \max \lambda(\overrightarrow{uv})$, and apply the following routine: while b is incident with an in-arc $\overrightarrow{ab}$, if there is a time label smaller than ℓ in $\lambda(\overrightarrow{ab})$, add $\overrightarrow{ab}$ to the path, update $(b, c) = (a, b)$ and $\ell = \max\{k \in \lambda(\overrightarrow{ab}) \mid k < \ell\}$. When the routine stops, add the path to $\mathcal{C}$, remove its vertices from P, and start again on a new leaf (or return $\mathcal{C}$ if P is empty). If v was incident with an out-arc, we would do the same but with out-arcs, start with the smallest possible time label, and update $\ell = \min\{k \in \lambda(\overrightarrow{ab}) \mid k > \ell\}$.

This algorithm computes its output in time $\mathcal{O}(\ell n)$: every arc is visited at most once, but we need to parse the time labels in order to see whether we can keep on extending the path or not. Furthermore, the set of leaves v where we start the routine are a temporal antichain: assume on the contrary that v_1 and v_2 are such vertices that are temporally connected, and assume without loss of generality that there is a path from v_1 to v_2 in the underlying oriented path; in this case, our algorithm would have added v_1 to the path that started being computed at v_2, a contradiction. Hence, $\mathcal{C}$ is a temporally disjoint path cover with the same size as a temporal antichain, proving that it is minimum-size and that temporal oriented lines satisfy the TD-Dilworth property.

(b) We give an algorithm that solves Temporal Path Cover on a temporal rooted directed tree $\mathcal{T} = (T, \lambda)$. First, we sort the vertices of T with respect to their topological distance from the root in T (with the highest distances first). Then, we construct a maximum-length temporal path covering the first uncovered vertex (which will be a sink of that path), and repeat until $\mathcal{T}$ is fully covered.

Note that this algorithm outputs $\mathcal{C}$ which is clearly a temporal path cover: every vertex is covered by some path of $\mathcal{C}$. Furthermore, it is an adaptation of the algorithm for temporal oriented lines: instead of successive leaves, we construct the paths from successive leaves with highest topological distance from the root. We will show that $\mathcal{C}$ is minimum-size, and later we will explain how to modify the algorithm in order to obtain a minimum-size temporally disjoint path cover.

Let S be the set of sinks of paths of $\mathcal{C}$. First, let v_i and v_j be two vertices of S (without loss of generality, assume that v_i was covered by the algorithm after v_j). They cannot be temporally connected, since otherwise, the graph being a temporal rooted directed tree, one of them is necessarily the predecessor of the other in a path from the root, and thus the maximum-length temporal path ending in v_j would necessarily contain v_i, since there is a temporal path from v_i to v_j, and thus v_i would have been covered at this step and cannot be in S. Hence, S is a temporal antichain. Furthermore, since $|\mathcal{C}| = |S|$, $\mathcal{C}$ is minimum-size and S is maximum-size, implying that temporal rooted directed trees satisfy the Dilworth property.

We now modify the algorithm to obtain a minimum-size temporally disjoint path cover. Indeed, we can see that the maximum-length temporal path construction, which is executed for every vertex of S, can re-cover some vertices that had already been covered at a previous step. Let v_i and v_j be two vertices of S such that their maximum-length temporal paths constructed by the algorithm P_i and P_j intersect. Since the graph is a temporal rooted directed tree, we can divide P_i and P_j into the following parts, without loss of generality: $P_i = P_i^{\text{top}} \cup (P_i \cap P_j) \cup P_i^{\text{bot}}$ and $P_j = (P_i \cap P_j) \cup P_j^{\text{bot}}$, where $P_i^{\text{top}} \cap P_j = P_i^{\text{bot}} \cap P_j = P_j^{\text{bot}} \cap P_i = \emptyset$ (note that we can have $P_i^{\text{top}} = \emptyset$). In other words, P_i^{top} (resp. P_i^{bot}) contains the vertices of $P_i \setminus P_j$ that are ancestors (resp. descendants) of $P_i \cap P_j$ in the underlying rooted tree, with the equivalent definition for P_j^{top} and P_j^{bot}. Hence, we can modify the algorithm by adding a loop that, for each such pair (P_i, P_j), defines these subpaths and then removes $P_i \cap P_j$ from P_j. Now, $\mathcal{C}$ will still be a temporal path cover, but the paths will be vertex-disjoint and thus temporally disjoint, and its size will not change. This implies that temporal rooted directed trees satisfy the TD-Dilworth property (contrasting the general temporal oriented trees), and thus the modified algorithm outputs the optimal solution for these two problems. The result of the algorithm and its modification is depicted in Figure 3.

Finally, one can check that the algorithm and its modification compute $\mathcal{C}$ in time $\mathcal{O}(\ell n^2)$. For each vertex in the antichain S, we have to construct the maximum-length temporal path. This can be done in time $\mathcal{O}(\ell n)$ by taking at every arc the largest label that allows to extend the path, thus we have to parse all the labels of every arc along the path, which can be of linear-size in the worst case. Since we can have a linear number of antichain vertices, we have a complexity of $\mathcal{O}(\ell n^2)$ to get the temporal path cover. The modification to make it temporally disjoint can be done in $\mathcal{O}(n^2)$ time afterwards, by considering all pairs of intersecting paths starting from the root. ◀

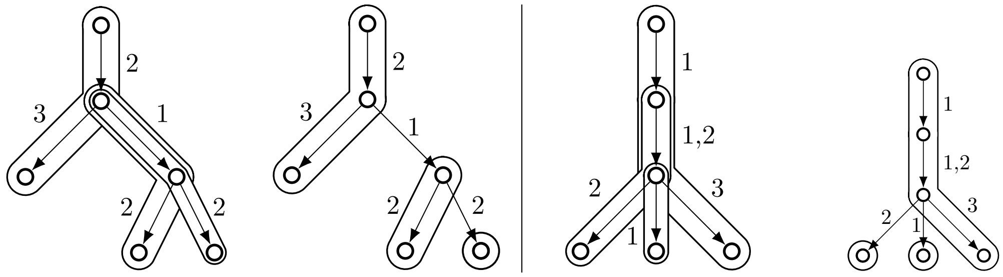


Figure 3 Minimum-size temporal path covers and temporally disjoint path covers of the same temporal rooted directed tree (on the left, with one label per arc; on the right, with any labels per arc), as computed by our algorithm and its modification in the proof of Theorem 4.

7 Algorithms for temporal digraphs of bounded treewidth

Recall that an algorithm is FPT with respect to some parameter k of the input, if it runs in time $f(k)n^{\mathcal{O}(1)}$ for inputs of size n, where f is any computable function; the algorithm is XP for this parameter if the running time is in $n^{f(k)}$ [12]. We prove the following theorem.

▶ **Theorem 5.** *There is an algorithm for* Temporally Disjoint Path Cover *on general temporal digraphs that is FPT with respect to the treewidth of the underlying undirected graph and the maximum number of labels per arc. For* Temporal Path Cover *on general temporal digraphs, there is an XP algorithm for the same parameter.*

Proof (sketch). To prove the theorem, we use the well-known concept of *nice tree decompositions* [27]. The algorithm is a classic bottom-up dynamic programming. To simplify the algorithm, we replace each arc by a set of parallel arcs, each of them with a distinct time-label. This makes it easier to deal with temporally disjoint paths, as in this representation, one arc cannot be used in two solution paths (for Temporally Disjoint Path Cover).

To give the intuition behind the algorithm, we informally describe the states of the dynamic programming. For a bag $X_v \subseteq V(G)$ corresponding to a node v of the tree decomposition, every state corresponds to a distinct way a potential solution interacts with X_v. Primarily, it consists of a set of (solution) subpaths that cover the vertices in X_v. Each subpath may consist of several disconnected parts (possibly, a part may have only one vertex). The order in which the vertices of each solution path appear is also specified. We also store the information, for every vertex, whether it is connected via an arc to one vertex (or two vertices) in its solution path, but lying outside the bag X_v, and whether this vertex lies in a lower (already handled) or upper (to be handled) part of the tree decomposition.

We show that this information is enough to encode a partial solution, and as there are at most $p = \binom{\text{tw}}{2} \cdot t_{\max}$ arcs in each bag, we deduce that the maximum possible number of states for a bag is at most $2^{O(p \log p)}$ in the case of Temporally Disjoint Path Cover (since every arc may appear in at most one solution path) and $n^{O(p \log p)}$ in the case of Temporal Path Cover (since a specific subpath may appear in any number of solution paths, so we must also encode the number of appearances of each subpath). The running times are essentially dominated by these functions.

Due to space constraints, the details are deferred to the full version [9]. ◀

8 Conclusion

We have initiated the study of two natural path covering problems in temporal DAGs, which, in the static case, are related to Dilworth's theorem and are polynomial-time solvable. Both problems become NP-hard for temporal DAGs, even in a very restricted setting. Interestingly, and somewhat unexpectedly, they behave differently on temporal oriented trees: we showed that Temporal Path Cover is polynomial-time solvable on temporal oriented trees (and a temporal version of Dilworth's theorem holds in this setting), while Temporally Disjoint Path Cover remains NP-hard for this class. On the other hand, this distinction is inverted in the parameterized case, where we obtained an FPT algorithm for Temporally Disjoint Path Cover but an XP algorithm for Temporal Path Cover. However, we do not know if our algorithms for treewidth and number of time-steps are optimal. In particular, can we obtain an FPT algorithm for Temporal Path Cover for this parameter? One could also explore other (structural) parameterizations of the problems.

To prove our polynomial-time algorithm for Temporal Path Cover on temporal oriented trees, we have reduced the problem to Clique Cover in a static undirected graph, which turns out to be weakly chordal. This is a powerful technique, and the correspondence between the two problems is quite enlightening for the structure of temporal paths in an oriented tree. Nevertheless, it seems unlikely that this particular technique can be used on temporal digraph classes that are far from trees, as it was essential for the proof that any two vertices are joined by only one path in the underlying tree. However, this general technique could likely be applied in other temporal settings.

We note that many of our results for Temporally Disjoint Path Cover also hold for its stricter vertex-disjoint version (note that a vertex-disjoint version of Temporally Disjoint Paths is studied in [24]), in particular, the NP-hardness result for restricted DAGs and the polynomial-time algorithms for rooted directed trees and oriented lines.

References

1 Eleni C. Akrida, George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, Paul G. Spirakis, and Viktor Zamaraev. How fast can we reach a target vertex in stochastic temporal graphs? *J. Comput. Syst. Sci.*, 114:65–83, 2020.

2 Eleni C. Akrida, George B. Mertzios, and Paul G. Spirakis. The temporal explorer who returns to the base. In Pinar Heggernes, editor, *Algorithms and Complexity - 11th International Conference, CIAC 2019, Rome, Italy, May 27-29, 2019, Proceedings*, volume 11485 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2019.

3 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *Int. J. Found. Comput. Sci.*, 14(2):267–285, 2003.

4 Manuel Cáceres. Minimum chain cover in almost linear time. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 31:1–31:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

5 Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 359–376. SIAM, 2022.

6 Manuel Cáceres, Brendan Mumey, Edin Husić, Romeo Rizzi, Massimo Cairo, Kristoffer Sahlin, and Alexandru I. Tomescu. Safety in multi-assembly via paths appearing in all path covers of a DAG. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(6):3673–3684, 2022.

7 Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distributed Syst.*, 27(5):387–408, 2012.

8 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.

9 Dibyayan Chakraborty, Antoine Dailly, Florent Foucaud, and Ralf Klasing. Algorithms and complexity for path covers of temporal DAGs. *arXiv preprint arXiv:2403.04589*, 2024.

10 Yangjun Chen and Yibin Chen. On the graph decomposition. In *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, pages 777–784, 2014.

11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

12 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4(8). Springer, 2015.

13 Pradipta Kumar Das, Himansu Sekhar Behera, Prabir Kumar Jena, and Bijaya K. Panigrahi. An intelligent multi-robot path planning in a dynamic environment using improved gravitational search algorithm. *Int. J. Autom. Comput.*, 18(6):1032–1044, 2021.

14 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

15 Jeff Erickson. *Algorithms.* self-published, 2019. URL: http://jeffe.cs.illinois.edu/teaching/algorithms/.

16 D. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

17 Delbert R Fulkerson. Note on Dilworth's decomposition theorem for partially ordered sets. In *Proceedings of the American Mathematical Society*, volume 7(4), pages 701–702, 1956.

18 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57).* North-Holland Publishing Co., NLD, 2004.

19 Ryan B. Hayward. Weakly triangulated graphs. *J. Comb. Theory, Ser. B*, 39(3):200–208, 1985. doi:10.1016/0095-8956(85)90050-4.

20 Ryan B. Hayward, Jeremy P. Spinrad, and R. Sritharan. Improved algorithms for weakly chordal graphs. *ACM Trans. Algorithms*, 3(2):14, 2007. doi:10.1145/1240233.1240237.

21 Petter Holme. Modern temporal network theory: a colloquium. *European Physical Journal B*, 88(9), 2015.

22 H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Transactions on Database Systems*, 15(4):558–598, 1990.

23 Naoyuki Kamiyama and Yasushi Kawase. On packing arborescences in temporal networks. *Inf. Process. Lett.*, 115(2):321–325, 2015.

24 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *J. Comput. Syst. Sci.*, 64(4):820–842, 2002.

25 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: temporally disjoint paths. *Autonomous Agents and Multi Agent Systems*, 37(1):1, 2023.

26 Nina Klobas, George B Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. *Autonomous Agents and Multi-Agent Systems*, 37(1):1, 2023.

27 T. Kloks. *Treewidth, Computations and Approximations.* Springer, 1994.

28 Shimon Kogan and Merav Parter. Faster and unified algorithms for diameter reducing shortcuts and minimum chain covers. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 212–239. SIAM, 2023.

29 Pascal Kunz, Hendrik Molter, and Meirav Zehavi. In which graph structures can we efficiently find temporally disjoint paths and walks? In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 180–188. ijcai.org, 2023.

30 L. Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Mathematics*, 2(3):253–267, 1972.

31 Veli Mäkinen, Alexandru I. Tomescu, Anna Kuosmanen, Topi Paavilainen, Travis Gagie, and Rayan Chikhi. Sparse dynamic programming on dags with small width. *ACM Trans. Algorithms*, 15(2), February 2019.

32 Loris Marchal, Hanna Nagy, Bertrand Simon, and Frédéric Vivien. Parallel scheduling of DAGs under memory constraints. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 204–213. IEEE Computer Society, 2018.

33 Andrea Marino and Ana Silva. Eulerian walks in temporal graphs. *Algorithmica*, 85(3):805–830, 2023.

34 George B. Mertzios, Hendrik Molter, Malte Renken, Paul G. Spirakis, and Philipp Zschoche. The complexity of transitively orienting temporal graphs. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*, volume 202 of *LIPIcs*, pages 75:1–75:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

35 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. In Christos D. Zaroliagis, Grammati E. Pantziou, and Spyros C. Kontogiannis, editors, *Algorithms, Probability, Networks, and Games - Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday*, volume 9295 of *Lecture Notes in Computer Science*, pages 308–343. Springer, 2015.

36 Simeon C. Ntafos and S. Louis Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Transactions on Software Engineering*, SE-5(5):520–529, 1979.

37 Jari Saramäki and Petter Holme, editors. *Temporal Network Theory*. Computational Social Sciences. Springer, Germany, October 2019.

38 Jeremy P. Spinrad and R. Sritharan. Algorithms for weakly triangulated graphs. *Discret. Appl. Math.*, 59(2):181–191, 1995. `doi:10.1016/0166-218X(93)E0161-Q`.

39 Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. Multi-agent pathfinding: Definitions, variants, and benchmarks. In Pavel Surynek and William Yeoh, editors, *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pages 151–159. AAAI Press, 2019.

40 Dawei Sun, Jingkai Chen, Sayan Mitra, and Chuchu Fan. Multi-agent motion planning from signal temporal logic specifications. *IEEE Robotics Autom. Lett.*, 7(2):3451–3458, 2022.

41 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016.

42 Haifeng Xu, Fei Fang, Albert Xin Jiang, Vincent Conitzer, Shaddin Dughmi, and Milind Tambe. Solving zero-sum security games in discretized spatio-temporal domains. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1500–1506. AAAI Press, 2014.

43 Yue Yin and Bo An. Efficient resource allocation for protecting coral reef ecosystems. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 531–537. AAAI Press, 2016.

44 Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R. Jennings. Optimal escape interdiction on transportation networks. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3936–3944. ijcai.org, 2017.

Covering and Partitioning of Split, Chain and Cographs with Isometric Paths

Dibyayan Chakraborty
School of Computing, University of Leeds, UK

Haiko Müller
School of Computing, University of Leeds, UK

Sebastian Ordyniak
School of Computing, University of Leeds, UK

Fahad Panolan
School of Computing, University of Leeds, UK

Mateusz Rychlicki
School of Computing, University of Leeds, UK

Abstract

Given a graph G, an *isometric path cover* of a graph is a set of isometric paths that collectively contain all vertices of G. An isometric path cover $\mathcal{C}$ of a graph G is also an *isometric path partition* if no vertex lies in two paths in $\mathcal{C}$. Given a graph G, and an integer k, the objective of Isometric Path Cover (resp. Isometric Path Partition) is to decide whether G has an isometric path cover (resp. partition) of cardinality k.

In this paper, we show that Isometric Path Partition is NP-complete even on split graphs, *i.e.* graphs whose vertex set can be partitioned into a clique and an independent set. In contrast, we show that both Isometric Path Cover and Isometric Path Partition admit polynomial time algorithms on cographs (graphs with no induced P_4) and chain graphs (bipartite graphs with no induced $2K_2$).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Isometric path partition (cover), chordal graphs, chain graphs, split graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.39

1 Introduction and results

Finding paths in graphs is of fundamental interest to the algorithmic graph theory community. An example is the Path Cover where the objective is to decide whether the vertex set of a graph can be covered by at most k paths. This problem is NP-hard even if $k = 1$ which is equivalent to Hamiltonian Path. Recently, researchers have studied the problem of covering graphs with *isometric paths* i.e. shortest path between its end-vertices. Given a graph G, an *isometric path cover* of a graph is a set of isometric paths that collectively contain all vertices of G. An isometric path cover $\mathcal{C}$ of a graph G is also an *isometric path partition* if no vertex lies in two paths in $\mathcal{C}$. Given a graph G, and an integer k, the objective of Isometric Path Cover (resp. Isometric Path Partition) is to decide whether G has an isometric path cover (resp. partition) of cardinality at most k. Besides algorithmic graph theory, Isometric Path Cover has been studied in different contexts like machine learning [14], combinatorial games [1] etc.

Despite being a natural covering problem, the algorithmic complexity of Isometric Path Cover has thus far remained mostly unexplored. Isometric Path Cover and Isometric Path Partition have been proven to be NP-hard in *chordal* graphs (i.e. graphs without induced cycles of order four or higher) and bipartite graphs of diameter 4 [5, 8]. The

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 39; pp. 39:1–39:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Parameterized complexity of Isometric Path Cover has been studied with respect to various parameters like solution size, neighbourhood diversity, etc [7, 8]. Approximability of Isometric Path Cover has also been studied [4, 5]. It was shown that Isometric Path Cover admits constant factor approximation algorithms on graph classes like chordal graphs and more generally on graphs with bounded treelength and bounded hyperbolicity, outerstring graphs, universally-signable graphs and more generally on (theta, pyrmaid, prism)-free graphs [4, 5]. It was also shown that Isometric Path Cover admits $O(\log n)$-approximation algorithm on general graphs [14]. On the other hand, polynomial time solvability of Isometric Path Cover is only known for special graph classes such as block graphs [12] which is a subclass of chordal graphs. This motivated us to study the computational complexities of Isometric Path Cover and Isometric Path Partition on *split* graphs which is a popular subclass of chordal graphs.

A graph is a *split* graph if the vertex set can be partitioned into a clique C and an independent set I. In this paper, we prove that the Isometric Path Partition problem remains NP-hard on split graphs answering an open question in the literature [5]

▶ **Theorem 1.** *Isometric Path Partition is* NP*-hard on split graphs.*

Our reduction techniques deviate significantly from the known ones which typically reduce the problem of partitioning a graph into induced P_k (for appropriately chosen k, also known as Induced P_k Partition) by adding few vertices of large degree, so that a path in the reduced graph is isometric if and only if it was an induced P_k in the original graph. However, one difficulty in applying this technique is that the complexity of Induced P_k Partition for $k \in \{3, 4\}$ on split graphs is not known. The problems of partitioning split graphs into (non-induced) P_3's can be solved in polynomial time [15] by reducing it to finding a maximum matching in some auxiliary graph.

We reduce from the NP-complete problem 3-XSAT [13] where the input is a CNF formula that has exactly 3 positive literals in every clause and every variable occurs exactly 3 times, and the objective is to decide if there is an assignment that satisfies exactly one literal from every clause. As a biproduct of our proof, we also get the following corollary. Isometric $P_{\geq t}$ Partition denotes the problem of partitioning the vertex set of a graph into isometric paths with at least t vertices.

▶ **Corollary 2.** *For $t \leq 3$, Isometric $P_{\geq t}$ Partition is* NP*-hard on split graphs.*

We note that the computational complexity of Isometric Path Cover on split graphs remains open. Theorem 1 motivates the study of Isometric Path Partition on natural subclasses of split graphs. *Threshold* graphs, the class of split graphs without an induced P_4, is one of the well-studied subclass of split graphs [11]. We show that both Isometric Path Cover and Isometric Path Partition admit polynomial-time algorithms on threshold graphs. In fact, we prove the following more general result. Cographs were introduced in [6] and are exactly the class of graphs with no induced P_4.

▶ **Theorem 3.** *Isometric Path Cover and Isometric Path Partition admit polynomial time algorithms on cographs.*

Our algorithm for Isometric Path Partition on cographs is based on dynamic programming on *cotrees* [6], which also characterise cographs. The class of cographs contains the class of threshold graphs, which are exactly the class of split graphs whose vertices in the independent set can be linearly ordered under inclusion of their open neighbourhoods. We note that designing algorithms for cographs is a popular direction of research in algorithmic graph theory [2, 3, 9, 10].

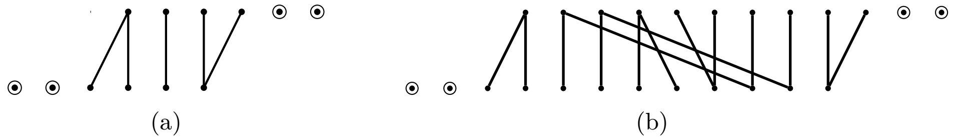


Figure 1 Normal forms of optimal solutions for Isometric Path Partition on chain graphs. Connected components or marked vertices indicate isometric paths in the solution.

Motivated by the success on cographs we then consider the class of *chain* graphs, the bipartite analogue of threshold graphs. A chain graph is a bipartite graph such that the vertices of each color class can be linearly ordered under inclusion of their open neighbourhoods. We show the following result:

▶ **Theorem 4.** *Isometric Path Cover and Isometric Path Partition admit polynomial time algorithms on chain graphs.*

Note that a path in a chain graph or a cograph is an isometric path if and only if it is induced. Due to Theorems 3 and 4, we have the following corollary.

▶ **Corollary 5.** *Induced Path Partition and Induced Path Cover admit polynomial time algorithms on cographs and chain graphs.*

Our algorithm for Isometric Path Partition on chain graphs is based on the observation that any (optimal) solution can be converted into one of three *normal forms*, two of which are illustrated in Figure 1 and where the induced paths can be thought of (roughly) as following the ordering of the vertices according to the neighbourhood inclusion relation. Obtaining the normal forms is also the main challenge behind the algorithm as it turned out to be surprising difficult to find the right order of operations required to transform any solution into one of these normal forms without sacrificing previously made progress. Once the normal form is obtained, however, Isometric Path Partition can be solved via a simple brute-force algorithm that guesses the important positions of a solution in normal form. Finally, a simple reduction from Isometric Path Cover to Isometric Path Partition on chain graphs then allows us to obtain the polynomial-time algorithm for Isometric Path Cover on chain graphs.

Structure of the paper. In Section 2 we introduce some definitions. In Section 3 and Section 4 we prove Theorems 3 and 4 respectively. In Section 5 we prove Theorem 1.

Statements whose full proofs are omitted due to space constraints can be found in the full version.

2 Preliminaries

All graphs considered here are finite, simple, and undirected. That is, a *graph* $G = (V, E)$ consists of a finite set V of *vertices* and a set $E \subseteq V^{(2)}$ of *edges*, where $V^{(2)}$ is the set of 2-element subsets of V. An edge $\{u, v\}$ will also be written as uv. The graph $H = (U, F)$ is a *subgraph* of $G = (V, E)$ if $U \subseteq V$ and $F \subseteq E$. The subgraph H is *induced* if $F = E \cap U^{(2)}$, denoted as $H = G[U]$. The *neighbourhood* $N_G(v)$, or simply $N(v)$ if G is clear from the context, of a vertex $v \in V$ in graph G is $\{u \mid uv \in E\}$. Moreover, for a subset V' of vertices, we denote by $N_G(V')$ the set $\bigcup_{v \in V'} N_G(v)$.

A *path* in G is a subgraph $P = (U, F)$ with $U = \{u_i \mid 0 \leq i \leq l\}$ and $F = \{u_{i-1}u_i \mid 1 \leq i \leq l\}$. We usually refer to P by the sequence $(u_0, u_1, u_2, \ldots, u_l)$. The *length* of this path is l, the number of edges in P.

The *distance* between two vertices u and v of G is the length of a shortest path between u and v. It is denoted by $d_G(u, v)$. A path $P = (u, \ldots, v)$ is *isometric* if $d_G(u, v) = d_P(u, v)$. Every isometric path is an induced path, and induced paths of length at most two are isometric.

3 Algorithms for Chain Graphs

Here, we provide our algorithms for Isometric Path Partition and Isometric Path Cover on chain graphs. A bipartite graph $G = (T, B, E)$ is a *chain graph* if, for every pair of vertices $v, u \in T$ or $v, u \in B$, we have $N(v) \subseteq N(u)$ or $N(v) \supseteq N(u)$, respectively; see also [16]. This implies an ordering $<$ of the vertices in T and the vertices in B such that $u < v$ if $N_G(u) \subseteq N_G(v)$. For convinience and by resolving ties arbitrarily, we will assume that $<$ is a total ordering on T and on B.

The main step of our proofingredient and the main challenge for the algorithms is to show that any solution to Isometric Path Partition on a chain graph can be transformed into an equally-sized solution that follows a certain normal form (Section 3.1). Assuming this normal form, then essentially allows us to solve Isometric Path Partition in polynomial-time via a brute-force approach (Section 3.2). Finally, our polynomial-time algorithm for Isometric Path Cover then uses a simple reduction from Isometric Path Cover to Isometric Path Partition (Section 3.2).

3.1 Normal Form

In this subsection we introduce our normal form for solutions to Isometric Path Partition on chain graphs and show that there is always an optimal solution adhering to this normal form. We start by introducing some important notation.

Let $G = (T, B, E)$ be a chain graph and let $\mathcal{P}$ be a set of isometric paths. A pattern can be thought of as a string over the symbols ∘, Λ, I, V, N, and ˚, where it is possible for symbols to overlap with each other; such as in NΛ or NV, which consists of two overlapping symbols N and Λ respectively N and V. We also allow special symbols to indicate that a symbol is repeated arbitrary many times, e.g., $\overset{*}{\circ}$, $\overset{*}{Λ}$, $\overset{*}{V}$, and $\overset{*}{˚}$ represent arbitrary many (non-crossing) ∘'s, Λ's, V's, and ˚'s, respectively. Moreover, ⌈-NN-⌋ represents arbitrary many N's that are pairwise in pattern ⌈NN⌋, e.g., ⌈NNNNN⌋ is an element of ⌈-NN-⌋.

We say that $\mathcal{P}$ *has pattern* α if the drawing of all paths in $\mathcal{P}$, *i.e.*, the drawing obtained by drawing all vertices in T from right to left in the order $<$ on top of all vertices in B drawn according to $<$ from left to right, looks like α. For instance, if $P \in \mathcal{P}$ is a single path, then P has pattern ∘, Λ, I, V, N, or ˚, if $|V(P) \cap T| = 0$ and $|V(P) \cap B| = 1$, $|V(P) \cap T| = 1$ and $|V(P) \cap B| = 2$, $|V(P) \cap T| = 1$ and $|V(P) \cap B| = 1$, $|V(P) \cap T| = 2$ and $|V(P) \cap B| = 1$, $|V(P) \cap T| = 2$ and $|V(P) \cap B| = 2$, or $|V(P) \cap T| = 1$ and $|V(P) \cap B| = 0$, respectively. Alternatively, we also say that P *is an α-path*, if P has pattern α. We denote by $\mathcal{P}_\alpha$ the set of all α-paths in $\mathcal{P}$ and we denote by $\mathcal{P}_\circ$ the set $\mathcal{P}_{{}_\circ} \cup \mathcal{P}^{\circ}$. Observe that since G is a chain graph every path in $\mathcal{P}$ is either a ∘-path, a Λ-path, an I-path, a V-path, a N-path, or a ˚-path.

We say that a path $P \in \mathcal{P}$ is *before (or to the left)* of a path $P' \in \mathcal{P}$ if $b < b'$ for every $b \in V(P) \cap B$ and $b' \in V(P') \cap B$ and moreover $t > t'$ for every $t \in V(P) \cap T$ and $t' \in V(P') \cap T$. We say that P is *after (or to the right of P')* if P' is before P. We say that P and P' *cross* if P is neither before nor after P'.

The main aim of this section is to show the following theorem.

▶ **Theorem 6.** *Let $G = (T, B, E)$ be a chain graph. G has an optimal isometric path partition that has one of the following patterns:*

(1) $\circ^*\wedge^*\vee^*{}^{\circ*}$, *i.e., any number of* ∘*-paths, followed by any number of* ∧*-paths, followed by any number of* ∨*-paths, followed by any number of* °*-paths,*

(2) $\circ^*\wedge^*\mathsf{I}\vee^*{}^{\circ*}$, *i.e., same as (1) but with exactly one* I*-path in the center,*

(3) $\circ^*\wedge^*\ulcorner\text{-}(\mathsf{N}\wedge^*\vee^*)\text{-}\urcorner\vee^*{}^{\circ*}$, *i.e., same as (2) but with the* I*-path replaced by any number of* ∧*-paths,* ∨*-paths, and* N*-paths such that every pair of those paths has one of the following patterns* ∧.∧, ∨.∨, ∧.∨, N∧, N∨, *or* NN.

Towards showing Theorem 6, we will introduce various intermediate normal forms and we will then show that a solution can be transformed step-by-step into more and more restrictive normal forms. Let $G = (T, B, E)$ be a chain graph and $\mathcal{P}$ be an isometric path partition of G. Most of our normal forms are based on the correct relationships between pairs of paths in $\mathcal{P}$. In particular, we say that two distinct paths P and P' in $\mathcal{P}$ are in *normal position* if either:

(1) If P or P' is a ∘-path, then $\{P, P'\}$ has pattern ∘∘, ∘∧, ∘∨, ∘N, or ∘°. In other words, all ∘-paths are to the left of all other paths in $\mathcal{P}$.

(2) If P or P' is a °-path, then $\{P, P'\}$ has pattern ∘°, ∧°, ∨°, N°, or °°. In other words, all °-paths are to the right of all other paths in $\mathcal{P}$.

(3) If P is a I-path and P' is a ∧-path (∨-path), then $\{P, P'\}$ has pattern ∧I (I∨). In other words, all I-paths are to the right of all ∧-paths and to the left of all ∨-paths in $\mathcal{P}$.

(4) If both P and P' are ∧-paths, then $\{P, P'\}$ has pattern ∧.∧. In other words, no two ∧-paths in $\mathcal{P}$ cross each other.

(5) If both P and P' are ∨-paths, then $\{P, P'\}$ has pattern ∨.∨. In other words, no two ∨-paths in $\mathcal{P}$ cross each other.

(6) If both P and P' are N-paths, then $\{P, P'\}$ has pattern NN.

(7) If P is a N-path and P' is a ∧-path (∨-path), then $\{P, P'\}$ has pattern ∧.N or N∧ (N.∨ or N∨).

(8) If P is a ∧-path and P' is a ∨-path, then $\{P, P'\}$ has pattern ∧.∨.

We say that $\mathcal{P}$ satisfies (i), $i \in [8]$, if condition (i) holds for every two distinct paths P and P' in $\mathcal{P}$. We are now ready to define our normal forms.

▶ **Definition 7.** *Let $\mathcal{P}$ be an isometric path partition of a chain graph G.*

- *We say that $\mathcal{P}$ is* NI-minimal *if either $\mathcal{P}_{\mathsf{N}} = \emptyset$ and $|\mathcal{P}_{\mathsf{I}}| \leq 1$ or $\mathcal{P}_{\mathsf{N}} \neq \emptyset$ and $\mathcal{P}_{\mathsf{I}} = \emptyset$.*
- *We say that $\mathcal{P}$ is in* S-normal form *if $\mathcal{P}$ is NI-minimal and $\mathcal{P}$ satisfies (1) and (2).*
- *We say that $\mathcal{P}$ is in* I-normal form, A-normal form, V-normal form, *or* N-normal form *if $\mathcal{P}$ satisfies (3), (4), (5), or (6), respectively.*
- *We say that $\mathcal{P}$ is in* mixed normal form *if $\mathcal{P}$ is in S-normal form and additionally $\mathcal{P}$ satisfies (7) and (8).*
- *We say that $\mathcal{P}$ is in* pair-normal form *if $\mathcal{P}$ is in mixed normal form, in I-normal form, in A-normal form, in V-normal form, and in N-normal form.*

All normal forms given above are purely defined in terms of restrictions on the relationships between pairs of paths in an isometric path partition. However, to obtain a normal form for the pattern given in Theorem 6 (3), we need to additionally put restrictions on the patterns allowed for triples of isometric paths. We say that $\mathcal{P}$ is in *AN-normal form* if there are no distinct $P_A \in \mathcal{P}_{\wedge}$, $P_N, P'_N \in \mathcal{P}_{\mathsf{N}}$ such that $\{P_N, P_A\}$ has pattern N∧ and $\{P'_N, P_A\}$ has pattern ∧.N. In other words, for every $P_A \in \mathcal{P}_{\wedge}$ it holds that either $\{P_A, P_N\}$ has pattern ∧.N for every $P_N \in \mathcal{P}_{\mathsf{N}}$ or $\{P_A, P_N\}$ has pattern N∧ for every $P_N \in \mathcal{P}_{\mathsf{N}}$. Similarily, we say that $\mathcal{P}$ is in

NV-normal form if and there are no distinct $P_V \in \mathcal{P}_{\mathsf{V}}$, $P_N, P'_N \in \mathcal{P}_{\mathsf{N}}$ such that $\{P_N, P_V\}$ has pattern N̸V and $\{P'_N, P_V\}$ has pattern NV. Finally, we say that $\mathcal{P}$ is in *normal form* if $\mathcal{P}$ is in pair-normal form, AN-normal form, and in NV-normal form.

Observe that any isometric path partition in normal form has one of the three patterns given in Theorem 6. To show Theorem 6, it therefore suffices to show the following.

▶ **Theorem 8** (⋆). *Let $G = (T, B, E)$ be a chain graph. There is an optimal isometric path partition of G in normal form.*

The main challenge for proving Theorem 8, whose proof turned out to be surprisingly difficult, is to define the right operations and to apply them in the right order to obtain more and more restricted normal forms while maintaining the progress already made. In particular, after defining the required operations, we will show how to obtain normal form in the following order: S-normal form, mixed normal form, I-normal form, A-normal form, N-normal form, V-normal form, AN-normal form, and finally NV-normal form.

We say that a pattern α implies a pattern β (denoted by $\alpha \to \beta$) if for any chain graph that contains a set $\mathcal{P}$ of pairwise disjoint isometric paths with pattern α there is a set $\mathcal{P}'$ of pairwise disjoint isometric paths with pattern β such that $V(\mathcal{P}) = V(\mathcal{P}')$. (Here $V(\mathcal{P})$ is the set of vertices covered by the paths in $\mathcal{P}$.) For a set Δ of patterns we write $\Delta \to \beta$ if $\delta \to \beta$ for every $\delta \in \Delta$.

For a pattern α, we denote by $\alpha^{\curvearrowright}$ the pattern obtained from α after rotating α by 180° degrees around the center of the pattern. For instance, Λ̸N$^{\curvearrowright}$ = N̸V, ΛN$^{\curvearrowright}$ = NV, ΛV$^{\curvearrowright}$ = ΛV, ΛI$^{\curvearrowright}$ = IV, and $\alpha = \alpha^{\curvearrowright\curvearrowright}$. Moreover, for a set of patterns Δ, we define $\Delta^{\curvearrowright}$ as $\{\, \delta^{\curvearrowright} \mid \delta \in \Delta \,\}$. The following observation is very useful to reduce the number of cases that have to be considered in the proofs and follows easily from the symmetry of chain graphs w.r.t. rotation by 180° degree around the center.

▶ **Observation 9.** *Let α and β be patterns. Then $\alpha \to \beta$ implies $\alpha^{\curvearrowright} \to \beta^{\curvearrowright}$.*

We say that a pattern α is *valid* if there is a set of pairwise disjoint isometric paths in some chain graph with pattern α. For a pattern α, we denote by $[\alpha]$, the set of all valid patterns that can be obtained from α after reordering the endpoints of the lines on the top and/or on the bottom of the pattern α in an arbitrary manner. For instance, [N∘] = {N∘, N̥, ∘N} and [ΛI] = {Λ\, /X, X, X, X\, /Λ}. Note that [N] does not contain any pattern where the lines of the N cross each other because those patterns are not valid.

The following auxiliary lemma provides the most important production rules on patterns that we will use to transform an arbitrary solution into a solution in normal form.

▶ **Lemma 10** (⋆). *The following holds:*

(a) [∘∘] → ∘∘ *and* [°°] → °°,

(b) [∘°] → ∘°,

(c) [∘I] → ∘I *and* [I°] → I°,

(d) [∘Λ] → ∘Λ *and* [V°] → V°,

(e) [Λ°] ∪ [II] → Λ° *and* [∘V] ∪ [II] → ∘V,

(f) [ΛI] ∪ [∘N] \ {∘N} → ΛI *and* [IV] ∪ [N°] \ {N°} → IV,

(g) [ΛΛ] → ΛΛ *and* [VV] → VV,

(h) [ΛV] ∪ [NI] → ΛV,

(i) [ΛN] \ {ΛN} → Λ̸N *and* [NV] \ {NV} → N̸V,

Sketch. (a) and (b) follow immediately because [∘∘] = {∘∘}, [⁚] = {⁚}, and [∘˚] = {∘˚}.

Let $G = (T, B, E)$ be a chain graph and let P and P' be two isometric paths in G. Let $(V(P) \cup V(P')) \cap T = \{t_1, \ldots, t_c\}$ and $(V(P) \cup V(P')) \cap B = \{b_1, \ldots, b_d\}$ such that $t_1 > \cdots > t_c$ and $b_1 < \cdots < b_d$.

Towards showing (e), assume that $\{P, P'\}$ has pattern α for some $\alpha \in$ [Λ˚] ∪ [‖]. Then, $c = 2$, $d = 2$ and b_1 must have degree at least 1 in G. Therefore, because $N_G(t_2) \subseteq N_G(t_1)$, it follows that $b_1 t_1 \in E(G)$, which in turn implies that $b_2 t_1 \in E(G)$ since $N_G(b_1) \subseteq N_G(b_2)$. Therefore, $\{(b_1, t_1, b_2), (t_2)\}$ is a pair of isometric paths with pattern Λ˚, as required. Because of Observation 9, we also obtain [∘V] ∪ [‖] → ∘V. The remaining cases can be shown in an analogous manner. ◀

The remainder of this subsection is about showing that there are optimal solutions that have more and more restrictive normal forms. As an illustrative example we show next how to obtain a mixed normal form from a solution in S-normal form. We start by stating the lemma required to obtain a S-normal form.

▶ **Lemma 11** (⋆). *Let $G = (T, B, E)$ be a chain graph. There is an optimal isometric path partition $\mathcal{P}$ which is in S-normal form.*

Building upon S-normal form, the following lemma now shows that we can achieve mixed normal form. The proof of the lemma is based on an exhaustive application of certain production rules from Lemma 10 combined with a potential function approach for showing that this process terminates.

▶ **Lemma 12** (⋆). *Let $G = (T, B, E)$ be a chain graph. There is an optimal isometric path partition of G in mixed normal form.*

Sketch. Let $G = (T, B, E)$ be a chain graph and let $\mathcal{P}$ be an optimal isometric path partition of G. Because of Lemma 11, we can assume that $\mathcal{P}$ is in S-normal form.

Let $\mathcal{P}'$ be the isometric path partition of G obtained from $\mathcal{P}$ after exhaustively applying the following transformation rules from Lemma 10:

(1) [ΛV] → ΛV,
(2) [ΛN] \ {ΛN} → NΛN, and
(3) [NV] \ {NV} → NVN,

Observe that if $\mathcal{P}'$ exists, then it trivially satisfies all the claims made in the statement of the lemma. It therefore suffices to show the existence of $\mathcal{P}'$ or in other words that the above rules can only be applied finitely often to $\mathcal{P}$.

Towards showing this we will define a potential function Φ that assigns a two dimensional vector of natural numbers to every isometric path partition of G. For the definition of Φ, we need the following additional notation. For two vertices u and v of G such that either $u, v \in B$ or $u, v \in T$, we denote by $[u, v]_G$ the set $\{u, v\} \cup \{\, w \mid u < w < v \,\}$ of vertices of G.

For a path $P = (p_1, p_2, p_3, p_4) \in \mathcal{P}_{\mathsf{N}}$, we denote by $W_G(P)$ the integer $|[p_1, p_3]_G| + |[p_2, p_4]_G|$. For a path $P = (p_1, p_2, p_3) \in \mathcal{P}_{\Lambda}$, we denote by $L_G(P)$ the integer $|[f_b, p_1]_G| + |[f_b, p_3]_G| + |[f_t, p_2]_G|$, where f_b is the smallest vertex in B and f_t is the largest vertex in T w.r.t. $<$.

For a path $P = (p_1, p_2, p_3) \in \mathcal{P}_{\mathsf{V}}$, we denote by $R_G(P)$ the integer $|[l_t, p_1]_G| + |[l_b, p_2]_G| + |[l_t, p_3]_G|$, where l_t is the smallest vertex in T and l_b is the largest vertex in B w.r.t. $<$.

For a isometric path partition $\mathcal{P}^\star$ of G, we define the first and second component of $\Phi(\mathcal{P}^\star)$ as follows.

$$\Phi(\mathcal{P}^\star)[1] = \sum_{P \in \mathcal{P}^\star_{\mathsf{N}}} W_G(P) \qquad \Phi(\mathcal{P}^\star)[2] = -\Big(\sum_{P \in \mathcal{P}^\star_{\Lambda}} L_G(P)\Big) - \Big(\sum_{P \in \mathcal{P}^\star_{\mathsf{V}}} R_G(P)\Big)$$

Let $\mathcal{P}^1$ and $\mathcal{P}^2$ be two isometric path partitions of G such that $\mathcal{P}^2$ is obtained from $\mathcal{P}^1$ by applying exactly one of the operations given in (1)–(3). We claim that $\Phi(\mathcal{P}^2) > \Phi(\mathcal{P}^1)$, where $>$ is the lexicographical ordering among two dimensional integer vectors. Because Φ is finite, *i.e.*, $(0, -|B||\mathcal{P}^\star_{\mathsf{\Lambda}} \cup \mathcal{P}^\star_{\mathsf{V}}|) \leq \Phi(\mathcal{P}^\star) \leq (|B||\mathcal{P}^\star_{\mathsf{N}}|, 0)$, this then shows that $\mathcal{P}'$ is well defined.

It suffices to show the claim for the cases that $\mathcal{P}^2$ is obtained from $\mathcal{P}^1$ using one of the operations (1)–(3). We show the case for operation (1) as an illustration and leave operations (2) and (3) for the full version. If $\mathcal{P}^2$ is obtained from $\mathcal{P}^1$ by applying operation (1) to two paths $\mathsf{P}_A \in \mathcal{P}^1_{\mathsf{\Lambda}}$ and $P_V \in \mathcal{P}^1_{\mathsf{V}}$, then $\Phi(\mathcal{P}^2)[1] = \Phi(\mathcal{P}^1)[1]$ and $\Phi(\mathcal{P}^2)[2] > \Phi(\mathcal{P}^1)[2]$ because no path in $\mathcal{P}^1$ other than P_A and P_V is changed and moreover because $\mathsf{\Lambda V}$ is the unique pattern that maximizes $\Phi(\{P, P'\})$ for any two paths $P \in \mathcal{P}^1_{\mathsf{\Lambda}}$ and $P' \in \mathcal{P}^1_{\mathsf{V}}$. ◀

Surprisingly, even after reaching mixed normal form, we are still rather far away from our final normal form. In particular, we will go through the following normal forms (in that order): I-normal form, A-normal form, N-normal form, V-normal form, AN-normal form, and finally NV-normal form ($\star$).

3.2 The Algorithms

Having developed our normal forms, we are now ready to show Theorem 4. We start by showing the result for Isometric Path Partition.

▶ **Lemma 13** ($\star$)**.** *Isometric Path Partition admits a polynomial-time algorithm on chain graphs.*

Sketch. Let $G = (T, B, E)$ be a chain graph. Theorem 6 implies that there is an optimal isometric path partition of G having pattern $\overset{*}{\circ}\overset{*}{\mathsf{\Lambda}}\overset{*}{\mathsf{V}}\overset{*}{\circ}$, $\overset{*}{\circ}\overset{*}{\mathsf{\Lambda}}\mathsf{I}\overset{*}{\mathsf{V}}\overset{*}{\circ}$, or $\overset{*}{\circ}\overset{*}{\mathsf{\Lambda}}\mathsf{N}\overset{*}{\mathsf{\Lambda}}\overset{*}{\mathsf{V}}\mathsf{N}\overset{*}{\mathsf{V}}\overset{*}{\circ}$. It therefore suffices to show that we can compute an optimal solution having any of these patterns in polynomial-time. It suffices to provide the proof for the most involved of those patterns, *i.e.*, the pattern $\overset{*}{\circ}\overset{*}{\mathsf{\Lambda}}\mathsf{N}\overset{*}{\mathsf{\Lambda}}\overset{*}{\mathsf{V}}\mathsf{N}\overset{*}{\mathsf{V}}\overset{*}{\circ}$, as the proof of the other two patterns is analogous. Hence, let $\mathcal{P}$ be an isometric path partition of G having pattern $\overset{*}{\circ}\overset{*}{\mathsf{\Lambda}}\mathsf{N}\overset{*}{\mathsf{\Lambda}}\overset{*}{\mathsf{V}}\mathsf{N}\overset{*}{\mathsf{V}}\overset{*}{\circ}$, then $\mathcal{P}$ can be defined by the following 6 numbers:

- the number p_1 of $\circ$-paths,
- the number p_2 of $\mathsf{\Lambda}$-paths that are to the left of any N-path,
- the number p_3 of N-paths,
- the number p_4 of $\mathsf{\Lambda}$-paths that are inside all N-paths,
- the number p_5 of V-paths that are inside all N-paths, and
- the number p_6 of V-paths that are to the right of any N-path.

We say that a tuple $U = (p_1, \ldots, p_6)$ of those six numbers is *valid* if there is an isometric path partition of G with pattern $\overset{*}{\circ}\overset{*}{\mathsf{\Lambda}}\mathsf{N}\overset{*}{\mathsf{\Lambda}}\overset{*}{\mathsf{V}}\mathsf{N}\overset{*}{\mathsf{V}}\overset{*}{\circ}$ with the number of paths as given by U. It is now straightforward to show that the validity of any tuple U can be verified in time $\mathcal{O}(|V(G)|)$ by checking the existence and non-existence of certain edges in G.

Putting everything together, we can solve Isometric Path Partition in time $\mathcal{O}(|V(G)|^7)$ by enumerating all of the at most $|V(G)|^6$ tuples U in time $\mathcal{O}(|V(G)|^6)$, testing their validity in time $\mathcal{O}(|V(G)|)$, and then returning the solution that corresponds to a valid tuple U minimizing $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + (|T| - p_2 - 2p_3 - p_4 - 2p_5 - 2p_6)$. ◀

We will now show that Isometric Path Cover on chain graphs can be reduced to Isometric Path Partition on chain graphs with the help of the following lemma.

▶ **Lemma 14** ($\star$)**.** *Let $G = (T, B, E)$ be a chain graph, $t \in T$ be a vertex such that $t > t'$ for all $t' \in T \setminus \{t\}$, and $b \in B$ be a vertex such that $b' < b$ for all $b' \in B \setminus \{b\}$. Then, there is an optimal isometric path cover $\mathcal{P}$ of G such that all vertices in $V(G) \setminus \{t, b\}$ appear in exactly one path in $\mathcal{P}$.*

Now to solve Isometric Path Cover on a chain graph $G = (T, B, E)$ we do the following. Fix two vertices $t \in T$ and $b \in B$ such that $t > t'$ for all $t' \in T \setminus \{t\}$, and $b' < b$ for all $b' \in B \setminus \{b\}$. From Lemma 14, we know that there is an optimal isometric path cover $\mathcal{C}$ such that no vertex in $V(G) \setminus \{t, b\}$ belongs to more than one path. Now we guess the numbers n_t and n_b such that the number of paths in $\mathcal{C}$ that contain t and b are n_t and n_b, respectively, and create a new graph G' by adding $n_t - 1$ copies $a_1, \ldots, a_{n_t-1}$ of t, and $n_b - 1$ copies $c_1, \ldots, c_{n_b-1}$ of b. Moreover, each vertex in $\{a_1, \ldots, a_{n_t-1}\}$ is adjacent to all the neighbours of t and each vertex in $\{c_1, \ldots, c_{n_b-1}\}$ is adjacent to all the neighbours of b. Then, clearly there is an isometric path partition $\mathcal{P}$ in G' of size $|\mathcal{C}|$. Also, any isometric path partition $\mathcal{Q}$ in G' can be converted into an isometric path cover in G of size $|\mathcal{Q}|$. Hence, we solve Isometric Path Partition on G' and output accordingly.

4 Algorithms on Cographs

Here we design polynomial time algorithms for Isometric Path Partition and Isometric Path Cover on cographs. Complement-reducible graphs (or *cographs* for short) were introduced in [6]. To define the class we use the operations *union* $\oplus$ and *join* $\otimes$ for graphs $G = (V, E)$ and $H = (U, F)$ with $V \cap U = \emptyset$, i.e., $G \oplus H = (V \cup U, E \cup F)$ and $G \otimes H = (V \cup U, E \cup F \cup \{vu \mid v \in V, u \in U\})$.

▶ **Definition 15.** *The cographs can be defined recursively:*

- *K_1 is a cograph.*
- *If G and H are cographs then so are $G \oplus H$ and $G \otimes H$.*
- *There are no other cographs.*

A *cotree* T of a graph $G = (V, E)$ can be defined as a rooted binary tree where the leaves are the vertices in V and the inner nodes are marked with $\oplus$ and $\otimes$ such that two vertices $u, v \in V$ are adjacent if and only if their least common ancestor in T is marked by $\otimes$. Then we say that G has a cotree.

▶ **Theorem 16** ([6])**.** *For every graph G the following conditions are equivalent.*

1. *G is a cograph.*
2. *G does not contain P_4 as induced subgraph.*
3. *G has a cotree.*

The following observations follow from the definition of cotrees.

▶ **Observation 17.** *Let T be a cotree of a cograph G. Let (v_1, v_2, v_3) be an induced path in G. Then, there is a node t labelled $\otimes$ in the tree such that the least common ancestor of v_1 and v_2, as well as v_2 and v_3 is t.*

▶ **Observation 18.** *Let T be a cotree of a cograph G. For a node t in T, let T_t be the subtree of T rooted at t. For a node t, if G_t is the cograph that has the cotree T_t, then G_t is the subgraph of G induced on $V(G_t)$. This implies that for any $u, v \in V(G_t)$, if $(u, v) \notin E(G_t)$, then $(u, v) \notin E(G)$.*

Now, we discuss an algorithm for Isometric Path Partition on cographs. For a cograph $G = (V, E)$ let $Q(G)$ denote the set of quadruples $(\ddot{p}, p_1, p_2, p_3)$ such that V has a partition $\mathcal{S}$ into $\ddot{p} + p_1 + p_2 + p_3$ subsets $S \subseteq V$ such that

- for $i \in \{1, 2, 3\}$, exactly p_i sets $S \in \mathcal{S}$ induce a graph $G[S]$ isomorphic to P_i, which is a path on i vertices, and
- for the remaining $\ddot{p}$ subsets $S \in \mathcal{S}$ the subgraph $G[S]$ is isomorphic to $2P_1$, which is an edgeless graph on two vertices.

Theorem 16 implies that each partition of a cograph into isometric paths consists of P_1s, P_2s and P_3s only.

▶ **Lemma 19.** *For a cograph G on n vertices we can compute $Q(G)$ in time $\mathcal{O}(n^{10})$.*

Proof. A cotree T of G can be computed in linear time [6]. We compute $Q(G)$ as follows:

$$\begin{aligned}
Q(K_1) &= \{(0, 1, 0, 0)\} \\
Q(G' \oplus H) &= \{(\ddot{p} + \ddot{q} + r, p_1 + q_1 - 2r, p_2 + q_2, p_3 + q_3) \mid 0 \le r \le \min\{p_1, q_1\}, \\
&\qquad (\ddot{p}, p_1, p_2, p_3) \in Q(G'), (\ddot{q}, q_1, q_2, q_3) \in Q(H)\} \\
Q(G' \otimes H) &= \{\big((\ddot{p} - k) + (\ddot{q} - l), (p_1 - i - l) + (q_1 - i - k), p_2 + q_2 + i, \\
&\qquad (p_3 + k) + (q_3 + l)\big) \mid 0 \le i, 0 \le k \le \ddot{p}, 0 \le i + k \le q_1, 0 \le l \le \ddot{q}, \\
&\qquad 0 \le i + l \le p_1, (\ddot{p}, p_1, p_2, p_3) \in Q(G'), (\ddot{q}, q_1, q_2, q_3) \in Q(H)\}
\end{aligned}$$

The equation for $\oplus$ holds because $G' \oplus H$ does not contain any edges between G' and H. Every path in $G' \oplus H$ is either a path in G' or in H. We can count a P_1 in G' and a P_1 in H as one $2P_1$ in the first coordinate or as two P_1 in the second. For $\otimes$ we can create i paths P_2 from a P_1 in G and a P_1 in H, k paths P_3 from one $2P_1$ in G and one P_1 in H, and l paths P_3 from one P_1 in G and one $2P_1$ in H. Using induction on the nodes of the cotree and the Observations 17 and 18, we prove that the above recursive formulae are correct.

Note that for any cograph G', the number of quadruples in $Q(G')$ is at most n^3, because, by knowing specific values for $\ddot{p}$, p_2 and p_3, we can determine p_1 through the equation $2\ddot{p} + p_1 + 2p_2 + 3p_3 = |V(G')|$. The cotree T has $2n - 1$ nodes. Now we estimate the time to compute $Q(G' \otimes H)$, which is asymptotically larger than that of $Q(G' \oplus H)$. For each $(\ddot{p}, p_1, p_2, p_3) \in Q(G')$ and $(\ddot{q}, q_1, q_2, q_3) \in Q(H)$, we run over three values k, l, i, each of them is upper bounded by n, and constructs tuples in $Q(G' \otimes H)$. The number of choices for k, l, i is at most n^3. Thus, for each each $(\ddot{p}, p_1, p_2, p_3) \in Q(G')$ and $(\ddot{q}, q_1, q_2, q_3) \in Q(H)$, we will be taking $O(n^3)$ time. Since the cardinalities of $Q(G')$ and $Q(H)$ are upper bounded by n^3 each, the total running time to compute $Q(G' \otimes H)$ is $O(n^9)$. Since the cotree has $2n - 1$ nodes, the total running time of our algorithm is $O(n^{10})$. ◀

▶ **Lemma 20.** *Isometric Path Partition can be solved in $\mathcal{O}(n^{10})$ time on cographs.*

Proof. The recurrence for Q leads to a dynamic programming algorithm computing the isometric path partition number of a cograph G, which is $\min\{p_1 + p_2 + p_3 \mid (0, p_1, p_2, p_3) \in Q(G)\}$. ◀

Next, we explain how to use the values in $Q(G)$ to solve Isometric Path Cover on connected cographs.

▶ **Lemma 21.** *Let G be a connected cograph. Let $Q(G)$ be the set defined before in this section. Then, the cardinality of an optimal isometric path cover of G is*

$$\min\{\ddot{p} + p_1 + p_2 + p_3 \mid (\ddot{p}, p_1, p_2, p_3) \in Q(G)\}.$$

Proof. Let $(\ddot{p}, p_1, p_2, p_3) \in Q(G)$. We claim that there is an isometric path cover of G of size $\ddot{p} + p_1 + p_2 + p_3$. Since $(\ddot{p}, p_1, p_2, p_3) \in Q(G)$, we know that $V(G)$ has a partition into $\ddot{p} + p_1 + p_2 + p_3$ subsets $S \subseteq V(G)$ of which,

(a) for $i \in \{1, 2, 3\}$, exactly p_i induced $G[S]$ is isomorphic to P_i, and

(b) for the remaining $\ddot{p}$ subsets we have $G[S]$ isomorphic to $2P_1$.

Since G is a connected cograph, in Case (b), there is an induced path of length 2 with the vertices in S being its end vertices. So these paths along with the paths in Case (a) forms an isometric path cover of G of size $\ddot{p} + p_1 + p_2 + p_3$.

Now we prove the reverse direction. Without loss of generality we assume that there is an optimal isometric path cover $\mathcal{P}$ of G where for each path $P \in \mathcal{P}$, its end vertices appear only in one path which is P. Let $\ell = |\mathcal{P}|$. Now we claim that there is a tuple $(\ddot{p}, p_1, p_2, p_3) \in Q(G)$ such that $\ddot{p} + p_1 + p_2 + p_3 = \ell$. Let $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$, where $\mathcal{P}_j$ contains all paths on j vertices from $\mathcal{P}$. Let $\mathcal{Q}_1 = \{V(P) \colon P \in \mathcal{P}_1\}$ and $\mathcal{Q}_2 = \{V(P) \colon P \in \mathcal{P}_2\}$. Now we construct $\ddot{\mathcal{Q}}$ and $\mathcal{Q}_3$. Initially, we set $\ddot{\mathcal{Q}} := \emptyset$ and $\mathcal{Q}_3 := \emptyset$. Consider a vertex z such that z is an intermediate vertex in a path in $\mathcal{P}_3$. Suppose z is present in n_z paths. Because of our assumption of $\mathcal{P}$, we know that z is the intermediate vertex in all those n_v paths in $\mathcal{P}_3$. Let $R_1, R_2, \ldots, R_{n_z}$ be these paths. Now let $S_{z,i}$ be the set containing the end vertices of R_i for all $i \in [n_z - 1]$ and $S_{z,n_z} = V(R_{n_z})$. Now we set $\ddot{\mathcal{Q}} := \ddot{\mathcal{Q}} \cup \{S_{z,i} : i \in [n_z - 1]\}$ and $\mathcal{Q}_3 := \mathcal{Q}_3 \cup \{S_{i,n_z}\}$. We do this procedure for each z such that it is an intermediate vertex of a path in $\mathcal{P}_3$. Let $\ddot{p} = |\ddot{\mathcal{Q}}|$ and $p_i = |\mathcal{Q}_i|$ for all $i \in \{1, 2, 3\}$. It is easy to see that $\ddot{p} + p_1 + p_2 + p_3 = \ell$ and the above construction of $\ddot{\mathcal{Q}}$ and $\mathcal{Q}_3$ implies that $(\ddot{p}, p_1, p_2, p_3) \in Q(G)$. ◀

Lemma 21 implies that Isometric Path Cover can be solved in time $O(n^{10})$ on cographs.

5 Hardness on Split Graphs

Here, we show that Isometric Path Partition is already NP-hard on split graphs.

▶ **Theorem 22** (⋆). *Isometric Path Partition is* NP*-hard on split graphs.*

Sketch. We provide a polynomial-time reduction from the NP-complete 3-XSAT problem [13, Lemma 5], where given a propositional formula Φ in CNF such that every clause of Φ has exactly 3 positive literals and every variable occurs exactly in 3 clauses of Φ, the task is to decide whether there is an assignment of the variables of Φ that satisfies exactly one literal from every clause. Let Φ be the given instance of 3-XSAT. We will construct a split graph $G = (T, B, E)$ where T is the independent set and B is the clique such that Φ has an assignment satisfying exactly one literal from every clause if and only if G has isometric path partition $\mathcal{P}$ of size at most $32n^2 - 4n$, where n is a number of clauses (and also the number of variables) of Φ. We shall frequently refer to B as "bottom" and T as "top".

Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of clauses of Φ and let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be the set of variables of Φ. G is constructed from three types of gadgets defined as follows and illustrated in Figure 2. For every variable x_a, G contains the gadget $G_{\mathcal{X}}(x_a)$ defined as follows. Let C_i, C_j and C_k with $1 \leq i < j < k \leq n$ be the 3 clauses that contain x_a. For every $b \in \{i, j, k\}$, $G_{\mathcal{X}}(x_a)$ has the vertices $x_{a,b}$, $\overline{x_{a,b}}$, $x^{\otimes}_{a,b}$ and $x^{\odot}_{a,b}$ with $x_{a,b}, \overline{x_{a,b}} \in B$ and $x^{\otimes}_{a,b}, x^{\odot}_{a,b} \in T$ and the edges $x_{a,b}x^{\otimes}_{a,b}$ and $\overline{x_{a,b}}x^{\odot}_{a,b}$. Additionally, $G_{\mathcal{X}}(x_a)$ contains the edges $\overline{x_{a,i}}x^{\otimes}_{a,j}$, $\overline{x_{a,j}}x^{\otimes}_{a,k}$, and $\overline{x_{a,k}}x^{\otimes}_{a,i}$. Intuitively, the gadget is used to ensure that all occurrences of the variable x_a are assigned in the same manner, which is achieved by forcing that any solution for G can cover all vertices in the gadget in only two manners, which are illustrated in Figure 2.

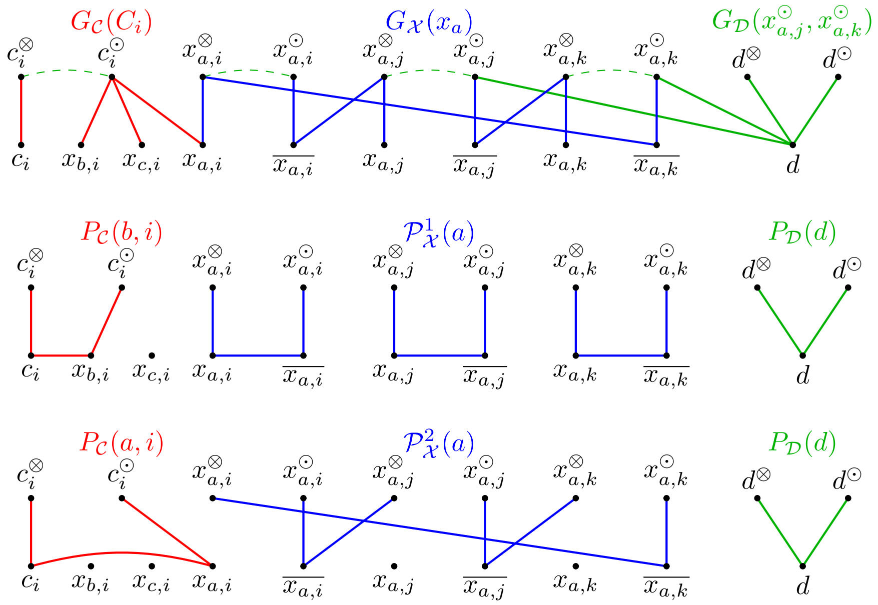


Figure 2 Illustration of the gadgets $G_{\mathcal{C}}(C_i)$, $G_{\mathcal{X}}(x_a)$, and $G_{\mathcal{D}}(x^{\odot}_{a,j}, x^{\odot}_{a,k})$ used in the proof of Theorem 1. The colors of the edges indicate, which gadget they belong to, *i.e.*, red, blue, and green edges are part of $G_{\mathcal{C}}(C_i)$, $G_{\mathcal{X}}(x_a)$, and $G_{\mathcal{D}}(x^{\odot}_{a,j}, x^{\odot}_{a,k})$, respectively. The top figure provides the edges that are part of each gadget (without the edges that are part of the clique on B). The dashed green edges indicate pairs of twin vertices, *i.e.*, the only pairs on the top that can be used as endpoints of isometric paths of length 3. The center and the bottom figure together provide the two possible configurations for how the vertices of the gadgets can be cover in an isometric path partition, which is also descripted in (1), (3), and (4). The center figure illustrates the case that the variable x_a is set to 0 and does not satisfy the clause C_i and the bottom figure illustrates the opposite case.

For every clause $C_i \in \mathcal{C}$, G contains the gadget $G_{\mathcal{C}}(C_i)$. The gadget $G_{\mathcal{C}}(C_i)$ consists of 1 new bottom vertex c_i, 2 new top vertices $c_i^{\otimes}$ and $c_i^{\odot}$, and the edges $c_i c_i^{\otimes}$ and $x_{a,i} c_i^{\odot}$ for every $x_a \in C_i$. This gadget $G_{\mathcal{C}}(C_i)$ will ensure that every clause in Φ is satisfied by exactly one literal. Given two distinct top vertices u and v, the last gadget $G_{\mathcal{D}}(u,v)$, which we call the *destroyer gadget*, has 1 bottom vertex $d_{u,v}$, 2 top vertices $d^{\otimes}_{u,v}$ and $d^{\odot}_{u,v}$ and the edges $d_{u,v} d^{\otimes}_{u,v}$, $d_{u,v} d^{\odot}_{u,v}$, $d_{u,v} u$ and $d_{u,v} v$. The purpose of the destroyer gadget is to ensure that the two top vertices u and v can not occur together in a path of length 3.

We are now ready to define the graph G. Let $G' = \bigcup_{x \in \mathcal{X}} G_{\mathcal{X}}(x) \cup \bigcup_{C \in \mathcal{C}} G_{\mathcal{C}}(C)$. Note that $|T(G')| = 6n + 2n = 8n$. We say that two top vertices v and v' from $T(G')$ are twins, if there exists $u \in B(G')$ such that $v = u^{\otimes}$ and $v' = u^{\odot}$. Then, G is the union of G' and an instance of the destroyer gadget $G_{\mathcal{D}}(u,v)$ for every two distinct top vertices u and v of $T(G')$ that are not twins. Let $B_{\mathcal{D}}$ be the set of all bottom vertices from all of the destroyer gadgets. Note that $|T(G)| = |T(G')|(|T(G')| - 2) + |T(G')| = 64n^2 - 8n$.

Consider an isometric path partition $\mathcal{P}$ of G with $|\mathcal{P}| = \frac{|T(G)|}{2} = 32n^2 - 4n$. The correctness of the reduction can now be shown rather straightforwardly from the following properties $(\star)$:

(1) For every $d \in B_{\mathcal{D}}$, the path $P_{\mathcal{D}}(d) = (d^{\otimes}, d, d^{\odot})$ is included in $\mathcal{P}$.

(2) If a path $P \in \mathcal{P}$ is of length 3, then its endpoints are twins.

(3) For every $i \in [n]$ there exists $a \in [n]$ such that $P_{\mathcal{C}}(a,i) = (c_i^{\otimes}, c_i, x_{a,i}, c^{\odot})$ is in $\mathcal{P}$.

(4) For any $a, i, j, k \in [n]$ with x_a belonging to $C_i \cap C_j \cap C_k$ and $i < j < k$, precisely one of the following scenarios occurs:

(a) The set of paths $\mathcal{P}^1_{\mathcal{X}}(a) = \{\, (x^{\otimes}_{a,\ell}, x_{a,\ell}, \overline{x_{a,\ell}}, x^{\odot}_{a,\ell}) \mid \ell \in \{i, j, k\} \,\}$ is a subset of $\mathcal{P}$, or

(b) $\mathcal{P}^2_{\mathcal{X}}(a) = \{\, (x^{\otimes}_{a,i}, \overline{x_{a,k}}, x^{\odot}_{a,k}), (x^{\otimes}_{a,j}, \overline{x_{a,i}}, x^{\odot}_{a,i}), (x^{\otimes}_{a,k}, \overline{x_{a,j}}, x^{\odot}_{a,j}) \,\}$ is a subset of $\mathcal{P}$.

(5) For every $a, i \in [n]$ and $P \in \mathcal{P}$, such that $x_a \in C_i$ and $x_{a,i} \in V(P)$, $P = P_{\mathcal{C}}(a, i)$ or $P \in \mathcal{P}^1_{\mathcal{X}}(a)$. ◀

6 Conclusion

In this paper we proved that Isometric Path Partition remains NP-hard on split graphs. We also showed that both Isometric Path Cover and Isometric Path Partition admit polynomial time algorithms on chain graphs and cographs. Algorithms faster than the ones provided in this paper would be interesting. Another direction of research is to look for other graph classes where Isometric Path Cover and Isometric Path Partition admit polynomial time algorithms. Graph classes like bipartite permutation graphs, proper interval graphs, strongly chordal split graphs, *etc.* are natural candidates. The computational complexities of Isometric Path Cover and Induced Path Cover on split graphs remain open.

References

1 I. Abraham, C. Gavoille, A. Gupta, O. Neiman, and K. Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 79–88, 2014.

2 K. Asdre and S. D. Nikolopoulos. A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs. *Networks: An International Journal*, 50(4):231–240, 2007.

3 H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6(2):181–188, 1993.

4 D. Chakraborty, J. Chalopin, F. Foucaud, and Y. Vaxès. Isometric path complexity of graphs. In *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

5 D. Chakraborty, A. Dailly, S. Das, F. Foucaud, H. Gahlawat, and S. K. Ghosh. Complexity and algorithms for ISOMETRIC PATH COVER on chordal graphs and beyond. In *ISAAC 2022*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, 2022.

6 D. G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Applied Mathematics*, 3(3):163–174, 1981.

7 M. Dumas, F. Foucaud, A. Perez, and I. Todinca. On graphs coverable by k shortest paths. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

8 H. Fernau, F. Foucaud, K. Mann, U. Padariya, and R. Rao K. N. Parameterizing path partitions. In Marios Mavronicolas, editor, *Algorithms and Complexity - 13th International Conference, CIAC 2023, Larnaca, Cyprus, June 13-16, 2023, Proceedings*, volume 13898 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2023.

9 Y. Gao, D. R. Hare, and J. Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013.

10 R. Lin, S. Olariu, and G. Pruesse. An optimal path cover algorithm for cographs. *Computers & Mathematics with Applications*, 30(8):75–83, 1995.

11 Nadimpalli VR Mahadev and Uri N Peled. *Threshold graphs and related topics*. Elsevier, 1995.

12 J. Pan and G. J. Chang. Isometric-path numbers of block graphs. *Information processing letters*, 93(2):99–102, 2005.

13 S. Porschen, T. Schmidt, E. Speckenmeyer, and A. Wotzlaw. XSAT and NAE-SAT of linear CNF classes. *Discrete Applied Mathematics*, 167:1–14, 2014.

14 M. Thiessen and T. Gärtner. Active learning of convex halfspaces on graphs. *Advances in Neural Information Processing Systems*, 34:23413–23425, 2021.

15 R. Van Bevern, R. Bredereck, L. Bulteau, J. Chen, V. Froese, R. Niedermeier, and G. J. Woeginger. Partitioning perfect graphs into stars. *Journal of Graph Theory*, 85(2):297–335, 2017.

16 M. Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

On Fourier Analysis of Sparse Boolean Functions over Certain Abelian Groups

Sourav Chakraborty
Indian Statistical Institute Kolkata, India

Swarnalipa Datta
Indian Statistical Institute Kolkata, India

Pranjal Dutta
National University of Singapore, Singapore

Arijit Ghosh
Indian Statistical Institute Kolkata, India

Swagato Sanyal
Indian Institute of Technology Kharagpur, India

Abstract

Given an Abelian group $\mathcal{G}$, a Boolean-valued function $f : \mathcal{G} \to \{-1, +1\}$, is said to be s-sparse, if it has at most s-many non-zero Fourier coefficients over the domain $\mathcal{G}$. In a seminal paper, Gopalan et al. [15] proved "Granularity" for Fourier coefficients of Boolean valued functions over $\mathbb{Z}_2^n$, that have found many diverse applications in theoretical computer science and combinatorics. They also studied structural results for Boolean functions over $\mathbb{Z}_2^n$ which are *approximately Fourier-sparse*. In this work, we obtain structural results for approximately Fourier-sparse Boolean valued functions over Abelian groups $\mathcal{G}$ of the form, $\mathcal{G} := \mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$, for distinct primes p_i. We also obtain a lower bound of the form $1/(m^2 s)^{\lceil \varphi(m)/2 \rceil}$, on the absolute value of the *smallest* non-zero Fourier coefficient of an s-sparse function, where $m = p_1 \cdots p_t$, and $\varphi(m) = (p_1 - 1) \cdots (p_t - 1)$. We carefully apply probabilistic techniques from [15], to obtain our structural results, and use some non-trivial results from algebraic number theory to get the lower bound.

We construct a family of at most s-sparse Boolean functions over $\mathbb{Z}_p^n$, where $p > 2$, for arbitrarily large enough s, where the minimum non-zero Fourier coefficient is $o(1/s)$. The "Granularity" result of Gopalan et al. implies that the absolute values of non-zero Fourier coefficients of any s-sparse Boolean valued function over $\mathbb{Z}_2^n$ are $\Omega(1/s)$. So, our result shows that one *cannot* expect such a lower bound for general Abelian groups.

Using our new structural results on the Fourier coefficients of sparse functions, we design an efficient sparsity testing algorithm for Boolean function, which tests whether the given function is s-sparse, or ϵ-far from any sparse Boolean function, and it requires $\mathsf{poly}((ms)^{\varphi(m)}, 1/\epsilon)$-many queries. Further, we generalize the notion of *degree* of a Boolean function over an Abelian group $\mathcal{G}$. We use it to prove an $\Omega(\sqrt{s})$ lower bound on the query complexity of any adaptive sparsity testing algorithm.

2012 ACM Subject Classification Mathematics of computing; Theory of computation

Keywords and phrases Fourier coefficients, sparse, Abelian, granularity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.40

Related Version *Full Version*: https://arxiv.org/abs/2406.18700 [7]

Funding *Sourav Chakraborty*: Supported by the Science & Engineering Research Board of the DST, India, through the MATRICS grant MTR/2021/000318.
Pranjal Dutta: Supported by the project "Foundation of Lattice-based Cryptography", funded by NUS-NCS Joint Laboratory for Cyber Security, Singapore.
Arijit Ghosh: Arijit Ghosh is partially supported by the Science & Engineering Research Board of the DST, India, through the MATRICS grant MTR/2023/001527.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 40; pp. 40:1–40:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Boolean functions are fundamental objects of study in computer science. For a discrete domain $\mathcal{D}$, a Boolean function $f : \mathcal{D} \to \{+1, -1\}$ models a decision task where each member of $\mathcal{D}$ is classified into one of two classes. Boolean functions play a vital role in the study of digital circuits and computer hardware. They are also significant in the study of algorithms and complexity, particularly in problems where the set $\mathcal{D}$ of instances is endowed with an algebraic structure. Examples of such problems include matrix multiplication and polynomial evaluation.

The case of Boolean function complexity with $\mathcal{D} = \mathbb{Z}_2^n$ has been widely studied. These functions are often analyzed in connection with their Fourier transform (see Section 2) and a significant amount of research has focused on the structural properties of Fourier spectra of important classes of these functions. One such class that this work focuses on is that of Fourier-sparse functions. These are functions with only a few non-zero Fourier coefficients, formally defined in Definition 18. We will denote by s_f the Fourier sparsity of a Boolean function f. Fourier sparsity and Fourier-sparse functions have known connections with a variety of areas of Boolean function analysis and computational complexity like property testing [15], learning theory [20, 3], distance estimation [35] and communication complexity [23, 31, 19, 27, 22]. These connections provide enough motivation to comprehend the structure of Fourier coefficients for Fourier-sparse Boolean functions.

In this work, we extend the study of Fourier-sparse Boolean functions to the domains $\mathcal{D}$, which are Abelian groups of the form $\mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$, where $p_1, \ldots, p_t$ are distinct prime numbers. Boolean functions over general Abelian groups have been studied in both mathematics and computer science. A celebrated result regarding such functions is Chang's Lemma [10]. Chang's lemma over $\mathbb{Z}_2^n$ has found numerous applications in complexity theory and algorithms [5, 9], analysis of Boolean functions [16, 31], communication complexity [31, 21], extremal combinatorics [13], and many more. Recently, [8] improved Chang's lemma over $\mathbb{Z}_2^n$ for some special settings of parameters, where Fourier sparsity played a crucial role. One motivation to study Fourier sparsity over a broader class of Abelian groups is to investigate possible generalizations of their bounds to those groups.

Fourier analysis over finite Abelian groups in cryptography. For the past three decades, the field of cryptography has been utilizing concepts derived from Fourier analysis, specifically over finite Abelian groups. Akavia, Goldwasser, and Safra [2] have combined some of these concepts to develop a comprehensive algorithm that can detect "large" Fourier coefficients of any concentrated function on finite Abelian groups, and compute a *sparse approximation* for the same. This algorithm has gained significant attention within the cryptography community, especially regarding the notion of "bit security" of the *discrete logarithm problem* (DLP), RSA, and learning with errors (LWE) problems; see [25, 14, 1]. In particular, the "nice" structural results on the Fourier coefficients of a Boolean-valued function over the general Abelian group are of utmost importance and interest from a crypto-theoretic point of view.

Interestingly, there are strong relationships between learning, sparsity, and sampling, in the context of Fourier-sparse Boolean functions. They have been rigorously studied in [28, 29, 30]. In [28], the authors asked the following:

▶ **Question 1.1.** *What can be said about the structure of the Fourier coefficients of a Boolean function f over an Abelian group $\mathcal{G} = \mathbb{Z}_p^n$, where the support is significantly smaller compared to $\mathcal{G}$?*

Gopalan et al. [15] proved that any non-zero Fourier coefficient of a Boolean function over $\mathbb{Z}_2^n$ with Fourier sparsity at most s_f, is *at least* $\frac{1}{s_f}$ in its absolute value. This gave a satisfactory answer of Question 1.1 over $\mathbb{Z}_2^n$. Furthermore, they proved *robust versions* of their result for functions which are *approximately Fourier-sparse*. Finally, their structural results were used to design a sample-efficient algorithm to test whether a function is Fourier-sparse.

In our work, we undertake the same task for Boolean functions over Abelian groups of the form $\mathbb{Z}_p^n$, for a prime $p \geq 3$. We prove lower bounds on the absolute value of any non-zero coefficient in terms of s_f and p. We also prove a tightness result that complements our lower bounds. In particular, our bound implies that a lower bound of $\frac{1}{\Theta(s_f)}$ that [15] showed *does not* hold anymore for $\mathbb{Z}_p^n$. Finally, we use our bounds to design a testing algorithm for Fourier-sparse Boolean functions over $\mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$. This is probably the first time, we have advanced on understanding some structure on the Fourier-sparse coefficients over a general Abelian group, and shed some light on Question 1.1.

It is well-known that any Abelian group is isomorphic to a group of the form $\mathbb{Z}_{p_1^{n_1}} \times \cdots \times \mathbb{Z}_{p_t^{n_t}}$ where $p_1, \ldots, p_t$ are prime numbers and $n_1, \ldots, n_t$ are positive integers. Unfortunately, our techniques fall short of handling even a simpler case of $\mathbb{Z}_{p^2}$, fundamentally because of the *absence of a linear structure* on the solution space of systems of equations over $\mathbb{Z}_{p^2}$. Therefore, we are leaving the task of investigating the existence of similar bounds and algorithmic results when the domain of the function is a general Abelian group to future research.

Why care about Fourier-sparse Boolean functions over Abelian groups?

There has been a considerable amount of interest in studying the complexity of reconstructing or learning functions of the form $f : \mathcal{D} \to \mathbb{C}$, where $\mathcal{D}$ is a known domain (more general than a hypercube, such as a general finite Abelian group), and f is Fourier-sparse; see [30, 26, 24, 11, 34]. Fourier-sparse functions over various finite Abelian groups have gained much interest with the advancement in sparse Fourier transform algorithms [17, 18, 4]. These algorithms improve the efficiency of the standard Fast Fourier Transform algorithms by taking advantage of the sparsity itself. To reliably use sparse Fourier transform algorithms, it is beneficial to have a way to *test* if a function is s-sparse or, more generally, to *estimate* the distance of a function to the closest s-sparse function. In this work, we consider the problem of non-tolerant sparsity-testing of Boolean functions over finite Abelian groups.

Finally, apart from mathematical curiosity and potential cryptographic applications (as mentioned previously), structural results on Fourier-sparse functions $f : \mathbb{Z}_N \to \mathbb{C}$, for some $N \in \mathbb{N}$, have also found algorithmic applications in SOS-optimization and control theory. These applications have further implications in certifying maximum satisfiability (MAX-SAT) and maximum k-colorable subgraph (MkCS) problems; see [12, 33, 32].

1.1 Our results

Throughout the article, we will be working with the Abelian group $\mathcal{G} := \mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$ where p_i are primes. Let $f : \mathcal{G} \to \{-1, +1\}$, and $f(x) = \sum_{\chi \in \hat{\mathcal{G}}} \widehat{f}(\chi)\chi(x)$ be the Fourier transform of f, where $\hat{\mathcal{G}}$ is the set of characters of the Abelian group $\mathcal{G}$.

We say a Boolean function is *s-sparse*, if it has at most s non-zero Fourier coefficients. [15] proved that for any s-sparse Boolean functions over $\mathbb{Z}_2^n$, the magnitude of the Fourier coefficients are k-granular where $k = \lceil \log_2 s \rceil + 1$. A real number is *$k$-granular* if it is an integer multiple of $1/2^k$. One wonders whether such a phenomenon still holds over a more general group $\mathcal{G}$.

This notion of granularity made sense over $\mathbb{Z}_2^n$, since in this case, all the Fourier coefficients are *rational* (and hence real) numbers. But when the domain of the function is a general group $\mathcal{G}$, the Fourier coefficients are necessarily complex numbers. So, we would like to suitably define granularity, and show that such a property still holds for s-sparse Boolean-valued functions over $\mathcal{G}$. Our first conceptual contribution in this paper is to generalize the notion of granularity appropriately.

▶ **Definition 1** (Granularity). *A complex number is said to be k-granular or has granularity k with respect to $\mathbb{Z}_p$ if it is of the form $\frac{g(\omega_p)}{p^k}$, where $g(X) \in \mathbb{Z}[X]$ and ω_p is a primitive p^{th} root of unity.*

More generally, a complex number is said to be $(m_1, \ldots, m_t)$-granular with respect to $(\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_t})$ if it is of the form $\frac{g(\omega_{p_1}, \ldots, \omega_{p_t})}{p_1^{m_1} \cdots p_t^{m_t}}$, where $g(X_1, \ldots, X_t) \in \mathbb{Z}[X_1, \ldots, X_t]$ and ω_{p_i} is a primitive p_i^{th} root of unity, $i \in [k]$.

Note that, this goes well with the definition of granularity in [15] for the case of $\mathbb{Z}_2$ as ω_2 is either $+1$ or -1 and hence $g(\omega_2)$ is an integer for any $g(X) \in \mathbb{Z}[X]$.

A function $f : \mathcal{G} \to \{-1, +1\}$ is said to be $(m_1, \ldots, m_t)$-granular with respect to $(\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_t})$ if each Fourier coefficient of f is $(m_1, \ldots, m_t)$-granular.

We will also need a *robust* version of the definition of granularity of a complex number.

▶ **Definition 2** (μ-close to k-granular). *A complex number v is said to be μ-close to k-granular with respect to $\mathbb{Z}_p$ if $|v - \frac{g(\omega_p)}{p^k}| \leq \mu$, for some non-zero polynomial $g(X) \in \mathbb{Z}[X]$.*

Note that a similar notion can be defined for the case of μ-close to $(m_1, \ldots, m_t)$-granular with respect to $(\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_t})$.

Now, we are ready to formally state our two main structural results. All our results hold for Boolean-valued functions over the more general Abelian group $\mathcal{G}$. But for simplicity of presentation and ease of understanding, we first present the result for Boolean-valued functions over $\mathbb{Z}_p^n$. Later, we present more general versions of our results, see the archived version of the paper [7] for more details.

Our first theorem says that for any Boolean-valued function over $\mathbb{Z}_p^n$ that is *close* to being sparse, all its large Fourier coefficients are close to being granular. This is a generalization of the structural theorem of [15, Theorem 3.3], which was proved over $\mathbb{Z}_2^n$.

▶ **Theorem 3** (Structure theorem 1). *Let $f : \mathbb{Z}_p^n \to \{-1, +1\}$ be a Boolean-valued function and let B be the set of characters corresponding to the set of s-largest Fourier coefficients of f (in terms of magnitude). If $\sum_{\chi \in B} |\widehat{f}(\chi)|^2 \geq (1 - \mu)$ then for all $\chi \in B$, $\widehat{f}(\chi)$ is $\frac{\mu}{\sqrt{s}}$-close to $\lceil \log_p s \rceil + 1$-granular.*

For a function $f : \mathbb{Z}_p^n \to \{-1, +1\}$, $\sum_{\chi \in B} |\widehat{f}(\chi)|^2 \geq (1 - \mu)$ (where B is the set of characters corresponding to the set of s largest coefficients of f) can also be stated as "there is an s-sparse function $g : \mathbb{Z}_p^n \to \mathbb{C}$ with the ℓ_2-distance between f and g is at most $\sqrt{\mu}$". But note that this *does not* guarantee that there is an s-sparse Boolean-valued function $g : \mathbb{Z}_p^n \to \{-1, +1\}$ with ℓ_2-distance between f and g being at most $\sqrt{\mu}$. However, our second theorem proves that one can indeed find an s-sparse Boolean-valued function in a close enough vicinity, thus generalizing [15, Theorem 3.4].

▶ **Theorem 4** (Structure theorem 2). *Let $f : \mathbb{Z}_p^n \to \{-1, +1\}$ be a Boolean-valued function and let B be the set of characters corresponding to the set of s-largest Fourier coefficients of f (in terms of magnitude). If $\sum_{\chi \in B} |\widehat{f}(\chi)|^2 \geq (1 - \mu)$, with $\mu \leq \frac{1}{8(p^2 s)^{p-1}}$ then there exists an s-sparse Boolean-valued function $F : \mathbb{Z}_p^n \to \{-1, +1\}$ with ℓ_2 distance between f and F is at most $\sqrt{2}\mu$.*

Theorem 3 and Theorem 4 can be suitably generalized to functions from $\mathcal{G}$ to $\{-1,+1\}$. But for the clarity of presentation, we state the generalized theorems and present their proofs in the archived version of the paper [7].

One important corollary of Theorem 3 is that for any s-sparse Boolean function $f : \mathbb{Z}_2^n \to \{-1,+1\}$ the non-zero Fourier coefficients has magnitude at least $1/2^k$, where $k = \lceil \log_2 s \rceil + 1$. Unfortunately, such a simple corollary cannot be claimed for s-sparse functions $f : \mathbb{Z}_p^n \to \{-1,+1\}$ if $p \neq 2$. The main reason is that the definition of granularity is for complex numbers, rather than real numbers and hence such a lower bound cannot be directly deduced. However, borrowing results from algebraic number theory, we can obtain a lower bound on the Fourier coefficients of Boolean-valued functions from $\mathbb{Z}_p^n$ to $\{-1,+1\}$ for any arbitrary prime p.

▶ **Theorem 5** (Fourier coefficient lower bound). *Let $f : \mathbb{Z}_p^n \to \{-1,+1\}$, with Fourier sparsity s_f. Then, for any $\chi \in \operatorname{supp}(f)$, we have $|\hat{f}(\chi)| \geq \frac{1}{(p^2 s_f)^{\lceil (p-1)/2 \rceil}}$.*

▶ Remark. One can also prove a weaker lower bound of the form $1/((s_f+1)\sqrt{s_f})^{s_f}$, which is p-independent; for details, see the archived version of the paper [7].

Observe that the lower bound in Theorem 5 is much lower than $1/s_f$. One may wonder how tight our result is. It is known that $1/s$ is tight for the case when $p = 2$. For example, consider the function $AND : \mathbb{Z}_2^n \to \{-1,+1\}$. Its non-empty Fourier coefficients are either $\frac{1}{2^{n-1}}$ or $-\frac{1}{2^{n-1}}$, while the empty (constant) coefficient being $1 - \frac{1}{2^{n-1}}$.

To our pleasant surprise, we construct s-sparse Boolean-valued functions over $\mathbb{Z}_p^n$, for $p \geq 5$, such that they have nonzero Fourier coefficients with absolute value being $o(1/s)$.

▶ **Theorem 6** (Small Fourier coefficients). *For every prime $p \geq 5$, and large enough n, there exist a positive constant α_p that depends only on p and a function $f : \mathbb{Z}_p^n \to \{-1,+1\}$ with Fourier sparsity s_f satisfying the following property:*

$$\min_{\chi \in \operatorname{supp}(f)} \left| \widehat{f}(\chi) \right| \leq 1/s_f^{1+\alpha_p}.$$

We prove a generalized version of the lower bound result (Theorem 5) for Boolean-valued functions over $\mathcal{G}$. The above example (from Theorem 6) can also be easily extended to obtain Boolean-value functions over $\mathcal{G}$ demonstrating similar bounds on the absolute value of the non-trivial Fourier coefficients.

Finally, we design efficient algorithms for testing whether a function $f : \mathbb{Z}_p^n \to \{-1,+1\}$ is s-sparse or "far" from s-sparse-Boolean. To state our results we need to define what we mean by a function $f : \mathbb{Z}_p^n \to \{-1,+1\}$ is ϵ-far from s-sparse.

▶ **Definition 7.** *A function $f : \mathbb{Z}_p^n \to \{-1,+1\}$ is ϵ-far from s-sparse-Boolean if for every s-sparse function $g : \mathbb{Z}_p^n \to \{-1,+1\}$ the ℓ_2-distance of f and g is at least $\sqrt{\epsilon}$.*[1]

We say that an algorithm (property tester) $\mathcal{A}$ ϵ-tests $\mathcal{C}$, for a class of functions $f : \mathbb{Z}_p^n \to \{-1,+1\}$, if given access to the truth table of a function f, whether $f \in \mathcal{C}$, or f, is "ϵ-far from $\mathcal{C}$" can be tested using $\mathcal{A}$ with success probability (called the *confidence*) $\geq 2/3$. The number of queries to the truth-table of f made by $\mathcal{A}$ is called the query complexity of $\mathcal{A}$.

[1] In property testing usually the distance measure used is Hamming distance between two Boolean functions. But since we are using ℓ_2 distance in our other theorem, so for ease of presentation, we have defined the farness in terms of ℓ_2 instead of Hamming distance. Also, note that for a pair of Boolean-valued functions the square of the ℓ_2 distance and Hamming distance are the same up to a multiplicative factor of 4, see the archived version of the paper [7]

Using the structure theorems (Theorem 3 and Theorem 4), we prove the following theorem which tests sparsity of a Boolean-valued function $f : \mathbb{Z}_p^n \to \{-1, +1\}$.

▶ **Theorem 8** (Testing s-sparsity). *For a fixed prime p, there is a non-adaptive* $\mathsf{poly}(s, 1/\epsilon)$ *query algorithm with confidence $2/3$, which tests whether a given function $f : \mathbb{Z}_p^n \to \{-1, +1\}$, is s-sparse or ϵ-far from s-sparse-Boolean.*

We can also obtain a similar testing algorithm for sparsity, for a Boolean-valued function over $\mathcal{G}$. The generalized algorithm is discussed in the archived version of the paper [7].

We complement our result by showing a query-complexity lower bound for sparsity-testing algorithms. Gopalan et al. [15] gave a $\Omega(\sqrt{s})$ lower bound for s-sparsity testing algorithms over $\mathbb{Z}_2^n$. An important component of their proof was to cleverly use an alternative notion of *degree* (borrowed from [6]) of a Boolean function over $\mathbb{Z}_2$. We also give a similar lower bound over $\mathbb{Z}_p^n$, by appropriately generalizing the useful notion of degree. For details on the definition of degree, see proof idea of Theorem 9 in Section 1.2 and in the archived version of the paper [7].

▶ **Theorem 9.** *For Boolean valued functions on $\mathbb{Z}_p^n$, to adaptively test s-sparsity, the query lower bound of any algorithm is $\Omega(\sqrt{s})$.*

Theorem 9 can be generalized for Boolean valued functions on $\mathcal{G}$, which will give us the same lower bound.

1.2 Proof ideas

In this section, we briefly outline the proof ideas of our main theorems. Although some of the proofs are indeed inspired by [15] (which worked over $\mathbb{Z}_2$), the proof techniques *does not* directly generalize over $\mathbb{Z}_p$. So, we will first discuss the proof ideas, and then clarify the differences between [15] and our techniques (see Section 1.2.1). While doing so, we will try to convey the hurdles for generalizing over $\mathbb{Z}_p$. Let us first sketch the proof of Theorem 3.

Proof idea of Theorem 3. Our goal is to show that if f is μ-close to some s-sparse complex-valued function in ℓ_2, then there exists a non-zero polynomial $g(X) \in \mathbb{Z}[X]$ such that the following properties hold.

1. The sum of the absolute values of its coefficients is *at most* p^k, where $k := \lceil \log_p s \rceil + 1$.
2. The *distance* between the absolute value of each non-zero Fourier coefficient of f and $|g(\omega_p)|/p^k$ is at most $\mu/\sqrt{s}$.

To show the above, we first utilize a probabilistic method to prove that for each character $\chi_i \in B$ in the Fourier support of f, there exists a matrix $A \in \mathbb{Z}_p^{k \times n}$, and a column vector $b \in \mathbb{Z}_p^{n \times 1}$, such that – (1) χ_i is a solution of the linear system $A\chi = b$, and (2) *no other* character in the Fourier support of f is a solution of $A\chi = b$, where B be the set of s-largest Fourier coefficients of f. After establishing the existence of such A and b, we consider the Fourier transform of the projection operator for the solution space of $A\chi = b$ (see the archived version of the paper [7]). The projection operator, as the name suggests, is an operator that projects $\mathbb{Z}_p^n$ onto a linear subspace which yields a partition of the Fourier spectrum of f. Then we show that the ℓ_2 Fourier weight of $S \cap H$, i.e., $\sum_{\chi \in S \cap H} |\widehat{f}(\chi)|^2$ is upper bounded by $\mu/\sqrt{s}$, where $S = \overline{B}$, and H is a coset of $A^\perp$ that are solutions to the system of linear equations $A\chi = b$. For details, see the archived version of the paper [7].

Proof idea of Theorem 5. If we put $\mu = 0$ in Theorem 3, we get that there exists a $g \in \mathbb{Z}[X]$, with the sum of the absolute values of its coefficients is *at most* p^k, such that $|\widehat{f}(\chi)| \geq |g(\omega_p)/p^k|$, where $k = \lceil \log_p s_f \rceil + 1$, s_f being the sparsity of the Boolean valued function f. The remaining part of the proof is to show that for any polynomial g with the aforementioned properties, $|g(\omega_p)|/p^k \geq 1/(p^2 s_f)^{\lceil (p-1)/2 \rceil}$.

As stated earlier, we use a non-trivial result from algebraic number theory (Theorem 25), which states that if $f \in \mathbb{Z}[x]$, such that $f(\omega_n) \neq 0$, where ω_n be a primitive root of unity, then, $|\prod_{i \in \mathbb{Z}_n^*} f(\omega_n^i)| \geq 1$. We also use the fact that the sum of the absolute values of the coefficients of g is at most p^k, to get an upper bound on the quantities $|g(\omega_p^i)|$, for any $i \in [p-1]$. Combining these two facts, we obtain our lower bound; for details see the archived version of the paper [7].

Proof idea of Theorem 4. We first show that the given function $f : \mathbb{Z}_p^n \to \{-1, +1\}$, that is μ-close to some s-sparse complex-valued function in ℓ_2, can be written as the sum of two functions F and G, where the Fourier coefficients of F are $\lceil \log_p s \rceil + 1$-granular and the absolute value of the Fourier coefficients of G are upper bounded by $\mu/\sqrt{s}$, where p is an odd prime. This follows from Theorem 3. Then we show that the range of F is $\{-1, +1\}$, which uses the following facts:

1. $(F + G)^2 = f^2 = 1$ and
2. F^2 is $2\lceil \log_p s \rceil + 1$-granular.

We compute $\mathbb{E}[G(x)^2]$ in order to find an upper bound on the Fourier coefficients of $H := G(2f - G)$, which helps us to conclude that $\widehat{F^2}(\chi) = 0$ for all $\chi \neq \chi_0$, and $\widehat{F^2}(\chi_0) = 1$, where χ_0 is the character which takes the value 1 at all points in $\mathbb{Z}_p^n$. Then we complete the proof by showing that the $\Pr_x[x \in \mathbb{Z}_p^n | f(x) \neq g(x)]$ is $\leq \mu^2/2$, which implies that F is $\sqrt{2}\mu$ close to f in ℓ_2 (see Lemma 22). This idea has also been employed in [15]. We have also extended this proof to a more general Abelian group $\mathcal{G}$, (see the archived version of the paper [7]).

Proof idea of Theorem 6. The example that we construct to prove Theorem 6 is a simple one. The crucial observation in this regard is that there are functions from $\mathbb{Z}_p$ to $\{-1, +1\}$, whose Fourier coefficients are smaller than $1/p$. In this work, we work with one such function $\mathbb{I}_{\geq \frac{p+1}{2}} : \mathbb{Z}_p \to \{-1, +1\}$. We claim that the composition function $AND_n \circ \mathbb{I}_{\geq \frac{p+1}{2}}$ is one such desired function from $\mathbb{Z}_p^n$ to $\{-1, +1\}$, such that its minimum Fourier coefficient is less than $1/p^n$. Here, we assume a trivial bound of p^n on the Fourier sparsity of the function $AND_n \circ \mathbb{I}_{\geq \frac{p+1}{2}}$.

Proof idea of Theorem 8. We sketch the algorithmic idea over $\mathbb{Z}_p^n$, where p is an odd prime; this idea can be canonically extended to more general Abelian groups $\mathcal{G}$. The main idea of the algorithm is to partition the set of characters into buckets and estimate the *weights* of the individual buckets (that is the sum of the squares of the absolute values of the Fourier coefficients corresponding to the characters in the buckets). We know from Theorem 5 that all the coefficients of an s-sparse function are at least as large as $1/(p^2 s)^{\lceil (p-1)/2 \rceil}$. So, we are certain that if the weights of the buckets can be approximated within an additive error or $\tau/3$, where $\tau \geq 1/(p^2 s)^{p-1}$, then in the case the function is s-sparse, not more than s of the buckets can have weight more than $\tau/2$. On the other hand, we will show that if the function f is ϵ-far from any s-sparse Boolean function then with a high probability at least $(s + 1)$ buckets will have weight more than τ, making the estimated weight at least $2\tau/3$; see the archived version of the paper for more details [7].

The challenge is that estimating the weights of the buckets is not easy if the characters the randomly partitioned into buckets. Here we use the ideas from Gopalan et al [15] and appropriately modify them to handle the technicalities of working with $\mathbb{Z}_p^n$. We choose the buckets carefully. The buckets corresponds to the cosets of $H^\perp$ in $\mathbb{Z}_p^n$, where H is a random subspace of $\mathbb{Z}_p^n$ of dimension, $t = \Theta(s^2)$. For such kinds of buckets, we show that estimation of the weight can be done using a small number of samples. We also need to use the concept of "random shift", see the archived version of the paper [7], to avoid the corner case of characters being put into the trivial bucket.

Unlike [15], it becomes a bit more challenging to prove that if f is ϵ-far from any s-sparse Boolean function over $\mathbb{Z}_p^n$, then with high probability at least $(s+1)$ buckets will have weight more than τ. Since we partition the set of characters by cosets, the events whether two characters land in the same bucket (that is same coset) are *not independent* – since two characters (in the case $p \neq 2$) can be scalar multiple of each other; this is where some additional care is required (which was not the case in [15]). Under random shifts and case-by-case analysis, we can show that the two events are *not correlated*, i.e., the covariance of the corresponding indicator variables of the two events is 0; see the archived version of the paper [7]. Thus, we can use Chebyshev's inequality and then Markov's inequality to bound the number of buckets that can be of low weight, or in other words prove that the number of "heavy" weight buckets is more than $s+1$. The proof of Theorem 8 is given in the archived version of the paper [7].

Proof idea of Theorem 9. In [15], Gopalan et al. proved a query lower bound over $\mathbb{Z}_2^n$, by using a natural notion of *degree* of a Booelan function, denoted $\deg_2$. They crucially used the fact that for a Boolean function f over $\mathbb{Z}_2$, $2^{\dim(f)} \geq s_f \geq 2^{\deg_2(f)}$; this was originally proved in [6]. To define the degree, let us consider all possible restrictions $f|_{V_{b,r_1,\ldots,r_t}}$ of f, where $V_{b,r_1,\ldots,r_t}$ is a coset of $V_{0,r_1,\ldots,r_t}$ in $\mathbb{Z}_p^n$ as defined as

$$V_{b,r_1,\ldots,r_k} \;:=\; \{x \in \mathbb{Z}_p^n : r_j \cdot x \;=\; b_j \pmod{p}\ \forall j \in [t]\}\ .$$

Then the degree over $\mathbb{Z}_p$ of f, denoted by $\deg_p$, is defined in the following way.

$$\deg_p(f) \;=\; \max_{\ell}\{\ell \;= \dim(V_{b,r_1,\ldots,r_t}) \;:\; s_{f|_{V_{b,r_1,\ldots,r_t}}} \;=\; p^{\dim(V_{b,r_1,\ldots,r_t})}\},$$

where $s_{f|_{V_{b,r_1,\ldots,r_t}}}$ is the Fourier sparsity of the function $f|_{V_{b,r_1,\ldots,r_t}}$. [15] defined the degree over $\mathbb{Z}_2^n$ via similar restrictions. However, [15] argued that this is a natural definition of the degree over $\mathbb{Z}_2^n$. To argue, observe that $\widehat{f} = \frac{1}{2^n} H_n f$, where H_n is the $2^n \times 2^n$ Hadamard matrix, when $x_i \in \mathbb{Z}_2^n$ are seen in lexicographic order. Consider the restrictions $f|_{V_{b,r_1,\ldots,r_t}}$ that takes the value 1 at those points such that each entry of $\widehat{f}|_{V_{b,r_1,\ldots,r_t}}$ is nonzero. Then $\deg_2$ can be defined as the dimension of the coset $V_{b,r_1,\ldots,r_t}$ which is *largest* amongst them! In that case, all the Fourier coefficients of $f|_{V_{b,r_1,\ldots,r_t}}$ are nonzero.

Interestingly, we can also show that the above definition of $\deg_p$ is natural, mainly because $\widehat{f} = \frac{1}{p^n} V_n f$, where V_1 is a $p \times p$ *Vandermonde* matrix V_1, whose (i,j)-th entry is $(V_1)_{i,j} := \omega_p^{(i-1)(j-1)}$, and V_n is defined by taking n many *Kronecker products* of V_1, i.e., $V_n := V_1^{\otimes n} = V_1 \otimes \cdots \otimes V_1$. Note that $H_n = V_n$, over $\mathbb{Z}_2^n$. Similar to [6], one can also show that $p^{\dim(f)} \geq s_f \geq p^{\deg_p(f)}$ (see the archived version of the paper [7]). This plays a crucial role in the proof.

We first define two distributions $\mathcal{D}_{Yes}$ and $\mathcal{D}_{No}$ on the set of Boolean valued functions from $\mathbb{Z}_p^n$ to $\{-1,+1\}$. Let us choose a *random* t-dimensional subspace H of $\mathbb{Z}_p^{Ct}$, for some parameter C (to be fixed later). Let $\mathcal{C}$ be the set of all cosets of H. There are 2 main steps as follows.

1. We construct random functions f by making f a constant on each coset of $\mathcal{C}$. The constant is chosen randomly from $\{-1,+1\}$. We call this probability distribution $\boldsymbol{\mathcal{D}_{yes}}$.
2. We choose a random function f randomly from $\mathbb{Z}_p^{Ct}$, conditioned on the fact that f is $2-\tau$ far in ℓ_2 from any function which has $\deg_p = t$, where τ is as defined in Theorem 8. Let us call this distribution $\boldsymbol{\mathcal{D}_{No}}$.

We show that if an adaptive query algorithm makes less than $q < \Omega(p^{t/2})$ queries, then the total variation distance $||\mathcal{D}_{Yes} - \mathcal{D}_{No}||_{TV}$ between the two distributions $\mathcal{D}_{Yes}$ and $\mathcal{D}_{No}$ is $\leq \frac{1}{3}$. This proves that any adaptive query algorithm which distinguishes between $\mathcal{D}_{Yes}$ and $\mathcal{D}_{No}$, i.e., where $||\mathcal{D}_{Yes} - \mathcal{D}_{No}||_{TV} > \frac{1}{3}$, *must* make at least $\Omega(p^{t/2})$ queries. This essentially proves Theorem 9.

1.2.1 Comparison with Gopalan et al. [15]

In this section, we discuss the main differences and points between our proof and the one by Gopalan et al. [15]. Our proofs (and the analysis of the testing algorithm) are more elaborate, subtle, case-dependent, and complicated than [15].

Linear dependence is *bad*. [15] used the similar idea as in Theorem 3, see the archived version of the paper for more details [7]. However, their proof took advantage of the fact that any two distinct non-zero vectors in $\mathbb{Z}_2^n$ are *linearly independent*. This is *not true* for $\mathbb{Z}_p^n$, as distinct vectors can be scalar multiples of each other; e.g., $(1,\cdots,1)$ and $(2,\cdots,2)$ in $\mathbb{Z}_p^n$, for any $p \geq 3$. Thus, our proof technique is capable of handling and overcoming these difficulties effectively. This is also why our analysis of the structure theorems and algorithms are very much case-dependent.

Different granularity. In case of $\mathbb{Z}_2^n$, [15] defined a function $f : \mathbb{Z}_2^n \to \{-1,+1\}$ to be k-granular, where k is a positive integer, if all of its nonzero Fourier coefficients are an *integer multiple* of $\frac{1}{2^k}$. In the case of $\mathbb{Z}_p^n$, since the Fourier coefficients are truly complex numbers, we define a function $f : \mathbb{Z}_p^n \to \{-1,+1\}$ to be k-granular if all of its nonzero Fourier coefficients are of the form $\frac{g(\omega_p)}{p^k}$ (see Definition 1), where $g \in \mathbb{Z}[X]$ and ω_p is a primitive p^{th} root of unity.

Algebraic integer lower bounds are *harder*. While proving a lower bound on the absolute value of the Fourier coefficients, in the case of $\mathbb{Z}_2^n$, proving a lower bound on the magnitude of a Fourier coefficient [15] translates to proving a lower bound on $|g(-1)|/2^k$ (since $\omega_2 = -1$). Note that $|g(-1)| \geq 1$, because $g(-1) \in \mathbb{Z} \setminus \{0\}$; hence the lower bound follows. However, for a general prime p, ω_p is *truly* a complex number, and hence, $g(\omega_p)$ is *no longer* an integer (and therefore, there is no way to conclude that $|g(\omega_p)| \geq 1$). We use a non-trivial result from algebraic number theory (Theorem 25) to tackle this problem, in the proof of Theorem 5. We also crucially use the fact that the sum of the absolute values of the coefficients of g is at most p^k, to get an upper bound on the quantities $|g(\omega_p^i)|$, for any $i \in [p-1]$. Clearly, this part requires much more non-triviality than [15].

Do not expect a tight linear lower bound. For $p = 2$, it follows from [15] that the lower bound on the absolute value of the Fourier coefficients is equal to $\frac{1}{s_f}$. We show in Theorem 6 that there exists a family of Boolean valued functions on $\mathbb{Z}_p^n$, $p > 3$ such that $\min_{\chi \in \mathrm{supp}(f)} \left|\widehat{f}(\chi)\right| \leq 1/s_f^{1+\alpha_p}$, hence proving that the lower bound on the absolute value of the Fourier coefficients is not linear in $\frac{1}{s_f}$ in case of $p > 3$.

Generalizing the notion of degree. For functions from $\mathbb{Z}_2^n$ to $\{-1,+1\}$, it is known that $\widehat{f} = \frac{1}{p^n} H_n f$, where H_n denotes the Hadamard matrices (see [6]). Then, the degree $\deg_2$ is defined naturally in the case of $\mathbb{Z}_2^n$. The definition of $\deg_p$ becomes slightly challenging, since the relation $\widehat{f} = \frac{1}{p^n} H_n f$ is simply *false* over $\mathbb{Z}_p^n$. To define $\deg_p$ for Boolean valued functions on $\mathbb{Z}_p^n$ (see the archived version of the paper [7]), where p is an odd prime, we have defined the matrix V_1 using Vandermonde matrix, and have inductively defined the matrices V_n, where n is a positive integer, via Kronecker product. Then we have shown that $\widehat{f} = \frac{1}{p^n} V_n f$, which helps us to define $\deg_p(f)$. We claim that this is an appropriate generalization of degree, since $H_n = V_n$, over $\mathbb{Z}_2^n$.

Finally, we remark that because of the *absence of a linear structure* on the solution space of systems of equations over arbitrary Abelian groups (e.g. $\mathbb{Z}_{p^2}$), we could not generalize this to arbitrary Abelian groups.

2 Preliminaries

2.1 Fourier Analysis over $\mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$

$\mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$ forms a finite Abelian group under addition whose order is $p_1^{n_1} \cdots p_t^{n_t}$, where $p_1, \ldots, p_t$ are distinct primes. Its individual components form a vector space, that is, $\mathbb{Z}_{p_i}^{n_i}$ is a vector space over the field $\mathbb{Z}_{p_i}$ for all $i \in \{1, 2, \ldots t\}$ whenever p_i's are primes. Throughout this paper we will denote $\mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$ by $\mathcal{G}$. So $\mathcal{G}$ is a finite Abelian group with $|\mathcal{G}| = p_1^{n_1} \cdots p_t^{n_t}$, where $|.|$ denotes the order of $\mathcal{G}$.

We will denote this root of unity by ω_p a p^{th} primitive root of unity, that is $e^{2\pi\iota/p}$.

Let us begine by defining the characters of $\mathcal{G}$.

▶ **Definition 10** (Character). *A character of $\mathcal{G}$ is a homomorphism $\chi : \mathcal{G} \to \mathbb{C}^\times$ of $\mathcal{G}$, that is, χ satisfies the following:*

$$\chi(x+y) = \chi(x)\chi(y), \;\; x, y \in \mathcal{G}.$$

Equivalently, a character χ of $\mathcal{G}$ can be defined by

$$\chi(x_1, \ldots, x_t) = \chi_{r_1}(x_1) \cdots \chi_{r_t}(x_t) = \omega_{p_1}^{r_1 \cdot x_1} \cdots \omega_{p_t}^{r_t \cdot x_t},$$

where χ_{r_i} is a character of $\mathbb{Z}_{p_i}^{n_i}$ for each i and is defined by $\chi_{r_i}(x_i) = \omega_{p_i}^{r_i \cdot x_i}$, $x_i, r_i \in \mathbb{Z}_{p_i}^{n_i}$ for all i, $r_i \cdot x_i = \sum_{j=1}^{n_i} r_{i(j)} x_{i(j)}$, $r_{i(j)}$ and $x_{i(j)}$ being the j^{th} component of r_i and x_i respectively (It follows from the fact that any character χ of $\mathcal{G}$ can be written as the product of characters of $\mathbb{Z}_{p_1}^{n_1}, \ldots, \mathbb{Z}_{p_t}^{n_t}$). Let us denote this character by $\chi_{r_1,\ldots,r_t}$.

Now let us look at some properties of characters.

▶ **Lemma 11.** *Let χ be a character of $\mathcal{G}$. Then,*

1. $\chi_0(x) = 1$ *for all* $x \in \mathcal{G}$.
2. $\chi(-x) = \chi(x)^{-1} = \overline{\chi(x)}$ *for all* $x \in \mathcal{G}$.
3. *For any character χ of $\mathcal{G}$, where $\chi \neq \chi_0$, $\sum_{x \in \mathcal{G}} \chi(x) = 0$.*
4. $|\chi_0(x)| = 1$ *for all* $x \in \mathcal{G}$.

Now let us define the dual group of $\mathcal{G}$.

▶ **Definition 12** (Dual group). *The set of characters of $\mathcal{G}$ forms a group under the operation $(\chi\psi)(x) = \chi(x)\psi(x)$ and is denoted by $\widehat{\mathcal{G}}$, where χ and ψ are characters of $\mathcal{G}$. $\widehat{\mathcal{G}}$ is called the dual group of $\mathcal{G}$.*

The following theorem states that $\mathcal{G}$ is isomorphic to its dual group.

▶ **Theorem 13.** *$\widehat{\mathcal{G}} \cong \mathcal{G}$.*

Let us look at the definition of Fourier transform for functions on $\mathcal{G}$.

▶ **Definition 14** (Fourier transform)**.** *For any function $f : \mathcal{G} \to \mathbb{C}$, the Fourier transform $\widehat{f} : \widehat{\mathcal{G}} \to \mathbb{C}$ is defined by*

$$\widehat{f}(\chi_{r_1,\dots,r_t}) = \frac{1}{|\mathcal{G}|} \sum_{x\in\mathcal{G}} f(x)\omega_{p_1}^{-r_1\cdot x_1} \cdots \omega_{p_t}^{-r_t\cdot x_t},$$

where $x_i, r_i \in \mathbb{Z}_{p_i}^{n_i}$ for all i, and $r_i \cdot x_i = \sum_{j=1}^{n_i} r_{i(j)}x_{i(j)}$, $r_{i(j)}$ and $x_{i(j)}$ being the j^{th} component of r_i and x_i respectively.

▶ Remark 15. The Fourier transform of a function $f : \mathcal{G} \to \mathbb{C}$ is defined by

$$\widehat{f}(\chi) = \frac{1}{|\mathcal{G}|} \sum_{x\in\mathcal{G}} f(x)\overline{\chi(x)},$$

where $\overline{\chi(x)}$ is the conjugate of $\chi(x)$. The Definition 14 follows from this, as $\chi = \chi_{r_1,\dots,r_t}$ for some $r_i \in \mathbb{Z}_{p_i}^{n_i}$, $i \in \{1, \dots, t\}$.

The following theorem states that any function from $\mathcal{G}$ to $\mathbb{C}$ can be written as a linear combination of characters of $\mathcal{G}$.

▶ **Theorem 16** (Fourier inversion formula)**.** *Any function $f : \mathcal{G} \to \mathbb{C}$ can be uniquely written as a linear combination of characters of $\mathcal{G}$, that is,*

$$f(x) = \sum_{\chi_{r_1,\dots,r_t}\in\widehat{\mathcal{G}}} \widehat{f}(\chi_{r_1,\dots,r_t})\omega_{p_1}^{r_1\cdot x_1} \cdots \omega_{p_t}^{r_t\cdot x_t},$$

where $x_i, r_i \in \mathbb{Z}_{p_i}^{n_i}$ for all i, $r_i \cdot x_i = \sum_{j=1}^{n_i} r_{i(j)}x_{i(j)}$, $r_{i(j)}$ and $x_{i(j)}$ being the j^{th} component of r_i and x_i respectively.

▶ **Theorem 17** (Parseval)**.** *For any two functions $f, g : \mathcal{G} \to \mathbb{C}$,*

$$\mathbb{E}_{x\in\mathcal{G}}[f(x)\overline{g(x)}] = \sum_{\chi\in\widehat{\mathcal{G}}} \widehat{f}(\chi)\overline{\widehat{g}(\chi)}.$$

More specifically, if $f : \mathcal{G} \to \{-1, +1\}$ is a Boolean-valued function then

$$\sum_{\chi\in\widehat{\mathcal{G}}} |\widehat{f}(\chi)|^2 = 1.$$

Now let us define the Fourier sparsity of a function f on $\mathcal{G}$.

▶ **Definition 18** (Sparsity and Fourier Support)**.**

- *The Fourier sparsity s_f of a function $f : \mathcal{G} \to \mathbb{C}$ is defined to be the number of non-zero Fourier coefficients in the Fourier expansion of f (Theorem 16). In this paper, by sparsity of a function, we will mean the Fourier sparsity of the function.*
- *Fourier support* $\mathrm{supp}(f)$ *of a function $f : \mathcal{G} \to \mathbb{C}$ denotes the set $\left\{\chi \mid \widehat{f}(\chi) \neq 0\right\}$.*

The following lemma states an important property of characters of a subgroup $H = H_1 \times \cdots \times H_t$ of $\mathcal{G}$.

▶ **Lemma 19.** *$\sum_{h\in H}\chi_h = |H|\cdot 1_{H^\perp}$, where $H = H_1\times\cdots\times H_t$ is a subgroup of $\mathcal{G}$, and $h = (h_1,\dots,h_t)$, $h_i\in H_i$ $\forall i$. That is,*

$$\sum_{h_1\in H_1,\dots,h_t\in H_t}\chi_{h_1}\cdots\chi_{h_t} = |H_1|\cdots|H_t|\cdot 1_{H_1^\perp}\cdots 1_{H_t^\perp}.$$

Here, for each i, H_i is a subgroup of $\mathbb{Z}_{p_i}^{n_i}$ and hence a subspace $\mathbb{Z}_{p_i}^{n_i}$, as $\mathbb{Z}_{p_i}^{n_i}$ is a vector space.

The proof of Lemma 19 is given in the archived version of the paper [7] due to lack of space.

▶ **Lemma 20.** *Let f, g be two Boolean valued functions from $\mathcal{G}$ to $\{-1,+1\}$. Then,*

$$|\widehat{fg}(\chi)| \le ||f||_2||g||_2,$$

for any character $\chi\in\widehat{\mathcal{G}}$.

The proof of Lemma 20 is given in the archived version of the paper [7] due to lack of space.

Now let us formally define the notion of ϵ-close and ϵ-far in ℓ_2 below.

▶ **Definition 21** (μ-close to s-sparse). *Let f and g be two functions with domain $\mathbb{Z}_p^n$ and range $\mathbb{C}$. Then the square of the ℓ_2 distance between f and g is defined as $\mathbb{E}_{x\in\mathcal{G}}[|f(x)-g(x)|^2]$. By Parseval's identity the square of the ℓ_2-distance between f and g can also be written as $\sum_{\chi\in\widehat{\mathbb{Z}_p^n}}|\widehat{(f-g)}(\chi)|^2$.*

We say that f is ϵ-close to g in ℓ_2 if the square of the ℓ_2 distance between f and g is less than ϵ. Similarly, f is ϵ-far from g in ℓ_2 if the square of the ℓ_2 distance between f and g is at least ϵ.

The following lemma gives us a relation between the ℓ_2 distance between two Boolean valued functions f and g defined in Definition 21 and $\Pr_x[x\in\mathcal{G}|f(x)\neq g(x)]$.

▶ **Lemma 22.** *The square of the ℓ_2 distance between two Boolean valued functions f and g defined in Definition 21 is equal to $4\Pr_x[x\in\mathcal{G}|f(x)\neq g(x)]$.* [2]

The proof of Lemma 22 is given in the archived version of the paper [7] due to lack of space.

Now let us define the total variation distance between two probability distributions.

▶ **Definition 23.** *Let $(\Omega,\mathcal{F})$ be a probability space, and P and Q be probability distributions defined on $(\Omega,\mathcal{F})$. The total variation distance between P and Q is defined in the following way.*

$$||P-Q||_{TV} = \sup_{A\in\mathcal{F}}|P(A)-Q(A)|.$$

▶ **Lemma 24.** *Given two probability distributions P and Q on a probability space $(\Omega,\mathcal{F})$, the total variation distance between P and Q is half of the L_1 distance between them. That is,*

$$||P-Q||_{TV} = \frac{1}{2}\sum_x|P(x)-Q(x)|.$$

[2] If f and g are two Booelan-valued functions then $\Pr_x[x\in\mathcal{G}|f(x)\neq g(x)]$ is also called the Hamming distance between the two functions. So the ℓ_2 norm between two Boolean-valued functions is 4 times the Hamming distance between two Boolean-valued functions.

3 Lower bound on the Fourier coefficients

In this section, we will prove Theorem 5 assuming Theorem 3, which gives us a lower bound on the Fourier coefficients of functions from $\mathbb{Z}_p^n$ to $\{-1,+1\}$, where p is a prime number. This is a generalization on the granularity of a function $f : \mathbb{Z}_2^n \to \mathbb{R}$ when the domain of the function is $\mathbb{Z}_p^n$. We will use the following theorem; for more details see the archived version of the paper [7].

▶ **Theorem 25.** *For $n \in \mathbb{Z}$, let ω_n be a primitive root of unity. Let $f \in \mathbb{Z}[x]$, such that $f(\omega_n) \neq 0$. Then,*

$$\left|\prod_{i\in\mathbb{Z}_n^*} f(\omega_n^i)\right| \geq 1 .$$

Proof of Theorem 5. If we put $\mu = 0$ in Theorem 3, we get that there exists a $g \in \mathbb{Z}[X]$, such that $|\widehat{f}(\chi)| \geq |g(\omega_p)/p^k|$, where $k = \lceil \log_p s_f \rceil + 1$, s_f being the sparsity of the Boolean valued function f. We know by Theorem 25 that $|\prod_{i=1}^{p-1} g(\omega_p^i)| \geq 1$. Therefore,

$$\left|\prod_{i=1}^{p-1} g(\omega_p^i)\right| \geq 1 \Rightarrow \prod_{i=1}^{p-1}\left|\frac{g(\omega_p^i)}{p^k}\right| \geq \left(\frac{1}{p^k}\right)^{p-1} \Rightarrow \left|\frac{g(\omega_p)}{p^k}\right| \geq \frac{1}{p^{k(p-1)}} \geq \frac{1}{(p^2 s_f)^{p-1}}.$$

For $p = 2$, $g(\omega_p^i)$ is an integer, so

$$\left|\frac{g(\omega_p)}{p^k}\right| \geq \frac{1}{4s_f} \tag{3.1}$$

For $p \neq 2$, the conjugate of $g(\omega_p^i)$, namely $\overline{g(\omega_p^i)}$, is nothing but $= g(\omega_p^{p-i})$. Since $|g|_1 \leq p^k$, it follows that for any $i \in [p-1]$, $|g(\omega_p^i)/p^k| \leq 1$. Therefore,

$$\begin{aligned}
\left|\prod_{i=1}^{p-1} g(\omega_p^i)\right| \geq 1 &\Rightarrow \prod_{i=1}^{(p-1)/2} |g(\omega_p^i)|^2 \geq 1 \\
&\Rightarrow \prod_{i=1}^{(p-1)/2} \left|\frac{g(\omega_p^i)}{p^k}\right|^2 \geq (\frac{1}{p^k})^{p-1} \\
&\Rightarrow \left|\frac{g(\omega_p)}{p^k}\right|^2 \geq \frac{1}{p^{k(p-1)}} \\
&\Rightarrow \left|\frac{g(\omega_p)}{p^k}\right| \geq \frac{1}{p^{k(p-1)/2}} \geq \frac{1}{(p^2 s_f)^{(p-1)/2}} .
\end{aligned} \tag{3.2}$$

So, from Equation (3.1) and Equation (3.2), we have

$$\left|\frac{g(\omega_p)}{p^k}\right| \geq \frac{1}{(p^2 s_f)^{\lceil (p-1)/2 \rceil}} . \qquad ◀$$

▶ Remark 26. Let p be a prime. The proof-technique of Theorem 5 gives the following. If the Fourier coefficients of a Boolean function f are k granular, where $k = \lceil \log_p s \rceil + 1$, then the Fourier coefficients of f^2 are $2k$-granular, and their absolute values are $\geq \frac{1}{(p^2 s)^{(p-1)}}$.

4 Sparse Boolean-valued function with small Fourier coefficients

In this section, for a fixed prime $p \geq 5$, and arbitrarily large s, we give an example of a function $f : \mathbb{Z}_p^n \to \{-1, +1\}$, such that the minimum of the absolute value of its Fourier coefficients is at most $o(1/s)$. In this section, we give the details of the construction of Theorem 6; see the archived version of the paper [7].

To prove Theorem 6 we define a function, which is basically composition of AND_n and univariate Threshold functions; we call AT.

▶ **Definition 27.** *Let us define function* $\mathsf{AT} : \mathbb{Z}_p^n \to \{-1, +1\}$ *by*

$$\mathsf{AT}(x_1, x_2, \ldots, x_n) \;:=\; \text{AND}_n\left(\mathbb{I}_{\geq \frac{p+1}{2}}(x_1), \mathbb{I}_{\geq \frac{p+1}{2}}(x_2), \ldots, \mathbb{I}_{\geq \frac{p+1}{2}}(x_n)\right),$$

where the univariate Threshold function $\mathbb{I}_{\geq \frac{p+1}{2}} : \mathbb{Z}_p \to \{-1, +1\}$, *is defined as:*

$$\mathbb{I}_{\geq \frac{p+1}{2}}(x) \;=\; \begin{cases} 1 & \text{for } x \geq \frac{p+1}{2} \\ -1 & \text{otherwise.} \end{cases}$$

▶ **Lemma 28.** *There is a Fourier coefficient of* AT, *whose absolute value is* $\frac{1}{p^{nc}}$, *where c is a constant* > 1.

The proof of Lemma 28 is given in the archived version of the paper [7]. Now we are ready to prove Theorem 6.

Proof of Theorem 6. This directly follows, as we can claim from Lemma 28 that there exists a family of functions in $\mathbb{Z}_p^n$ whose absolute value of the minimum coefficient is not linear in $\frac{1}{\text{sparsity}}$, but actually $= \Omega(\frac{1}{\text{sparsity}^{1+\epsilon_p}})$. where $\epsilon_p > 0$, is a p-dependent constant. ◀

▶ **Lemma 29.** *There exists a function whose one of the Fourier coefficients is less than that of* AT.

The proof of Lemma 29 is given in the archived version of the paper [7].

5 Testing Algorithm for Sparsity

Here we present the whole algorithm (Algorithm 1) for sparsity testing over $\mathbb{Z}_p^n$ (see next page). However, we describe the various steps of the algorithm in the archived version of the paper [7], including the correctness and the analysis of the algorithm in the archived version of the paper [7].

6 Conclusion

Gopalan et al. [15] was the first to study the problem of testing Fourier sparsity of Boolean function over $\mathbb{Z}_2^n$. Along the way, they were able to drive fundamental properties of Boolean functions over $\mathbb{Z}_2^n$, like *Granularity* of the Fourier spectrum, that have found many other applications [3]. In this work, we have extended their results for groups that can be written as $\mathbb{Z}_{p_1}^{n_1} \times \cdots \times \mathbb{Z}_{p_t}^{n_t}$.

The most natural question that arises from our work will be to study this problem for (finite) general Abelian groups. Unfortunately, our work does not extend to finite Abelian groups, because of the *absence* of the component-wise vector space structure. In particular, it would be nice to get a lower bound on the absolute value of non-zero Fourier coefficients of a sparse Boolean-valued function over $\mathbb{Z}_{p^2}^n$. In any way, our results in this paper should be seen as a first step toward solving the lower bound problem over finite Abelian group.

Finally, we ask whether it is possible to show a better (*p-dependent*) lower bound over $\mathbb{Z}_p^n$ on the query-complexity of any adaptive sparsity testing algorithm.

Algorithm 1 Test Sparsity.

```
Input: $s, \epsilon$, and query access to $f : \mathbb{Z}_p^n \to \{-1, +1\}$.
Output: YES, if $f$ is $s$-sparse, and NO, if it is $\epsilon$-far from any Boolean valued function.
Parameter setting: Set $t := \lceil 2\log_p s + \log_p 20 \rceil + 1, \tau := \min(\frac{\epsilon^2}{40p^t}, \frac{1}{(p^2 s)^{p-1}})$ $M := O(\log(p^t) \cdot \frac{1}{\tau^2})$
Choose $v_1, \ldots, v_t$ linearly independent vectors uniformly at random from $\mathbb{Z}_p^n$.
Let $H = \text{Span}\{v_1, \ldots, v_t\}$
Pick $(z_1, x_1), \ldots, (z_M, x_M)$ uniformly and independently from $H \times \mathbb{Z}_p^n$
Query $f(x_1), \ldots, f(x_M)$ and $f(x_1 - z_1), \ldots, f(x_M - z_M)$
for For every $r \in \mathbb{Z}_p^n$ do
    Let $\frac{1}{M}\sum_{i=1}^M \chi_r(z_i) f(x_i) f(x_i - z_i)$ be the estimate of $wt(r + H^\perp)$
end for
if number of $r$ for which the estimate of $wt(r + H^\perp)$ is $\geq \frac{2\tau}{3}$ is $\leq s$ then
    Output YES
else
    Output NO
end if
```

References

1 Adi Akavia. *Learning noisy characters, multiplication codes, and cryptographic hardcore predicates.* PhD thesis, Massachusetts Institute of Technology, 2008.

2 Adi Akavia, Shafi Goldwasser, and Shmuel Safra. Proving Hard-Core Predicates Using List Decoding. In *FOCS*, pages 146–157, 2003.

3 Srinivasan Arunachalam, Sourav Chakraborty, Troy Lee, Manaswi Paraashar, and Ronald De Wolf. Two New Results About Quantum Exact Learning. *Quantum*, 5:587, 2021.

4 Mitali Bafna, Jack Murtagh, and Nikhil Vyas. Thwarting Adversarial Examples: An L_0-Robust Sparse Fourier Transform. In *NeurIPS*, volume 31, 2018.

5 Eli Ben-Sasson, Noga Ron-Zewi, Madhur Tulsiani, and Julia Wolf. Sampling-Based Proofs of Almost-Periodicity Results and Algorithmic Applications. In *ICALP*, volume 8572, pages 955–966, 2014.

6 Anna Bernasconi and Bruno Codenotti. Spectral Analysis of Boolean Functions as a Graph Eigenvalue Problem. *IEEE Trans. Computers*, 48(3):345–351, 1999.

7 Sourav Chakraborty, Swarnalipa Datta, Pranjal Dutta, Arijit Ghosh, and Swagato Sanyal. On Fourier analysis of sparse Boolean functions over certain Abelian groups, 2024. `arXiv:2406.18700`.

8 Sourav Chakraborty, Nikhil S. Mande, Rajat Mittal, Tulasimohan Molli, Manaswi Paraashar, and Swagato Sanyal. Tight Chang's-Lemma-Type Bounds for Boolean Functions. In *FSTTCS*, volume 213, pages 10:1–10:22, 2021.

9 Siu On Chan, James R. Lee, Prasad Raghavendra, and David Steurer. Approximate Constraint Satisfaction Requires Large LP Relaxations. *J. ACM*, 63(4):34:1–34:22, 2016.

10 Mei-Chu Chang. A polynomial bound in Freiman's theorem. *Duke Mathematical Journal*, 113(3):399–419, 2002.

11 Xue Chen and Anindya De. Reconstruction under outliers for Fourier-sparse functions. In *SODA*, pages 2010–2029, 2020.

12 Hamza Fawzi, James Saunderson, and Pablo A Parrilo. Sparse sum-of-squares certificates on finite Abelian groups. In *IEEE Conference on Decision and Control (CDC)*, pages 5909–5914, 2015.

13 Ehud Friedgut, Jeff Kahn, Gil Kalai, and Nathan Keller. Chvátal's conjecture and correlation inequalities. *J. Comb. Theory, Ser. A*, 156:22–43, 2018.

14 Steven D. Galbraith, Joel Laity, and Barak Shani. Finding Significant Fourier Coefficients: Clarifications, Simplifications, Applications and Limitations. *Chic. J. Theor. Comput. Sci.*, 2018, 2018.

15 Parikshit Gopalan, Ryan O'Donnell, Rocco A Servedio, Amir Shpilka, and Karl Wimmer. Testing Fourier dimensionality and sparsity. *SIAM Journal on Computing*, 40(4):1075–1100, 2011.

16 Ben Green and Tom Sanders. Boolean functions with small spectral norm. *Geometric and Functional Analysis*, 18(1):144–162, 2008.

17 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse Fourier transform. In *STOC*, pages 563–578, 2012.

18 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and practical algorithm for sparse Fourier transform. In *SODA*, pages 1183–1194, 2012.

19 Hamed Hatami, Kaave Hosseini, and Shachar Lovett. Structure of protocols for XOR functions. *SIAM Journal on Computing*, 47(1):208–217, 2018.

20 Ishay Haviv and Oded Regev. The list-decoding size of Fourier-sparse Boolean functions. *ACM Transactions on Computation Theory*, 8(3):1–14, 2016.

21 Kaave Hosseini, Shachar Lovett, and Grigory Yaroslavtsev. Optimality of Linear Sketching Under Modular Updates. In *CCC*, volume 137, pages 13:1–13:17, 2019.

22 Nikhil Shekhar Mande and Swagato Sanyal. On Parity Decision Trees for Fourier-Sparse Boolean Functions. *ACM Transactions on Computation Theory*, 16(2):1–26, 2024.

23 Ashley Montanaro and Tobias Osborne. On the communication complexity of XOR functions, 2010. arXiv:0909.3392.

24 Lucia Morotti. Reconstruction of Fourier sparse signals over elementary Abelian groups. *Electronic Notes in Discrete Mathematics*, 43:161–167, 2013.

25 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40, 2009.

26 Mark Rudelson and Roman Vershynin. On sparse reconstruction from Fourier and Gaussian measurements. *Communications on Pure and Applied Mathematics*, 61(8):1025–1045, 2008.

27 Swagato Sanyal. Fourier Sparsity and Dimension. *Theory Comput.*, 15:1–13, 2019.

28 Meera Sitharam. Pseudorandom generators and learning algorithms for AC. In *STOC*, pages 478–486, 1994.

29 Meera Sitharam and Timothy Straney. Sampling Boolean functions over Abelian groups and applications. *Applicable Algebra in Engineering, Communication and Computing*, 11:89–109, 2000.

30 Meera Sitharam and Timothy Straney. Derandomized Learning of Boolean Functions over Finite Abelian Groups. *International Journal of Foundations of Computer Science*, 12(04):491–516, 2001.

31 Hing Yin Tsang, Chung Hoi Wong, Ning Xie, and Shengyu Zhang. Fourier Sparsity, Spectral Norm, and the Log-Rank Conjecture. In *FOCS*, pages 658–667, 2013.

32 Jianting Yang, Ke Ye, and Lihong Zhi. Computing sparse Fourier sum of squares on finite abelian groups in quasi-linear time, 2023. arXiv:2201.03912.

33 Jianting Yang, Ke Ye, and Lihong Zhi. Fourier sum of squares certificates, 2023. arXiv:2207.08076.

34 Jianting Yang, Ke Ye, and Lihong Zhi. Lower Bounds of Functions on Finite Abelian Groups. In *International Computing and Combinatorics Conference*, pages 157–170, 2023.

35 Grigory Yaroslavtsev and Samson Zhou. Fast Fourier Sparsity Testing. In *SOSA*, pages 57–68, 2020.

Krenn-Gu Conjecture for Sparse Graphs

L. Sunil Chandran
Indian Institute of Science, Bengaluru, India

Rishikesh Gajjala
Indian Institute of Science, Bengaluru, India

Abraham M. Illickan
University of California, Irvine, CA, USA

Abstract

Greenberger–Horne–Zeilinger (GHZ) states are quantum states involving at least three entangled particles. They are of fundamental interest in quantum information theory, and the construction of such states of high dimension has various applications in quantum communication and cryptography. Krenn, Gu and Zeilinger discovered a correspondence between a large class of quantum optical experiments which produce GHZ states and edge-weighted edge-coloured multi-graphs with some special properties called the *GHZ graphs*. On such GHZ graphs, a graph parameter called *dimension* can be defined, which is the same as the dimension of the GHZ state produced by the corresponding experiment. Krenn and Gu conjectured that the dimension of any GHZ graph with more than 4 vertices is at most 2. An affirmative resolution of the Krenn-Gu conjecture has implications for quantum resource theory. Moreover, this would save huge computational resources used for finding experiments which lead to higher dimensional GHZ states. On the other hand, the construction of a GHZ graph on a large number of vertices with a high dimension would lead to breakthrough results.

In this paper, we study the existence of GHZ graphs from the perspective of the Krenn-Gu conjecture and show that the conjecture is true for graphs of vertex connectivity at most 2 and for cubic graphs. We also show that the minimal counterexample to the conjecture should be 4-connected. Such information could be of great help in the search for GHZ graphs using existing tools like PyTheus. While the impact of the work is in quantum physics, the techniques in this paper are purely combinatorial, and no background in quantum physics is required to understand them.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Graph colourings, Perfect matchings, Quantum Physics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.41

Related Version *Full Version*: `https://arxiv.org/abs/2407.00303`

Funding *L. Sunil Chandran*: Supported by SERB Core Research Grant CRG/2022/006770: "Bridging Quantum Physics with Theoretical Computer Science and Graph Theory".

1 Introduction

Quantum entanglement theory implies that two particles can influence each other, even though they are separated over large distances. In 1964, Bell demonstrated that quantum mechanics conflicts with our classical understanding of the world, which is local (i.e. information can be transmitted maximally with the speed of light) and realistic (i.e. properties exist prior to and independent of their measurement) [2]. Later, in 1989, Greenberger, Horne, and Zeilinger (abbreviated as GHZ) studied what would happen if more than two particles are entangled [8]. Such states in which three particles are entangled $(|GHZ_{3,2}\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle))$ were observed rejecting local-realistic theories [4, 21]. While the study of such states started purely out of fundamental curiosity [24, 16, 17], they are now used in many applications in quantum information theory, such as quantum computing [11]. They are also essential for early tests of quantum computing tasks [28], and quantum cryptography in quantum networks[22].

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 41; pp. 41:1–41:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Zeilinger became a co-recipient of the Nobel Prize for Physics in 2022, for experiments with entangled photons, establishing the violation of Bell inequalities and pioneering quantum information science. We note that the work on experimentally constructing GHZ states is at the heart of Zeilinger's Nobel prize-winning work [1]. Increasing the number of particles involved and the dimension of the GHZ state is essential both for foundational studies and practical applications. Motivated by this, a huge effort is being made by several experimental groups around the world to push the size of GHZ states. Photonic technology is one of the key technologies used to achieve this goal [28, 27]. The Nobel Laureate himself, with some co-authors, proposed a scheme of optical experiments in order to achieve this, which gives an opportunity for graph theorists to get involved in this fundamental research: In 2017, Krenn, Gu and Zeilinger [14] discovered (and later extended [10, 9]) a bridge between experimental quantum optics and graph theory. They observed that large classes of quantum optics experiments (including those containing probabilistic photon pair sources, deterministic photon sources and linear optics elements) can be represented as an edge-coloured edge-weighted graph, though the edge-colouring goes a little beyond what graph theorists are used to. Conversely, every edge-coloured edge-weighted graph (also referred to as an experiment graph) can be translated into a concrete experimental setup. This technique has led to the discovery of new quantum interference effects and connections to quantum computing [10]. Furthermore, it has been used as the representation of efficient AI-based design methods for new quantum experiments [15, 23].

However, despite several efforts, a way to generate a GHZ state of dimension $d > 2$ with more than $n = 4$ photons with perfect quality and finite count rates without additional resources [13] could not be found. This led Krenn and Gu to conjecture that it is not possible to achieve this physically (stated in graph theoretic terms in Conjecture 6). They have also formulated this question purely in graph theoretic terms and publicised it widely among graph theorists for a resolution [19]. We now formally state this problem in graph-theoretic terms and explain its equivalence in quantum photonic terms. For a high-level overview of how the experiments are converted to edge-coloured edge-weighted graphs, we refer the reader to the appendix of [6]. For the exact details, the reader can refer to [14, 10, 9, 13].

1.1 Graph theoretic preliminaries and notations

We first define some commonly used graph-theoretic terms. For a graph G, let $V(G), E(G)$ denote the set of vertices and edges, respectively. We use $\kappa(G)$ to denote the vertex connectivity of G. For $S \subseteq V(G)$, $G[S]$ denotes the induced subgraph of G on S. $\mathbb{N}, \mathbb{N}_0, \mathbb{C}$ denote the set of natural numbers, non-negative numbers and complex numbers, respectively. The cardinality of a set $\mathcal{S}$ is denoted by $|\mathcal{S}|$. For a positive integer r, $[r]$ denotes the set $\{1, 2 \ldots, r\}$. Given a multi-graph, its skeleton is its underlying simple graph. We do not consider self-loops in multi-graphs.

Usually, in an edge colouring, each edge is associated with a natural number. However, in such edge colourings, the edges are assumed to be monochromatic. But in the graphs corresponding to experiments, we are allowed to have bichromatic edges, i.e. one half coloured by a certain colour and the other half coloured by a different colour. For example, in the graph shown in Figure 1a, the simple edge between vertices 4 and 6 is a bichromatic edge. We develop some new notation to describe bichromatic edges.

Each edge of a multi-graph can be thought to be formed by two half-edges, i.e., an edge e between vertices u and v, consists of the half-edge starting from the vertex u to the middle of the edge e (hereafter referred to as the u-half-edge of e) and the half-edge starting from the vertex v to the middle of the edge e (hereafter referred to as the v-half-edge e). Thus, the edge set E of the multi-graph gives rise to the set of half-edges H, with $|H| = 2|E|$. For an edge e between vertices u and v, we may denote the v-half-edge of e by e_v and u-half edge of e by e_u. Consider the edge e between vertices 4 and 6 in Figure 1a. The 4-half edge of e (e_4) is of colour red, and the 6-half edge (e_6) is of colour green.

The type of edge colouring that we consider in this paper is more aptly called a *half-edge colouring.* It is a function from H to $\mathbb{N}_0$, say $c : H \to \mathbb{N}_0$. (Note that we use non-negative numbers to name the colours.) In other words, each half-edge gets a colour. An edge is called monochromatic if both its half-edges get the same colour (in which case we may use $c(e)$ to denote this colour); otherwise, it is called a bi-chromatic edge. In Figure 1a, the colour 0 is shown in red, and the colour 1 is shown in green. It is easy to see that $c(e_4) = 0$ and $c(e_6) = 1$ (recall that e is the simple edge between vertices 4 and 6). Consider the edge e' between vertices 1 and 6. As $c(e'_1) = c(e'_6) = 0$, e' is monochromatic and moreover, $c(e') = 0$. We then assign a weight $w(e) \in \mathbb{C}$ to each such coloured edge e. We denote the multi-graph G with the edge colouring c and edge weights $w(e)$ as G_c^w.

We call a subset P of edges in this edge-weighted edge-coloured graph a perfect matching if each vertex in the graph has exactly one edge in P incident on it.

▶ **Definition 1.** *The weight of a perfect matching P, $w(P)$ is the product of the weights of all its edges* $\prod\limits_{e \in P} w(e)$

▶ **Definition 2.** *The weight of an edge-coloured edge-weighted multi-graph G_c^w is the sum of the weights of all perfect matchings in G_c^w.*

A vertex colouring vc associates a colour i to each vertex in the graph for some $i \in \mathbb{N}$. We use $vc(v)$ to denote the colour of vertex v in the vertex colouring vc. A vertex colouring vc *filters out* a sub-graph $\mathcal{F}(G_c, vc)$ of G_c on $V(G_c)$ where for an edge $e \in E(G_c)$ between vertices u and v, $e \in E(\mathcal{F}(G_c, vc))$ if and only if $c(e_u) = vc(u)$ and $c(e_v) = vc(v)$. Filtering also extends to weighted graphs where the weight of each edge in $\mathcal{F}(G_c^w, vc)$ is the same as its weight in G_c^w. Let vc be a vertex colouring in which $1, 2, 3, 6$ are associated with the colour green and $4, 5$ are associated with the colour red. The filtering operation of vc on the edge-coloured graph G_c^w shown in Figure 1a is given in Figure 1b. A vertex colouring vc is defined to be feasible in G_c^w if $\mathcal{F}(G_c^w, vc)$ has at least one perfect matching. It is easy to see that each perfect matching P is part of $\mathcal{F}(G_c^w, vc)$ for a unique vertex colouring vc. Such a P is said to induce vc. It is interesting to notice that there is a partition of perfect matchings (not edges) based on the vertex colourings.

▶ **Definition 3.** *The weight of a vertex colouring vc in the multi-graph G_c^w is denoted by $w(G_c^w, vc)$ and is equal to the weight of the graph $\mathcal{F}(G_c^w, vc)$.*

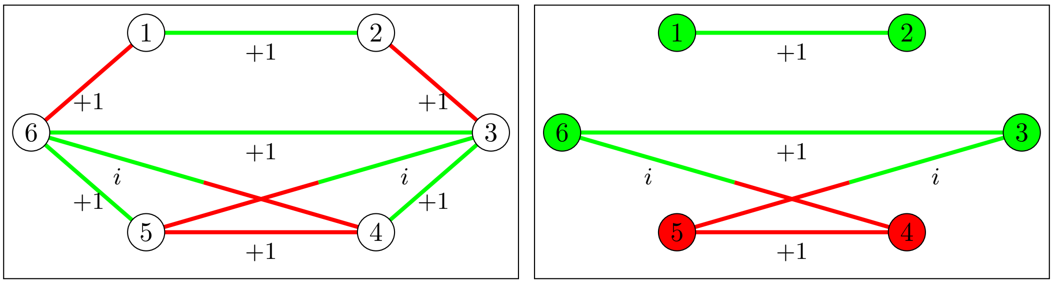
The weight of a vertex colouring, which is not feasible, is zero by default.

▶ **Definition 4.** *An edge-coloured edge-weighted graph is said to be GHZ, if:*
1. *All feasible monochromatic vertex colourings have a weight of* 1.
2. *All non-monochromatic vertex colourings have a weight of* 0.

An example of a GHZ graph is shown in Figure 1a.

▶ **Definition 5.** *The dimension of a GHZ graph G_c^w, $\mu(G, c, w)$ is the number of feasible monochromatic vertex colourings (having a weight of* 1*).*

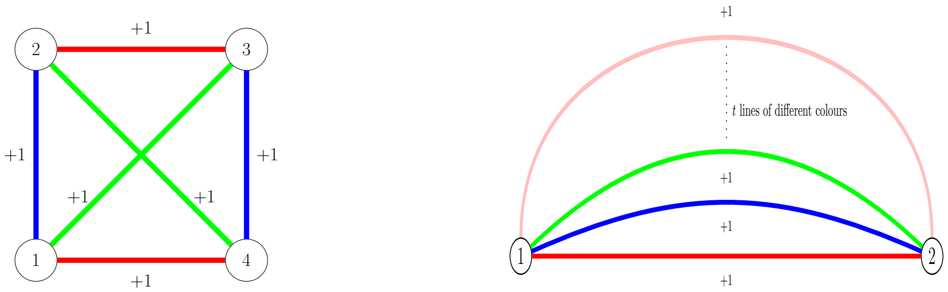
For a given multi-graph G (experimental set up), many possible edge-colourings (mode numbers of photons) and edge-weight (amplitude of photon pairs) assignments may lead it a GHZ graph (GHZ state). Finding a GHZ graph with n vertices and dimension d would



(a) An edge coloured edge-weighted graph G_c^w.

(b) $\mathcal{F}(G_c^w, vc)$, where $vc(i)$ is green for $i \in [1, 2, 3, 6]$ and red for $i \in [4, 5]$.

Figure 1 vc-Filtering of a graph.



(a) K_4 with 3 dimensions.

(b) K_2 with t dimensions.

Figure 2 GHZ graphs.

immediately lead to an experiment which result in a d-dimensional GHZ state with n particles. For each such GHZ graph, a dimension is achieved. The maximum dimension achieved over all possible GHZ graphs with the unweighted uncoloured simple graph G as their skeleton is known as the *matching index* of G, denoted by $\mu(G)$. In Figure 2b, we have an edge-coloured edge-weighted GHZ K_2 of dimension t. Note that t can be arbitrarily large. Therefore, $\mu(K_2) = \infty$. However, only two particles are involved; for a GHZ state to form, we need more than two particles. Therefore, such a construction will not give a GHZ state. We note that the matching index is defined for a simple graph G by taking the maximum over all possible multi-graphs with a skeleton G. For instance, the simple graph K_2 has only one edge. However, we considered all possible multi-graphs having a skeleton K_2 to define $\mu(K_2)$.

It is easy to see that if a graph has a perfect matching, it must contain an even number of vertices. So, we consider matching indices of graphs with even and at least 4 vertices for the rest of the manuscript. From Figure 2a, we know that $\mu(K_4) \geq 3$ and, despite the use of huge computational resources [15, 23, 20], this is the only (up to an isomorphism) known graph of the matching index at least 3. Any graph with a matching index of at least 3 and $n > 4$ vertices would lead to a new GHZ state of dimension at least 3 with $n > 4$ entangled particles. Motivated by this, this problem has been extensively promoted[19, 12]. Krenn and Gu conjectured that

▶ **Conjecture 6.** *If* $|V(G)| > 4$, *then* $\mu(G) \leq 2$

Several cash rewards were also announced for a resolution of this conjecture [12]. We note the following implications of resolving this conjecture

1. Finding a counterexample for this conjecture would uncover new peculiar quantum interference effects of a multi-photonic quantum system using which we can create new GHZ states

2. **a.** Proving this conjecture would immediately lead to new insights into resource theory in quantum optics
   b. Proving this conjecture for different graph classes would help us understand the properties of a counterexample and guide experimentalists in finding it. This is particularly important since huge computational efforts are going into finding such graphs [15, 23, 20].

A graph is matching covered if every edge of it is part of at least one perfect matching. If an edge e is not part of any perfect matching M, then we call the edge e to be redundant. By removing all redundant edges from the given graph G, we get its unique maximum matching covered sub-graph $mcg(G)$. Note that a colouring c and a weight assignment w of G induces a colouring and a weight assignment for every subgraph of G, respectively. When there is no scope for confusion, we use c, w itself to denote this induced colouring and weight assignment, respectively. It is easy to see that if c and w make G a GHZ graph, they also make $mcg(G)$ a GHZ graph and $\mu(G, c, w) = \mu(mcg(G), c, w)$. Therefore, $\mu(G) = \mu(mcg(G))$.

One can also notice that, if there are two edges, say e, e' between vertices u and v such that $c(e_u) = c(e'_u)$ and $c(e_v) = c(e'_v)$, then they can be replaced with an edge e'' such that $w(e'') = w(e) + w(e')$,$c(e''_u) = c(e_u)$ and $c(e''_v) = c(e_v)$. Such a reduction will retain the GHZ property and dimension of the graph. Therefore, in the rest of the manuscript, we only deal with such reduced graphs, i.e, between two vertices between vertices u and v and given $i, j \in [\mu(G)]$ there exists at most one edge e such that $c(e_u) = i$ and $c(e_v) = j$. We also note that, if an edge e has weight 0, it can be treated as if the edge were absent.

1.2 Related work

No destructive interference. The special case of all edges having a real positive weight corresponds to the case when there is no destructive interference. With this restriction, Krenn-Gu conjecture was resolved due to the following observation by Bogdanov [3].

▶ **Theorem 7.** *In a coloured multi-graph G_c with $|V(G)| > 4$, if there exist three monochromatic perfect matchings of different colours, then there must be a non-monochromatic perfect matching.*

Due to this result, when there is no destructive interference, every matching covered graph non-isomorphic to K_4 can achieve a maximum dimension of 1 or 2 and thus can be classified into Type 1 and Type 2 graphs(See [6] for detailed discussion). Chandran and Gajjala [6] gave a structural classification for Type 2 graphs. They further proved that for any half-edge colouring and edge weight assignment on a simple Type 2 unweighted uncoloured graph, a dimension of 3 or more can not be achieved! The computational aspects of the vertex colourings arising from these experiments were studied by Vardi and Zhang [25, 26]

Absence of bi-coloured edges. The problems get easier in the absence of bi-coloured edges and have opened up work in several directions. We list some of the known results in this direction. Cervera-Lierta et al. [5] used SAT solvers to prove that if the number of vertices is 6 or 8, the maximum dimension achievable is 2 or, at most, 3, respectively. Chandran and Gajjala [7] proved that the maximum dimension achievable for an $n > 4$ vertex graph is less than $\dfrac{n}{\sqrt{2}}$.

Unrestricted results. For the general case, the only known result is due to Mantey [18]. He proved the following theorem using the Gröbner basis.

▶ **Theorem 8.** *If $|V(G)| = 4$, then $\mu(G) \leq 3$. Moreover, if $\mu(G, c, w) = 3$, then between any pair of vertices in G_c^w, there is exactly one non-zero edge (and isomorphic to the coloured graph shown in Figure 2a).*

Surprisingly, there is no known analytical proof even for such *small* graphs. We encourage the reader to attempt to prove Theorem 8 to understand the difficulty arising due to multi-edges. One has to tune 54 variables which can be complex numbers (the number of possible edges when 3 colours are allowed) such that 81 equations (the number of possible vertex colourings when 3 colours are allowed) are satisfied, even for graphs as small as 4 vertex graphs.

1.3 Our results

We give the first results, which resolve the Krenn-Gu conjecture for a large class of graphs in the completely general setting, that is, when both bi-coloured edges and multi-edges are allowed. We prove that the Krenn-Gu conjecture is true for all graphs with vertex connectivity at most 2 in the full version.

▶ **Theorem 9.** *For a graph G, if $\kappa(G) \leq 2$, then $\mu(G) \leq 2$.*

Our next main contribution is a reduction technique, which implies Theorem 10. We explain and prove our reduction in Section 2. We introduce a scaling lemma in Section 1.4, which gives us an equivalent version of Krenn-Gu conjecture and which may turn out to be more useful in some situations.

▶ **Theorem 10.** *Given a graph G with $\kappa(G) \leq 3$ and $V(G) > 4$, there exists a graph G' with $|V(G')| \leq |V(G)| - 2$ and $\mu(G') \geq \mu(G)$.*

Due to Theorem 10, a minimal counter-example (a counter-example with the minimum number of vertices) to Krenn-Gu conjecture must be 4-connected. Using Theorem 10, we can resolve Krenn-Gu conjecture for some interesting graph classes like cubic graphs (that is, 3 regular graphs). We prove Theorem 11 and Theorem 12 in Section 2.3.

▶ **Theorem 11.** *If the maximum degree of a graph G is 3, Conjecture 6 is true.*

▶ **Theorem 12.** *If the minimum degree of a graph G is 3, then $\mu(G) \leq 3$*

1.4 Reformulation of Krenn-Gu conjecture

Recall that we denote the weight of the vertex colouring vc over a set of vertices $U \subseteq V(G)$ as $w(U, vc)$, which is the sum of weights of all perfect matching on $G[U]$ which induce the vertex colouring vc on U and we denote the monochromatic vertex colouring $vc : V \to \{i\}$ by $\mathbf{i}_V$.

For a graph G, let $U, U' \subseteq V(G)$. Let $vc : U \to \mathbf{N}$ and $vc' : U' \to \mathbf{N}$. If $vc(v) = vc'(v)$ for all $v \in U \cap U'$, we call vc, vc' to be compatible with each other. When vc, vc' are compatible, we define their union $[vc \cup vc'] : U \cup U' \to \mathbf{N}$ as follows: $[vc \cup vc'](v) = vc(v)$ for $v \in U$ and $[vc \cup vc'](v) = vc'(v)$ for $v \in U'$.

We broaden the definition of GHZ graphs to *g-GHZ* graphs. An edge-coloured edge-weighted graph G_c^w satisfying the following properties is defined to be g-GHZ

1. All feasible monochromatic vertex colourings have a non-zero weight (instead of necessarily being 1).
2. All non-monochromatic vertex colourings have a weight of 0.

Note that this generalization allows each of the monochromatic vertex colourings to have different weights. The dimension of a g-GHZ graph is the number of feasible monochromatic vertex colourings. For a graph G, the maximum dimension achievable over all possible g-GHZ colouring and weight assignments is its *generalized matching index* $\mu_g(G)$.

▶ **Conjecture 13.** *$\mu_g(K_4) = 3$ and for a graph G which is non-isomorphic to K_4, $\mu_g(G) \leq 2$.*

We prove that Conjecture 6 and Conjecture 13 are equivalent. Trivially, a counter-example to Conjecture 6 would immediately give a counter-example to Conjecture 13. We prove that any counter-example to Conjecture 13 would also yield a counter-example to Conjecture 6 in Lemma 14. This reformulation is more suitable for our proofs in the following sections.

▶ **Lemma 14** (Scaling lemma). *If there is a graph G_c^w, which is g-GHZ, then there is a graph $G_c^{w'}$, which is a GHZ graph with the same dimension.*

Proof. We denote the weight of the vertex colouring $w(\mathbf{i}_V, G)$ using $W(i)$. Note that by definition of g-GHZ graphs, the weight of a monochromatic colouring $W(i)$ is always non-zero. So, for each edge $e \in G_c$ whose half-edges are of colour i, j, we assign the weight

$$w'(e) = w(e)(W(i)W(j))^{-1/n}$$

Let M be a matching in G_c, which induces the vertex colouring vc_M. The weight of an edge $e \in M$ between vertices u, v will be,

$$w'(e) = w(e)(W(vc_M(u))W(vc_M(v)))^{-1/n}$$

As each vertex is incident by exactly one edge of the perfect matching M, the weight of M will be

$$w'(M) = w(M) \prod_{v \in V} W(vc_M(v))^{-1/n}$$

Since the weights of all perfect matchings which induce a vertex colouring, vc will increase by a factor of $\prod_{v \in V} W(vc(v))^{-1/n}$, the weight of the vertex colouring vc will be

$$w'(vc) = w(vc) \prod_{v \in V} W(vc(v))^{-1/n}$$

As $w(vc)$ is zero for all non-monochromatic vertex colourings, $w'(vc)$ will remain to be zero.

For a monochromatic vertex colouring $vc = \mathbf{i}_V$, we know that $vc(v) = i$ for all $v \in V$. Therefore, $w'(\mathbf{i}_V) = w(\mathbf{i}_V)((w(\mathbf{i}_V))^{-1/n})^n = 1$

Therefore, $G_c^{w'}$ is a GHZ graph. ◀

2 Reduction

We prove a stronger theorem than Theorem 10 as stated below.

▶ **Theorem 15.** *Let G be a multi-graph with a vertex cut S of size 3. Let V_1 and V_2 be a partition of $V(G) \setminus S$ such that V_1 and V_2 are non-empty and there are no edges between V_1 and V_2 in G. Moreover, let $|V_1|$ be odd and $|V_2|$ be even. There exists a graph G' such that $|V(G')| \leq |V_1| + 3 \leq |V(G)| - 2$ and $\mu(G') \geq \mu(G)$.*

Let w, c be a colouring and weight assignment of G for which $\mu(G, c, w) = \mu(G)$. Then G' would be a graph on the vertex set $V_1 \bigsqcup S$. The edge set, the edge weight function and the edge colouring of G' would be the same as in $G[V_1 \bigsqcup S]$ except for the edges with both endpoints inside S; We redefine the set of edges, edge-weight function, edge-colouring within S. Let c' and w' represent the edge-colouring and the edge-weight function of G'. We will show that $\mu(G') \geq \mu(G', c', w') = \mu(G, c, w) = \mu(G)$.

Let $V = V(G)$ and $S = \{u_1, u_2, u_3\}$. Note that any perfect matching of G, should match an odd number of vertices of S to V_1 (since $|S|$ is odd). Therefore the perfect matchings of G can be grouped into 4 types:

Type 0: Let $\mathcal{P}_0$ denote the set of all perfect matchings on $G[V_1 \cup S]$ in which all the three vertices of S are matched with vertices in V_1. Let $W_0(vc)$ denote the sum of weights of the perfect matchings from $\mathcal{P}_0$ that induce the vertex colouring vc on $V_1 \bigsqcup S$. Let $\mathcal{P}'_0$ denote the set of all perfect matchings on $G[V_2]$. Let $W'_0(vc)$ denote the sum of weights of the perfect matchings from $\mathcal{P}'_0$ that induce the vertex colouring vc on V_2. Type 0 perfect matchings of G are the perfect matchings that belong to $\mathcal{P}_0 \times \mathcal{P}'_0$. Clearly, the sum of the weights of Type 0 perfect matchings that induce the colouring vc equals $W_0(vc)W'_0(vc)$.

Type *i*: For $i \in \{1, 2, 3\}$, Let $\mathcal{P}_i$ denote the set of perfect matchings of $V_1 \bigsqcup \{u_i\}$, and let $\mathcal{P}'_i$ denote the set of perfect matchings of $V_2 \bigsqcup (S \setminus \{u_i\})$. Let $W_i(vc)$ denote the sum of weights of all perfect matchings from $\mathcal{P}_i$ that induce the vertex colouring vc on $V_1 \bigsqcup \{u_i\}$ and $W'_i(vc)$ denote the sum of weights of all perfect matchings from $\mathcal{P}'_i$ that induce the colouring vc on $V_2 \bigsqcup (S \setminus \{u_i\})$. Type i matchings of G are the perfect matchings that belong to $\mathcal{P}_i \times \mathcal{P}'_i$. Clearly, the sum of weights of Type i perfect matchings that induce the colouring vc equals $W_i(vc)W'_i(vc)$. From the above discussion, it is easy to see that

$$w(vc) = \sum_{i \in \{0,1,2,3\}} W_i(vc)W'_i(vc) \tag{1}$$

Recall that c_{V_2} denotes the monochromatic vertex colouring with the colour c on V_2. Let us partition $[\mu(G)]$ into $\mathcal{C}_1 \sqcup \mathcal{C}_2$ such that $c \in \mathcal{C}_1$, if and only if there exists some colouring c' on $V_1 \sqcup S$ such that $W'_0(c_{V_2})W_0(c') \neq 0$. The remaining colors from $[\mu(G)]$ belong to $\mathcal{C}_2$ (Note that this happens if $W'_0(c_{V_2}) = 0$ or if for all colourings c' on $V_1 \sqcup S$, $W_0(c') = 0$.). We will now prove Theorem 15 in two cases. When $\mathcal{C}_1 = \emptyset$ (Theorem 1) and $\mathcal{C}_1 \neq \emptyset$ (Theorem 2)

▶ **Theorem 1.** *Let G be a multi-graph with a vertex cut of size 3. If $\mathcal{C}_1 = \emptyset$ (as defined earlier), then $\mu(G) \leq \mu(K_4)$*

▶ **Theorem 2.** *Let G be a multi-graph with a vertex cut of size 3. If $\mathcal{C}_1 \neq \emptyset$ (as defined earlier), then there exists a graph G' such that $|V(G')| \leq |V_1| + 3 \leq |V(G)| - 2$ and $\mu(G') \geq \mu(G)$. Recall that V_1 and V_2 is a partition of $V(G) \setminus S$ such that V_1 and V_2 are non-empty and there are no edges between them. Moreover $|V_1|$ is odd and $|V_2|$ is even.*

2.1 Construction for Theorem 1

Let $V(G') = \{v_0, v_1, v_2, v_3\}$. The reader may mentally map the vertices v_1, v_2, v_3 to the vertices u_1, u_2, u_3 of the vertex cut S and v_0 to the set of vertices V_1. Note that G' is a multi-graph (without self loops) and each pair of vertices from $V(G')$ may have many edges between them. In fact, we would define $\mu(G)^2$ number of edges between each pair of vertices in $\{v_0, v_1, v_2, v_2\}$, one edge for each ordered pair in $[\mu(G)] \times [\mu(G)]$.

Let $i, j \in \{1, 2, 3\}$ with $i \neq j$. Let $(p, q) \in [\mu(G)] \times [\mu(G)]$. We will define an edge e for each such (p, q) between v_i and v_j. This edge e would be coloured such that the v_i-half edge of e (i.e., e_{v_i}) has colour p and the v_j-half edge of e (i.e., e_{v_j}) has colour q. The weight of the edge would be $w(e) = \sum_{c \in \mathcal{C}_2} w(c_{V_2} p_{u_i} q_{u_j})$ The reader may recall that $c_{V_2} p_{u_i} q_{u_j}$ represents the colouring in which V_2, u_i, u_j are coloured with c, p, q respectively. Its weight is the weight of the induced subgraph of G on $V_2 \cup \{u_i, u_j\}$ filtered out by the colouring $c_{V_2} p_{u_i} q_{u_j}$. In this case, $\mathcal{C}_1 = \emptyset$; so $\mathcal{C}_2 = [\mu(G)]$; and therefore there are $\mu(G)$ terms in the summation.

We will now consider the edges between v_0 and v_i, when $i \in \{1, 2, 3\}$. Let $(p, q) \in [\mu(G)] \times [\mu(G)]$. We will define an edge e for each such (p, q) between v_0 and v_i. This edge e would be coloured such that the v_0-half edge of e (i.e., e_{v_0}) has colour p and the v_i-half edge of e (i.e., e_{v_i}) has colour q. The weight of such an edge e is defined as $w(e) = w(p_{V_1} q_{u_i})$

2.2 Proof of construction for Theorem 1

Consider any vertex colouring $vc : V' \to \mathbf{N}$ on G'. We will prove that $w(V(G'), vc) = 0$, if vc is non-monochromatic and $w(V(G'), vc) = 1$, if vc is monochromatic.

Let the vertex colouring vc be $i_{v_0} j_{v_1} k_{v_2} l_{v_3}$. To find the weight of vc, we consider the subgraph of G' filtered out by vc. For instance, between the vertices v_2 and v_3, the colouring vc would filter exactly one edge, and such an edge would have the v_2-half edge of colour k and v_3-half edge of colour l. Clearly, this gives a graph which is isomorphic to K_4 (with some edges possibly getting a weight of zero). Observe that there are only three perfect matchings in K_4. So, it is now easy to find the weight of vc by enumeration.

As an example, consider the vertex colouring in which v_0, v_1 are coloured red and v_2, v_3 are coloured green and blue, respectively. Its filtering is shown in Figure 3. Its weight would be $w_1 w_1' + w_2 w_2' + w_3 w_3'$.

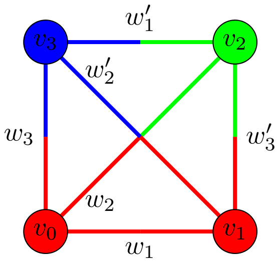


Figure 3 K_4 obtained by a filtering operation.

We will first compute the weight of perfect matching $\{\{v_0, v_1\}, \{v_2, v_3\}\}$ in vc. By substituting the edge weights from the construction, we get the weight of the perfect matching to be

$$= w(i_{V_1} j_{u_1}) \sum_{c \in [\mu(G)]} w(c_{V_2} k_{u_2} l_{u_3}) = \sum_{c \in [\mu(G)]} w(i_{V_1} j_{u_1}) w(c_{V_2} k_{u_2} l_{u_3})$$

As $w(i_{V_1} j_{u_1}) w(c_{V_2} k_{u_2} l_{u_3})$ is exactly the sum of weights of Type 1 perfect matchings for the vertex colouring $i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}$

$$= \sum_{c \in [\mu(G)]} W_1(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) W_1'(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) \tag{2}$$

Similarly, the weight of the weight of perfect matching $\{\{v_0, v_2\}, \{v_1, v_3\}\}$ in vc is

$$= \sum_{c\in[\mu(G)]} W_2(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) W_2'(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) \tag{3}$$

and the weight of the weight of perfect matching $\{\{v_0, v_3\}, \{v_1, v_2\}\}$ in vc is

$$= \sum_{c\in[\mu(G)]} W_3(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) W_3'(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) \tag{4}$$

As the weight of the vertex colouring vc' is the sum of these three perfect matchings, by adding the equations Equations (2)–(4), we get

$$w(vc') = \sum_{r\in\{1,2,3\}} \sum_{c\in[\mu(G)]} W_r(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) W_r'(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2})$$

By rearranging the summation and using Equation (1),

$$= \sum_{c\in[\mu(G)]} \sum_{r\in\{1,2,3\}} W_r(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) W_r'(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}) = \sum_{c\in[\mu(G)]} w(i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2})$$

First suppose that $vc = i_{v_0} j_{v_1} k_{v_2} l_{v_3}$ is non-monochromatic. Clearly, $i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}$ is also non-monochromatic for all $c \in [\mu(G)]$. Since the weight of all such colourings is zero, their sum is also zero. It now follows that $w(vc) = 0$

On the other hand, suppose $vc = i_{v_0} j_{v_1} k_{v_2} l_{v_3}$ is monochromatic, i.e., $i = j = k = l$. Clearly, if $c = i$, $i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}$ is also monochromatic and has weight 1. If $c \neq i$, $i_{V_1} j_{u_1} k_{u_2} l_{u_3} c_{V_2}$ is non-monochromatic and has weight 0. Since we take the sum over all c, it now follows that $w(vc) = 1$.

2.3 Applications of construction for Theorem 1

▶ **Corollary 16.** *If the minimum degree of a graph G is 3, then $\mu(G) \leq 3$*

Proof. Let u be a three-degree vertex, and x, y, z be its neighbours. Let $V_1 = \{u\}$, $S = \{x, y, z\}$ and $V_2 = V \setminus \{u, x, y, z\}$. Note that there are no Type 0 matchings as u can match with at most one vertex in S in any perfect matching. Therefore, $W_0(c)$ is zero for all $c \in [\mu(G)]$. It is now easy to see that $\mathcal{C}_1 = \emptyset$. Therefore, from Theorem 1, $\mu(G) \leq \mu(K_4)$. From Theorem 8, it is known that the matching index of graphs with 4 vertices is at most 3. Therefore, $\mu(G) \leq \mu(K_4) = 3$. ◀

▶ **Corollary 17.** *Conjecture 6 is true for all graphs whose maximum degree is at most* 3.

Proof. From Corollary 16, we know that $\mu(G) \leq 3$. Towards a contradiction, let $\mu(G) = 3$ for graph with $|V(G)|$ >4 and maximum degree 3. Therefore, there exists a colouring c and weight assignment w such that $\mu(G, c, w) = 3$. Let the three colours be $1, 2, 3$.

We first claim that between any pair of vertices, there is at most one non-zero edge incident on it. Suppose not. Then, there exist vertices u and x_1 with multiple non-zero edge between them. Since the maximum degree of the skeleton G is at most 3, there exists a vertex set (for instance, all neighbours of u if the degree is 3) $\{x_1, x_2, x_3\}$ which separates u from the $V - \{u, x_1, x_2, x_3\}$. Let $V_1 = \{u\}$, $S = \{x_1, x_2, x_3\}$ and $V_2 = V \setminus \{u, x_1, x_2, x_3\}$. Recall that $\mathcal{C}_1 = \emptyset$. By the construction from Section 2.1, the weights of edges between u and the vertices $\{x_1, x_2, x_3\}$ remain unchanged. Therefore, we obtain a graph with 4 vertices of dimension 3 such that a pair of vertices have multiple non-zero edges between them. But this is not possible from Theorem 8. Therefore, there are no multi-edges in G_c^w.

Since the matching index is 3, there are perfect matchings (of non-zero edges) of colours $1, 2, 3$. Therefore, from Theorem 7, there must be at least one non-monochromatic perfect matching M (of non-zero edges), say inducing the non-monochromatic vertex colouring vc. But there is exactly one non-zero edge of colours $1, 2, 3$ incident on u. Therefore, for any vertex colouring vc, there can be at most one perfect matching M' inducing vc. It now follows that $w'(M) = w'(vc) = 0$. Therefore, there must be an edge, say of colour 1, whose weight is zero; hence, the monochromatic vertex colouring of 1 must be zero. Contradiction.

Therefore, $\mu(G) \leq 2$ when $|V(G)| > 4$. ◀

2.4 Construction for Theorem 2

Recall that $S = \{u_1, u_2, u_3\}$ is a vertex cut separating V_1 from V_2 in the graph G, where $|V_1|$ is odd, $|V_2|$ is even. Assume that $\mathcal{C}_1 \neq \emptyset$. For this case, we will construct an edge-weighted, edge coloured multi-graph G' with $V(G') = V_1 \cup S$ and $\mu(G') \geq \mu(G)$. Since $|V_2| \geq 2$, $|V(G')| \leq |V(G)| - 2$.

If a pair of vertices is such that at least one of them lies in V_1, then the set of edges in G' between this pair of vertices is the same as those in G with the same weights and colours. Between the pairs of vertices with both vertices from S, we define one edge for each pair of colours in $[\mu(G)] \times [\mu(G)]$. Thus there would be $(\mu(G))^2$ edges between each pair, $\{u_i, u_j\}, i \neq j$.

We now describe how to assign weight to a coloured edge e between the vertices u_i and u_j, where $i < j$ and $i, j \in \{1, 2, 3\}$ such that the u_i-half of e is coloured p and the u_j-half of e is coloured q.

$$w'(e) = \sum_{c \in \mathcal{C}_2} W(c_{V_2} p_{u_i} q_{u_j}) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{W(c_{V_2} p_{u_i} q_{u_j})}{W(c_{V_2})} \tag{5}$$

▶ Remark 18. Recall that our plan is to remove V_2 and the edges incident on V_2 completely in order to get the reduced graph G'. This should be done without losing the information about the weights of monochromatic vertex colourings of V_2 to make sure that $\mu(G')$ does not become smaller than $\mu(G)$. The weight assignment is similar in spirit to the weight assignment done for the construction of Theorem 1. The expression here is more complicated in this case, because of the adjustments required to make it work: This will be clear when the reader goes through the proof carefully.

2.5 Proof of the construction for Theorem 2

Consider any vertex colouring $vc' : V_1 \cup S \to \mathbf{N}$ of G'. Let us denote vc' more explicitly, using the notation describe in Section 2.1, as $(\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3}$. Note that α here is the vertex colouring induced on V_1 by the vertex colouring vc', i, j, k are the colours of u_1, u_2, u_3 respectively under the vertex colouring vc'. (Note that we use the notation $(\alpha)_{V_1}$ to emphasize that this is not necessarily a monochromatic colouring of V_1, using a single colour named α; rather it can be any vertex colouring, monochromatic or non-monochromatic.) We will use the notation of w' to denote the weights of vertex colourings of G' and its subgraphs and w to denote the weights of vertex colourings of G and its subgraphs. For example, $w(i_{V_1} j_S)$ is the weight of the vertex colouring $i_{V_1} j_S$ with respect to the edge-set of G, whereas $w'(i_{V_1} j_S)$ denotes the weight of the same vertex colouring with respect to the edge-set of G'. Our intention is to prove that $w'((\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3}) = 0$, whenever $(\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3}$ is non-monochromatic and $w'((\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3}) \neq 0$, whenever $(\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3}$ is monochromatic, i.e. $i = j = k$ and α is a monochromatic vertex colouring of V_1 using colour i, i.e. $(\alpha)_{V_1} = i_{V_1}$. The reader may note that we are not insisting the weight to be equal to 1 in the monochromatic case; non-zero is sufficient since we can then use the Scaling Lemma (Lemma 14) to scale the weights of the monochromatic vertex colourings to 1, thus constructing an edge-weight function w' and edge-colouring c' of G' such that $\mu(G', c', w') = \mu(G)$ implying $\mu(G') \geq \mu(G)$.

Recall that $vc' = (\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3}$ filters out a simple graph from G' and the weight of vc' is the sum of weights of all the perfect matchings (PMs) in this filtered out simple graph. Perfect matchings (PMs) of this graph can be grouped into 4 categories: (1) Type $0'$: PMs containing none of the edges with both endpoints in S. (2) Type $1'$: PMs containing the edge (u_2, u_3) (3) Type $2'$: PMs containing the edge (u_1, u_3) (4) Type $3'$: PMs containing the edge (u_1, u_2) For $t = 0, 1, 2, 3$, we denote the total weight of Type t' PMs by W'_t. Clearly $w'(vc') = \sum_{0 \leq t \leq 3} W'_t$.

Now let us consider a corresponding vertex colouring of G, $vc = (\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3} c_{V_2}$, which is obtained by taking the same vertex colouring vc' for $V_1 \cup S$, and then extending it by the monochromatic vertex colouring c_{V_2} of V_2, using the colour c. Since we may use any colour $c \in [\mu(G)]$ to extend the vertex colouring vc' of $V_1 \cup S$ to a vertex colouring of $V_1 \cup S \cup V_2$, it is more appropriate to call the extended colouring $(\alpha)_{V_1} i_{u_1} j_{u_2} k_{u_3} c_{V_2}$, it is better to denote $vc(c)$, rather than just vc.

The weight of vc on G can also be decomposed into 4 terms corresponding to 4 different groups of perfect matchings of the subgraph filtered out by vc from G.

(1) Type 0: The PMs in which all the three vertices of S are matched to vertices in V_1.

(2) Type t for $t = 1, 2, 3$: The PMs in which only u_t is matched to some vertex of V_1, and the remaining two vertices of S are either matched to each other or to vertices of V_2.

The total weight of the perfect matchings (in the the subgraph of G filtered out by the vertex colouring $vc(c)$) of Type$t, 0 \leq t \leq 3$ will be denoted by $W_t(c)$ where $c \in [u(G)]$. Clearly $w(vc(c)) = \sum_{0 \leq t \leq 3} W_t(c)$. Now we show that W'_t can be expressed as a (weighted) sum of $W_t(c)$ over the colours $c \in [\mu(G)]$.

▶ **Observation 19.** *For* $t = 1, 2, 3 : W'_t = \sum_{c \in \mathcal{C}_2} W_t(c) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{W_t(c)}{w(c_{V_2})}$

Proof. Let us calculate the total weight W'_1 of Type 1 perfect matchings of the vertex colouring $\alpha_{V_1} i_{u_1} j_{v_2} k_{v_3}$. (The case when $t = 2, 3$ is similar.) Clearly these PMs are obtained by adding the edge (u_2, u_3) of colour (j, k) to each perfect matching of the induced subgraph on $V_1 \cup \{u_1\}$ (after filtering out by the vertex colouring $\alpha_{V_1} i_{u_1}$). Therefore the total weight of these PMs can be written as $W'_1 = w'(\alpha_{V_1} i_{u_1}) w'(e)$ where e is the edge between u_2 and u_3 of colour (i, j), that is, the edge e with u_2-half of e coloured j and u_3-half of e coloured k. Now substituting for $w'(e)$, the right hand side of equation 5, we get

$$W'_1 = w'(\alpha_{V_1} i_{u_1}) \left(\sum_{c \in \mathcal{C}_2} w(c_{V_2} j_{u_2} k_{u_3}) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{w(c_{V_2} j_{u_2} k_{u_3})}{w(c_{V_2})} \right) \tag{6}$$

Note that $w'((\alpha)_{V_1} i_{u_1}) = w((\alpha)_{V_1} i_{u_1})$ since in the induced subgraph on $V_1 \cup \{u_1\}$ the edge set, weights and colour are same for both G and G'. It follows that,

$$W'_1 = \sum_{c \in \mathcal{C}_2} w((\alpha)_{V_1} i_{u_1})) w(c_{V_2} j_{u_2} k_{u_3}) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{w((\alpha)_{V_1} i_{u_1})) w(c_{V_2} j_{u_2} k_{u_3})}{w(c_{V_2})} \tag{7}$$

Noting that $w(\alpha_{V_1} i_{u_1}) w(c_{V_2} j_{u_2} k_{u_3}) = w(\alpha_{V_1} i_{u_1} j_{u_2} k_{u_3} c_{V_2}) = W_1(c)$ we get,

$$W'_1 = \sum_{c \in \mathcal{C}_2} w(\alpha_{V_1} i_{u_1} c_{V_2} j_{u_2} k_{u_3}) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{w(\alpha_{V_1} i_{u_1} c_{V_2} j_{u_2} k_{u_3})}{w(c_{V_2})} = \sum_{c \in \mathcal{C}_2} W_1(c) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{W_1(c)}{w(c_{V_2})}$$

Similar arguments allow us establish the required result for $t = 2, 3$ also. ◀

▶ **Observation 20.** $w(vc') = \sum_{c \in \mathcal{C}_2} w(vc(c)) + \dfrac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \dfrac{w(vc(c))}{w(c_{V_2})}$

Proof. Now $w'(vc') = W_0' + W_1' + W_2' + W_3'$

$= W_0' + \sum_{c \in \mathcal{C}_2} (W_1(c) + W_2(c) + W_3(c)) + \dfrac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \dfrac{W_1(c) + W_2(c) + W_3(c)}{w(c_{V_2})}$

Recall that for any colour $c \in [\mu(G)]$, $w(vc(c)) = W_0(c) + W_1(c) + W_2(c) + W_3(c)$. Note that for colours $c \in \mathcal{C}_2$, $W_0(c) = W_0'.w(c_{V_2})$; and therefore $W_1(c) + W_2(c) + W_3(c) = w(vc(c))$, the weight of the vertex colouring $\alpha_{V_1} i_{u_1} j_{u_2} k_{u_3} c_{V_2}$. Now W_0' can be trivially rewritten as $\dfrac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \dfrac{W_0'.w(c_{V_2})}{w(c_{V_2})}$. Since $W_0'.w(c_{V_2}) = W_0(c)$, this expression can be rewritten as $\dfrac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \dfrac{W_0(c)}{w(c_{V_2})}$.

So we can combine the terms of this expression, term by term with the terms inside the sum over $c \in \mathcal{C}_1$ and rewrite the expression as follows:

$$= \sum_{c \in \mathcal{C}_2} w(vc(c)) + \frac{1}{|\mathcal{C}_1|} \sum_{c \in \mathcal{C}_1} \frac{W_0(c) + W_1(c) + W_2(c) + W_3(c)}{w(c_{V_2})} \tag{8}$$

Since for colours $c \in \mathcal{C}_1$, $W_0(c) + W_1(c) + W_2(c) + W_3(c) = w(vc(c))$ we get the required result. ◀

If vc' is non-monochromatic, then $vc(c)$ is also non-monochromatic for any $c \in [\mu(G)]$. Therefore, $w(vc(c)) = 0$ for all c and hence $w'(vc') = 0$ from Observation 20.

If vc' is monochromatic, say of colour i, $vc(c)$ will be monochromatic if and only if $c = i$. Therefore, if $i \in \mathcal{C}_2$, then $w(vc') = w(vc(i)) = 1$. Similarly, if $i \in \mathcal{C}_1$, then $w'(vc') = \dfrac{1}{|\mathcal{C}_1| w(\mathbf{i}_{V_2})}$ from Observation 20. In both the cases this will be non-zero as required.

The weights can now be readjusted by the Scaling Lemma (Lemma 14) to get a GHZ graph.

2.6 Limitations of our reduction

A careful reader might observe that the difficulty in extending our reduction technique to cuts of larger size comes from the case when two newly introduced edges are part of the same perfect matching. For instance, for the case when there is a 4 vertex cut $\{u_1, u_2, u_3, u_4\}$ separating V_1, V_2 (both of even size) in G, one could try to extend our ideas and capture the weights from $V_2 \cup \{u_3, u_4\}$ and $V_2 \cup \{u_1, u_2\}$ on the edges $(v_3, v_4), (v_1, v_2)$ of G', respectively. However, this would create some extra terms in G' due to the perfect matchings in which (v_1, v_2) and (v_3, v_4) are contained. Such terms could destroy the GHZ property of G'. We believe that finding a way to bypass this difficulty and finding a more general reduction will resolve Krenn and Gu's conjecture for all graphs.

References

1 Alain Aspect, John F. Clauser, and Anton Zeilinger. The nobel prize in physics 2022. bit.ly/3RZmMYg. Accessed: 14-02-2023.

2 J. S. Bell. On the einstein podolsky rosen paradox. *Physics Physique Fizika*, 1:195–200, November 1964. doi:10.1103/PhysicsPhysiqueFizika.1.195.

3 Ilya Bogdanov. Solution to graphs with only disjoint perfect matchings. bit.ly/3x8hUGQ. Accessed: 09-02-2023.

4 Dik Bouwmeester, Jian-Wei Pan, Matthew Daniell, Harald Weinfurter, and Anton Zeilinger. Observation of three-photon greenberger-horne-zeilinger entanglement. *Phys. Rev. Lett.*, 82:1345–1349, February 1999. doi:10.1103/PhysRevLett.82.1345.

5 Alba Cervera-Lierta, Mario Krenn, and Alán Aspuru-Guzik. Design of quantum optical experiments with logic artificial intelligence. *CoRR*, abs/2109.13273, 2021. arXiv:2109.13273.

6 L. Sunil Chandran and Rishikesh Gajjala. Perfect matchings and quantum physics: Progress on krenn's conjecture. *CoRR*, abs/2202.05562, 2022. arXiv:2202.05562.

7 L.Sunil Chandran and Rishikesh Gajjala. Graph-theoretic insights on the constructability of complex entangled states. *CoRR*, abs/2304.06407, 2023. doi:10.48550/arXiv.2304.06407.

8 Daniel M. Greenberger, Michael A. Horne, and Anton Zeilinger. *Going Beyond Bell's Theorem*, pages 69–72. Springer Netherlands, Dordrecht, 1989. doi:10.1007/978-94-017-0849-4_10.

9 Xuemei Gu, Lijun Chen, Anton Zeilinger, and Mario Krenn. Quantum experiments and graphs. iii. high-dimensional and multiparticle entanglement. *Physical Review A*, 99, March 2019. doi:10.1103/PhysRevA.99.032338.

10 Xuemei Gu, Manuel Erhard, Anton Zeilinger, and Mario Krenn. Quantum experiments and graphs ii: Quantum interference, computation, and state generation. *Proceedings of the National Academy of Sciences*, 116(10):4147–4155, 2019. doi:10.1073/pnas.1815884116.

11 Xuemei Gu and Mario Krenn. Compact greenberger-horne-zeilinger state generation via frequency combs and graph theory. *Frontiers of Physics*, 15(6), November 2020. doi:10.1007/s11467-020-1028-7.

12 Mario Krenn. Inherited vertex coloring of graphs. bit.ly/40GJXL3. Accessed: 09-02-2023.

13 Mario Krenn, Xuemei Gu, and Daniel Soltész. Questions on the structure of perfect matchings inspired by quantum physics. *arXiv preprint arXiv:1902.06023*, 2019.

14 Mario Krenn, Xuemei Gu, and Anton Zeilinger. Quantum experiments and graphs: Multiparty states as coherent superpositions of perfect matchings. *Physical review letters*, 119 24:240403, 2017.

15 Mario Krenn, Jakob S. Kottmann, Nora Tischler, and Alán Aspuru-Guzik. Conceptual understanding through efficient automated design of quantum optical experiments. *Phys. Rev. X*, 11:031044, August 2021. doi:10.1103/PhysRevX.11.031044.

16 Jay Lawrence. Rotational covariance and greenberger-horne-zeilinger theorems for three or more particles of any dimension. *Phys. Rev. A*, 89:012105, January 2014. doi:10.1103/PhysRevA.89.012105.

17 Jay Lawrence. Mermin inequalities for perfect correlations in many-qutrit systems. *Phys. Rev. A*, 95:042123, April 2017. doi:10.1103/PhysRevA.95.042123.

18 Kevin Mantey. Krenn-gu conjecture is true for graphs with four vertices. http://tinyurl.com/4e5zjvpx. Accessed: 14-02-2024.

19 Dustin Mixon. A graph colouring problem from quantum physics with prizes! bit.ly/3Xk5KFm. Accessed: 09-02-2023.

20 Aaron Neugebauer. Rainbow matchings in color-spanned graphs. *Bachelor Thesis, Universität Würzburg*, 2022. Accessed: 09-02-2023. URL: bit.ly/40CJCsV.

21 Jian-Wei Pan, Dik Bouwmeester, Matthew Daniell, Harald Weinfurter, and Anton Zeilinger. Experimental test of quantum nonlocality in three-photon greenberger–horne–zeilinger entanglement. *Nature*, 403(6769):515–519, February 2000. doi:10.1038/35000514.

22 Matej Pivoluska, Marcus Huber, and Mehul Malik. Layered quantum key distribution. *Phys. Rev. A*, 97:032312, March 2018. doi:10.1103/PhysRevA.97.032312.

23 Carlos Ruiz-Gonzalez, Sören Arlt, Jan Petermann, Sharareh Sayyad, Tareq Jaouni, Ebrahim Karimi, Nora Tischler, Xuemei Gu, and Mario Krenn. Digital discovery of 100 diverse quantum experiments with pytheus. *Quantum*, 7:1204, 2023. doi:10.22331/Q-2023-12-12-1204.

24 Junghee Ryu, Changhyoup Lee, Marek Żukowski, and Jinhyoung Lee. Greenberger-horne-zeilinger theorem for n qudits. *Phys. Rev. A*, 88:042101, October 2013. doi:10.1103/PhysRevA.88.042101.

25 Moshe Y. Vardi and Zhiwei Zhang. Quantum-inspired perfect matching under vertex-color constraints. *CoRR*, abs/2209.13063, 2022. doi:10.48550/arXiv.2209.13063.

26 Moshe Y. Vardi and Zhiwei Zhang. Solving quantum-inspired perfect matching problems via tutte-theorem-based hybrid boolean constraints. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 2039–2048. ijcai.org, 2023. doi:10.24963/IJCAI.2023/227.

27 Xi-Lin Wang, Luo-Kan Chen, W. Li, H.-L. Huang, C. Liu, C. Chen, Y.-H. Luo, Z.-E. Su, D. Wu, Z.-D. Li, H. Lu, Y. Hu, X. Jiang, C.-Z. Peng, L. Li, N.-L. Liu, Yu-Ao Chen, Chao-Yang Lu, and Jian-Wei Pan. Experimental ten-photon entanglement. *Phys. Rev. Lett.*, 117:210502, November 2016. doi:10.1103/PhysRevLett.117.210502.

28 Han-Sen Zhong, Yuan Li, Wei Li, Li-Chao Peng, Zu-En Su, Yi Hu, Yu-Ming He, Xing Ding, Weijun Zhang, Hao Li, Lu Zhang, Zhen Wang, Lixing You, Xi-Lin Wang, Xiao Jiang, Li Li, Yu-Ao Chen, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. 12-photon entanglement and scalable scattershot boson sampling with optimal entangled-photon pairs from parametric down-conversion. *Phys. Rev. Lett.*, 121:250505, December 2018. doi:10.1103/PhysRevLett.121.250505.

Applications of Littlestone Dimension to Query Learning and to Compression

Hunter Chase
Department of Mathematics, Statistics, and Computer Science,
University of Illinois at Chicago, IL, USA

James Frietag
Department of Mathematics, Statistics, and Computer Science,
University of Illinois at Chicago, IL, USA

Lev Reyzin
Department of Mathematics, Statistics, and Computer Science,
University of Illinois at Chicago, IL, USA

Abstract

In this paper we give several applications of Littlestone dimension. The first is to the model of Angluin and Dohrn [1], where we extend their results for learning by equivalence queries with random counterexamples. Second, we extend that model to infinite concept classes with an additional source of randomness. Third, we give improved results on the relationship of Littlestone dimension to classes with extended d-compression schemes, proving the analog of a conjecture of Floyd and Warmuth [4] for Littlestone dimension.

2012 ACM Subject Classification Theory of computation → Query learning; Theory of computation → Randomness, geometry and discrete structures; Mathematics of computing → Combinatoric problems

Keywords and phrases compression scheme, query learning, random queries, Littlestone dimension

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.42

Related Version *Full Version*: `https://arxiv.org/abs/2310.04812`

Funding This research was supported in part by award ECCS-2217023 from the National Science Foundation and National Science Foundation CAREER award 1945251.

Acknowledgements The authors wish to thank Shai Ben-David for comments on an early draft.

1 Introduction

In query learning, a learner attempts to identify an unknown concept from a collection via a series of data requests called queries. Typically, algorithms designed for learning in this setting attempt to bound the number of required queries to identify the target concept in the worst-case scenario. If one imagines the queries of the learner being answered by a teacher, the usual setup imagines the teacher answering queries in an adversarial manner, with minimally informative answers. Alternatively, for a given algorithm, the bounds for the traditional model are on the *worst-case answers* over *all potential targets*. In variations of the model, one of these two factors is usually modified.

For instance, Kumar, Chen, and Singla [7] study the case in which the answers are assumed to be maximally informative in a certain sense. In this manuscript, we first work in the setup originating with Angluin and Dohrn [1], where we assume that the answers to the queries are randomly selected with respect to some fixed probability distribution.

Consider a concept class $\mathcal{C} = \{C_1, \dots, C_n\}$, subsets of a fixed set X. Fix a target concept $A \in \mathcal{C}$. An *equivalence query* consists of the learner submitting a hypothesis $B \in \mathcal{C}$ to a teacher, who either returns *yes* if $A = B$, or a counterexample $x \in A \triangle B$. In the former case, the learner has learned A, and in the latter case, the learner uses the new information to update and submit a new hypothesis.

Angluin and Dohrn [1] fix a probability distribution μ on X and assume that the teacher selects the counterexamples randomly with respect to μ restricted to $A \triangle B$. They show that for a concept class $\mathcal{C}$ of size n, there is an algorithm in which the expected number of queries to learn any concept is at most $\log_2(n)$. It is natural to wonder whether there is a combinatorial notion of dimension which can be used to bound the expected number of queries independent of the size of the class - perhaps even in infinite classes. In fact, Angluin and Dohrn [1] (Theorem 25) already consider this and show that the VC-dimension of the concept class is a lower bound on the number of expected queries. On the other hand, Angluin and Dohrn [1] (Theorem 26), using an example of [9], show that the VC-dimension *cannot* provide an upper bound for the number of queries.

The motivation for bounds depending on some notion of dimension rather than the number of concepts is two-fold:

- Many combinatorial notions of dimension (e.g. Littlestone or VC) of a class $\mathcal{C}$ can be small while $|\mathcal{C}|$ is large.
- Investigating this model of learning in settings where $\mathcal{C}$ is an infinite class will require methods and bounds that do not use $|\mathcal{C}|$.

Roughly speaking, the *Littlestone dimension* (or Ldim) [9] of a concept class $\mathcal{C}$ over domain X is the maximal depth of a complete binary decision tree T such that T's nodes are associated with elements of X and T's edges are associated with binary labels such that each root-to-leaf path in T agrees in the labeling of its respective elements with some concept in $\mathcal{C}$. If we require all nodes of T at the same depth to be associated with the same element of X, this yields the definition of VC dimension.

We show that the Littlestone dimension provides such an upper bound; we give an algorithm that yields a bound that is linear in the Littlestone dimension for the expected number of queries needed to learn any concept. In Section 2 we establish the bounds for finite concept classes $\mathcal{C}$.

In Section 3 we give a specific example that shows finite Littlestone dimension of an infinite class $\mathcal{C}$ is not sufficient to guarantee the learnability of the class in the model of Angluin and Dohrn [1]. That is, we show the expected number of queries is impossible to bound over all target concepts, even in very simple infinite classes. Suppose that the target concept is itself selected randomly with respect to some (perhaps unrelated to the feedback mechanism) probability distribution. In this case, we give an algorithm so that the expected number of queries (over both sources of randomness) is at most $\tilde{O}(d)$ where d is the Littlestone dimension of the class $\mathcal{C}$. This result uses the bounds developed in Section 2 in an essential way, in particular by using the finite class's Littlestone dimension instead of its size.

In Section 4, we give another application of Littlestone dimension - to compression schemes, which answers a question of Johnson and Laskowski [6] on d-compression with b extra bits, a notion originating with Floyd and Warmuth [4]. The existence of a d-compression is closely related to various notions of learning; d-compressibility of a class $\mathcal{C}$ implies the class has VC-dimension at most d. A famous conjecture of Floyd and Warmuth [4] asks if every VC-class has a d-compression where d is the VC-dimension.[1] Our result in Section 4 proves a strong version of the conjecture for Littlestone dimension. In Section 4 we also explain some of the many variants of this problem which have been previously solved.

2 Random counterexamples and EQ-learning

In this section, we essentially work in the setting of Angluin and Dohrn [1] with slightly different notation. Throughout this section, let X be a finite set, let $\mathcal{C}$ be a set system on X, and let μ be a probability measure on X. For $A, B \in \mathcal{C}$, let

$$\Delta(A, B) = \{x \in X \mid A(x) \neq B(x)\}$$

denote the symmetric difference of A and B.

▶ **Definition 2.1.** *We denote, by $\mathcal{C}_{\bar{x}=\bar{i}}$ for $\bar{x} \in X^n$ and $\bar{i} \in \{0,1\}^n$, the set system $\{A \in \mathcal{C} \mid A(x_j) = i_j, j = 1, \ldots, n\}$. For $A \in \mathcal{C}$ and $a \in X$, we let*

$$u(A, a) = \mathrm{Ldim}(\mathcal{C}) - \mathrm{Ldim}(\mathcal{C}_{a=A(a)}).$$

For any $a \in X$, either $\mathcal{C}_{a=1}$ or $\mathcal{C}_{a=0}$ has Littlestone dimension strictly less than that of $\mathcal{C}$ and so:

▶ **Lemma 2.2.** *For $A, B \in \mathcal{C}$ and $a \in X$ with $A(a) \neq B(a)$,*

$$u(A, a) + u(B, a) \geq 1.$$

Next, we define a directed graph that is similar to the *elimination graph* of Angluin and Dohrn [1].

▶ **Definition 2.3.** *We define the* thicket query graph *$G_{TQ}(\mathcal{C}, \mu)$ to be the weighted directed graph on vertex set $\mathcal{C}$ such that the directed edge from A to B has weight $d(A, B)$ equal to the expected value of* $\mathrm{Ldim}(\mathcal{C}) - \mathrm{Ldim}(\mathcal{C}_{x=B(x)})$ *over $x \in \Delta(A, B)$ with respect to the distribution $\mu|_{\Delta(A,B)}$.* [2]

▶ **Definition 2.4.** *The* query rank *of $A \in \mathcal{C}$ is defined as:* $\inf_{B \in \mathcal{C}}(d(A, B))$.

▶ **Lemma 2.5.** *For any $A \neq B \in \mathcal{C}$, $d(A, B) + d(B, A) \geq 1$.*

Proof. Noting that $\Delta(A, B) = \Delta(B, A)$, and using Lemma 2.2:

$$\begin{aligned} d(A, B) + d(B, A) &= \sum_{a \in \Delta(A,B)} \frac{\mu(a)}{\mu(\Delta(A, B))}(u(A, a) + u(B, a)) \\ &\geq \sum_{a \in \Delta(A,B)} \frac{\mu(a)}{\mu(\Delta(A, B))} \\ &= 1. \end{aligned}$$ ◀

▶ **Definition 2.6** (Angluin and Dohrn [1], Definition 14)**.** *Let G be a weighted directed graph and $l \in \mathbb{N}$, $l > 1$. A* deficient l-cycle *in G is a sequence $v_0, \ldots v_{l-1}$ of distinct vertices such that for all $i \in [l]$, $d(v_i, v_{(i+1) (\mod l)}) \leq \frac{1}{2}$ with strict inequality for at least one $i \in [l]$.*

[1] Resolving whether there is an $O(d)$ compression has a reward of 600 dollars [12].

[2] Here one should think of the query by the learner as being A, and the actual hypothesis being B. The teacher samples from $\Delta(A, B)$, and the learner now knows the value of the hypothesis on x.

The next result is similar to Theorems 16 (the case $l = 3$) and Theorem 17 (the case $l > 3$) of Angluin and Dohrn [1], but our proof is rather different (note that the case $l = 2$ follows easily from Lemma 2.5).

▶ **Theorem 2.7.** *The thicket query graph $G_{TQ}(\mathcal{C}, \mu)$ has no degenerate l-cycles for $l \geq 2$.*

The analogue of Theorem 16 of Angluin and Dohrn [1] can be adapted in a very similar manner to the technique employed by them. However, the analogue of the proof of Theorem 17 of Angluin and Dohrn [1] falls apart in our context; the reason is that Lemma 2.2 is analogous to their Lemma 6 (and Lemma 2.5 is analogous to their Lemma 13), but our lemmas involve inequalities instead of equations. The inductive technique of Angluin and Dohrn [1, Theorem 17] is to shorten degenerate cycles by considering the weights of a particular edge in the elimination graph along with the weight of the edge in the opposite direction. Since one of those weights being large forces the other to be small (by the *equalities* of their lemmas), the induction naturally separates into two useful cases. In our thicket query graph, things are much less tightly constrained - one weight of an edge being large does not force the weight of the edge in the opposite direction to be small. However, the technique employed in our proof seems to be flexible enough to adapt to prove Theorems 16 and 17 of Angluin and Dohrn [1].

Proof. Suppose the vertices in the degenerate l-cycle are $A_0, \dots, A_{l-1}$. By the definition of degenerate cycles and $d(-,-)$, we have, for each $i \in \mathbb{Z}/l\mathbb{Z}$, that

$$\sum_{a \in \Delta(A_i, A_{i+1})} \frac{\mu(a)}{\mu(\Delta(A_i, A_{i+1}))} u(A_i, a) \leq \frac{1}{2}.$$

Clearing the denominator we have

$$\sum_{a \in \Delta(A_i, A_{i+1})} \mu(a) u(A_i, a) \leq \frac{1}{2} \mu(\Delta(A_i, A_{i+1})). \tag{2.1}$$

Note that throughout this argument, the coefficients are being calculated modulo l. Notice that for at least one value of i, the inequality in 2.1 must be strict.

Let G, H be a partition of

$$\mathcal{X} = \{A_1, \dots, A_l\}.$$

Now define

$$D(G,H) := \{a \in X \mid \forall A_1, B_1 \in G, \forall A_2, B_2 \in H, A_1(a) = B_1(a), A_2(a) = B_2(a), A_1(a) \neq A_2(a)\}.$$

The following fact follows from the definition of $\Delta(A, B)$ and $D(-,-)$.

▶ **Fact 2.8.** *The set $\Delta(A_i, A_{i+1})$ is the disjoint union, over all partitions of $\mathcal{X}$ into two pieces G, H such that $A_i \in G$ and $A_{i+1} \in H$ of the sets $D(G, H)$.*

Now, take the sum of the inequalities 2.1 as i ranges from 1 to l. On the LHS of the resulting sum, we obtain

$$\sum_{i=1}^{l} \left(\sum_{G,H \text{ a partition of } \mathcal{X},\, A_i \in G, A_{i+1} \in H} \left(\sum_{a \in D(G,H)} \mu(a) u(A_i, a) \right) \right).$$

On the RHS of the resulting sum, we obtain

$$\frac{1}{2}\sum_{i=1}^{l}\left(\sum_{G,H\text{ a partition of }\mathcal{X},\, A_i\in G, A_{i+1}\in H}\left(\sum_{a\in D(G,H)}\mu(a)\right)\right).$$

Given a partition G, H of $\{A_1, \dots, A_l\}$ we note that the term $D(G,H) = D(H,G)$ appears exactly once as an element of the above sum for a fixed value of i exactly when $A_i \in G$ and $A_{i+1} \in H$ or $A_i \in H$ and $A_{i+1} \in G$.

Consider the partition G, H of $\mathcal{X}$. Suppose that $A_j, A_{j+1}, \dots, A_k$ is a block of elements each contained in G, and that A_{j-1}, A_{k+1} are in H. Now consider the terms $i = j - 1$ and $i = k$ of the above sums (each of which where $D(G,H)$ appears).

On the left hand side, we have $\sum_{a\in D(G,H)} \mu(a)u(A_{j-1}, a))$ and $\sum_{a\in D(G,H)} \mu(a)u(A_k, a))$. Note that for $a \in D(G,H)$, we have $a \in \Delta(A_{j-1}, A_k)$. So, by Lemma 2.2, we have

$$\sum_{a\in D(G,H)} \mu(a)u(A_{j-1}, a) + \sum_{a\in D(G,H)} \mu(a)u(A_k, a) \geq \sum_{a\in D(G,H)} \mu(a).$$

On the RHS, we have

$$\frac{1}{2}\left(\sum_{a\in D(G,H)} \mu(a) + \sum_{a\in D(G,H)} \mu(a)\right) = \sum_{a\in D(G,H)} \mu(a).$$

For each G, H a partition of X, the terms appearing in the above sum occur in pairs as above by Fact 2.8, and so, we have the LHS is at least as large as the RHS of the sum of inequalities 2.1, which is impossible since one of the inequalities must have been strict by our degenerate cycle. ◀

▶ **Theorem 2.9.** *There is at least one element $A \in \mathcal{C}$ with query rank at least $\frac{1}{2}$.*

Proof. If not, then for every element $A \in \mathcal{C}$, there is some element $B \in \mathcal{C}$ such that $d(A, B) < \frac{1}{2}$. So, pick, for each $A \in \mathcal{C}$, an element $f(A)$ such that $d(A, f(A)) < \frac{1}{2}$. Now, fix $A \in \mathcal{C}$ and consider the sequence of elements of $\mathcal{C}$ given by $(f^i(A))$; since $\mathcal{C}$ is finite, at some point the sequence repeats itself. So, take a list of elements $B, f(B), \dots, f^n(B) = B$. By construction, this yields a bad cycle, contradicting Theorem 2.7. ◀

2.1 The thicket max-min algorithm

In this subsection we show how to use the lower bound on query rank proved in Theorem 2.9 to give an algorithm that yields the correct concept in linearly (in the Littlestone dimension) many queries from $\mathcal{C}$. The approach is fairly straightforward – essentially the learner repeatedly queries the highest query rank concept. The approach is similar to that taken in Angluin and Dohrn [1, Section 5] but with query rank in place of their notion of *informative*.

Now we informally describe the thicket max-min-algorithm. At stage i, the learner is given information of a concept class $\mathcal{C}_i$. The learner picks the query

$$A = \text{argmax}_{A\in\mathcal{C}_i}\left(\min_{B\in\mathcal{C}_i} d_{\mathcal{C}_i}(A, B)\right).$$

The algorithm halts if the learner has picked the actual concept C. If not, the teacher returns a random element $a_i \in \Delta(A, C)$ at which point the learner knows the value of $C(a_i)$. Then

$$\mathcal{C}_{i+1} = (\mathcal{C}_i)_{a_i=C(a_i)}.$$

Let $T(\mathcal{C})$ be the expected number of queries before the learner correctly identifies the target concept.

▶ **Theorem 2.10.** *The expected number of queries to learn a concept in a class $\mathcal{C}$ is less than or equal to* $2\,\mathrm{Ldim}(\mathcal{C})$.

Proof. The expected drop in the Littlestone dimension of the concept class induced by any query before the algorithm terminates is at least $1/2$ by Theorem 2.9; so the probability that the drop in the Littlestone dimension is positive is at least $1/2$ for any given query. So, from $2n$ queries, one expects at least n drops in Littlestone dimension, at which point the class is learned. ◀

3 Equivalence queries with random counterexamples and random targets

Let $\mathcal{C}$ consist the collection of intervals $\left\{\left(\frac{1}{n+1}, \frac{1}{n}\right) \mid n \in \mathbb{N}\right\}$ with μ the Lebesgue measure on the unit interval. This concept class has Littlestone dimension one since any two concepts are disjoint. There is no upper bound on the number of expected queries (using the model with random counterexamples of the previous section) that is uniform over all targets.

To see why, suppose the learner guesses interval $\left(\frac{1}{n+1}, \frac{1}{n}\right)$ for some n. For any $\epsilon > 0$ there is $N \in \mathbb{N}$ such that with probability greater than $1-\epsilon$, the learner gets a counterexample from the interval they guessed, $\left(\frac{1}{n+1}, \frac{1}{n}\right)$. Of course, even with this additional information, no matter the learner's guess at any stage at which they have received only negative counterexamples, this is clearly still the case. Thus, there can be no bound on expected queries which is uniform over all target concepts.

In this section we introduce an additional source of randomness, which allows for learning over infinite classes $\mathcal{C}$.[3] So, suppose $\mathcal{C}$ is a (possibly infinite) set of concepts on a set X. Suppose that we have probability measures μ on X and τ on $\mathcal{C}$. Suppose a target $A \in \mathcal{C}$ is selected randomly according to the distribution τ and the counterexamples to equivalence queries are selected randomly according to the distribution μ.

▶ **Theorem 3.1.** *Suppose that $\mathcal{C}$ is countable with finite Littlestone dimension d. There is an algorithm such that the expected number of queries over distributions μ on X and τ on $\mathcal{C}$ is at most $\tilde{O}(d)$.*

Proof. Let $\epsilon_k = \frac{1}{2^{k+1}}$ for $k \in \mathbb{N}$. The idea of the algorithm is to run our earlier algorithm on a $1-\epsilon_k$ fraction of the concepts with respect to the measure τ.

At stage k of the algorithm, we observe the following. Since $\mathcal{C}$ is countable, enumerate the collection $\mathcal{C} = \{C_i\}_{i \in \mathbb{N}}$. Then since $\sum_{i=1}^{\infty} P(C_i) = 1$, for any $\epsilon_k > 0$, there is $N_k = N(\epsilon_k) \in \mathbb{N}$ such that $\sum_{i=1}^{\infty} P(C_i) \geq 1 - \epsilon_k$.

Conditional on the target being among the first N_k concepts, the next idea is to run the algorithm from the previous section on this finite set for n steps where n is such that the probability that we have not identified the target after n steps is less than ϵ, for some $0 < \epsilon < 1$. This number $n = n_{d,\epsilon}$ depends only on the Littlestone dimension and ϵ, but not on N as we will explain.

We now bound the probability that the algorithm has not terminated after n steps, conditional on the target being in the first N_k many concepts. Since at any step, the probability that the Littlestone dimension drops is at least $\frac{1}{2}$ by Theorem 2.9, the probability that the algorithm has not terminated after n steps is at most the probability of a binomial random variable with probability $\frac{1}{2}$ achieving at most $d-1$ successes in n attempts, which is

[3] One might also think of the random EQ learning of Angluin and Dohrn as analyzing the maximum number of expected number of queries over all possible targets, while our model will analyze the *expected* number of queries where the expectation is taken over the concepts (with a fixed but arbitrary distribution) and over the counterexamples.

$$\sum_{k=0}^{d-1} \binom{n}{k} \left(\frac{1}{2}\right)^n \leq n^d/2^n.$$

Note that $n^d/2^n < \epsilon$ whenever $n - d\log n > \log\left(\frac{1}{\epsilon}\right)$. Hence,

$$n \geq \tilde{O}(d + \log(1/\epsilon))$$

is sufficient.

So at stage k, we run the algorithm for n steps as specified above. Either the target concept is found or we continue to stage $k+1$ on the larger concept class N_k. Since

$$(1-\epsilon_1)\left(\sum_{k=1}^{\infty} \epsilon_k\right) = 1/2\sum_{k=1}^{\infty} 1/2^{k+1} < 1,$$

the expected total number of queries is still bounded by $\tilde{O}(d + \log(1/\epsilon))$.[4] ◀

4 Compression schemes and stability

In this section, we follow the notation and definitions given in Johnson and Laskowski [6] on *compression schemes*, a notion due to Littlestone and Warmuth [10]. Roughly speaking, $\mathcal{C}$ admits a *d-dimensional compression scheme* if, given any finite subset F of X and some $f \in \mathcal{C}$, there is a way of encoding the set F with only d-many elements of F in such a way that F *can be recovered.*

We will give a formal definition, but we note that numerous variants of this idea appear throughout the literature, including as size d-array compression [2], extended compression schemes with b extra bits [4], and as unlabeled compression schemes [8]. In the definitions below, we let $dom(f)$ denote the domain of f and $\mathcal{C}_{\text{fin}}$ the restriction of $\mathcal{C}$ to finite subsets.

The following definition gives the notion of compression we consider within this section; the notion is equivalent to the notion of a d-compression with b extra bits [4]. The equivalence of these two notions is proved by Johnson and Laskowski [6, Proposition 2.1]. In our compression schemes, the role of the b extra bits is played by the reconstruction functions, and of course, the number of extra bits can be bounded in terms of the number of reconstruction functions (and vice versa). Of course, one is interested in optimizing both the size of the compression and the number of reconstruction functions (extra bits) in general.

▶ **Definition 4.1.** *We say that a concept class $\mathcal{C}$ has a d-*compression *if there is a compression function $\kappa : \mathcal{C}_{\text{fin}} \to X^d$ and a finite set $\mathcal{R}$ of reconstruction functions $\rho : X^d \to 2^X$ such that for any $f \in \mathcal{C}_{\text{fin}}$*

1. *$\kappa(f) \subseteq dom(f)$*
2. *$f = \rho(\kappa(f))|_{dom(f)}$ for at least one $\rho \in \mathcal{R}$.*

We work with the above notion mainly because it is the notion used in Johnson and Laskowski [6], and our goal is to improve a result therein. That result was later improved by Laskowski and appears in the unpublished notes of Guingona [5] (Theorem 4.1.3). When the original work on this result was completed, we were not aware of the work of Guingona [5], but as it turns out, our result improves both of these (the latter uses exponentially many reconstruction functions, while we use linearly many).

[4] There isn't anything particularly special about the sequence ϵ_k that we chose. Any sequence (ϵ_k) going to zero whose sum converges can be seen to work in the algorithm and affects only the constants in the expected number of steps, which we are not optimizing.

Johnson and Laskowski [6] prove that a concept class with finite Littlestone dimension has an extended d-compression for some d.[5] The precise value of d is not determined there, but was conjectured to be the Littlestone dimension. In Theorem 4.4, we will show that d can be taken to be the Littlestone dimension and $d + 1$ many reconstruction functions suffice.[6]

The question in Johnson and Laskowski [6] is the analogue (for Littlestone dimension) of a well-known open question from VC-theory [4]: is there a bound $A(d)$ linear in d such that every class of VC-dimension d has a compression scheme of size at most $A(d)$? In general, there is known to be a bound that is at most exponential in d [11].

▶ **Definition 4.2.** *Suppose* $\mathrm{Ldim}(\mathcal{C}) = d$*. Given a partial function* f*, say that* f *is* exceptional *for* $\mathcal{C}$ *if for all* $a \in \mathrm{dom}(f)$*,*

$$\mathcal{C}_{(a,f(a))} := \{g \in \mathcal{C} \mid g(a) = f(a)\}$$

has Littlestone dimension d*.*

▶ **Definition 4.3.** *Suppose* $\mathrm{Ldim}(\mathcal{C}) = d$*. Let* $f_{\mathcal{C}}$ *be the partial function given by*

$$f_{\mathcal{C}}(x) = \begin{cases} 0 & \mathrm{Ldim}(\mathcal{C}_{(x,0)}) = d \\ 1 & \mathrm{Ldim}(\mathcal{C}_{(x,1)}) = d \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is clear that $f_{\mathcal{C}}$ extends any partial function exceptional for $\mathcal{C}$.

▶ **Theorem 4.4.** *Any concept class* $\mathcal{C}$ *of Littlestone dimension* d *has an extended* d*-compression with* $(d + 1)$*-many reconstruction functions.*

Proof. If $d = 0$, then $\mathcal{C}$ is a singleton, and one reconstruction function suffices. So we may assume $d \geq 1$.

Fix some $f \in \mathcal{C}_{\text{fin}}$ with domain F. We will run an algorithm to construct a tuple of length at most d from F by adding one element at each step of the algorithm. During each step of the algorithm, we also have a concept class $\mathcal{C}_i$, with $\mathcal{C}_0 = \mathcal{C}$ initially.

If f is exceptional in $\mathcal{C}_{i-1}$, then the algorithm halts. Otherwise, pick either:

- $a_i \in F$ such that $f(a_i) = 1$ and

  $$(\mathcal{C}_{i-1})_{(a_i,1)} := \{g \mid g \in \mathcal{C}_{i-1},\, g(a_i) = 1\}$$

  has Littlestone dimension less than $\mathrm{Ldim}(\mathcal{C}_{i-1})$. In this case, set $\mathcal{C}_i := (\mathcal{C}_{i-1})_{(a_i,1)} = \{g \mid g \in \mathcal{C}_{i-1},\, g(a_i) = 1\}$.
- $d_i \in F$ such that $f(d_i) = 0$ and

  $$(\mathcal{C}_{i-1})_{(d_i,0)} := \{g \mid g \in \mathcal{C}_{i-1},\, g(d_i) = 0\}$$

  has Littlestone dimension less than $\mathrm{Ldim}(\mathcal{C}_{i-1})$. In this case, set $\mathcal{C}_i := (\mathcal{C}_{i-1})_{(d_i,0)}$.

[5] Their result is formulated for the sets of realizations of first-order formulas that are *stable*, but their proofs work for general concept classes, and Chase and Freitag [3] explain that stability is equivalent to finite Littlestone dimension.

[6] After proving this, we became aware of the unpublished result of Laskowski appearing as [5, Theorem 4.1.3] which shows one can take d to be the Littlestone dimension and uses 2^d many reconstruction functions.

We allow the algorithm to run for at most d steps. There are two distinct cases. If our algorithm has run for d steps, let $\kappa(f)$ be the tuple $(\bar{a}, \bar{d})$ of all of the elements a_i as above followed by all of the elements d_i as above for $i = 1, \ldots, d$. By choice of a_i and d_i, this tuple consists of d distinct elements. By construction the set

$$\mathcal{C}_{(\bar{a},\bar{d})} := \{g \in \mathcal{C} | \, g(a_i) = 1, \, g(d_i) = 0\}$$

has Littlestone dimension 0, that is, there is a unique concept in this class. So, given $(c_1, c_2, \ldots, c_n) \in X^d$ consisting of distinct elements, for $i = 0, \ldots, d$, we let $\rho_i(c_1, \ldots, c_n)$ be some g belonging to

$$\{g \in \mathcal{C} \, | \, g(c_j) = 1 \text{ for } j \leq i, \, g(c_j) = 0 \text{ for } j > i\},$$

if such a g exists. By construction, for some i, the Littlestone dimension of the concept class $\{g \in \mathcal{C} \cap F \, | \, g(c_j) = 1 \text{ for } j \leq i, \, g(c_j) = 0 \text{ for } j > i\}$ is zero, and so g is uniquely specified and will extend f.

We handle cases where the algorithm halts early by augmenting two of the reconstruction functions ρ_0 and ρ_1 defined above. Because ρ_0 and ρ_1 have so far only been defined for tuples consisting of d distinct elements, we can extend these to handle exceptional cases by generating tuples with duplicate elements.

If the algorithm stops at some step $i > 1$, then it has generated a tuple of length $i - 1$ consisting of some elements a_j and some elements d_k. Let $\bar{a}$ consist of the elements a_j chosen during the algorithm, and let $\bar{d}$ consist of the elements d_k chosen during the running of the algorithm. Observe that f is exceptional for $\mathcal{C}_{(\bar{a},\bar{d})}$.

If $\bar{a}$ is not empty, with initial element a', then let $\kappa(f) = (\bar{a}, a', \bar{d}, a', \ldots, a') \in F^d$. From this tuple, one can recover $(\bar{a}, \bar{d})$ (assuming $\bar{a}$ is nonempty), so we let $\rho_1(\bar{a}, a', \bar{d}, a', \ldots, a')$ be some total function extending $f_{\mathcal{C}_{(\bar{a},\bar{d})}}$, which itself extends f. So $\rho_1(\bar{a}, \bar{d})$ extends f whenever the algorithm halts before step d is completed *and* some a_i was chosen at some point. If $\bar{a}$ is empty, then let $\kappa(f) = (\bar{d}, d', \ldots, d') \in F^d$, where d' is the initial element of $\bar{d}$. From this tuple, one can recover $(\emptyset, \bar{d})$ (assuming $\bar{a}$ is empty), so we let $\rho_0(\bar{d}, d', \ldots, d')$ be total function extending $f_{\mathcal{C}_{(\emptyset,\bar{d})}}$, which itself extends f. Finally, if the algorithm terminates during step 1, then it has generated the empty tuple. In this case, let $\kappa(f) = (c, \ldots, c)$ for some $c \in F$. Then $\text{Ldim}(\mathcal{C}) = \text{Ldim}(\mathcal{C}_{(c,l)})$ for some $l \in \{0, 1\}$. In particular, if we have defined $\kappa(f') = (c, \ldots, c)$ above for some f' where the algorithm only returns c (rather than the empty tuple), then $1 - l = f'(c) \neq f(c)$, and so any such f' is handled by ρ_{1-l}. So we may overwrite ρ_l to set $\rho(c, \ldots, c)$ to be a total function extending $f_{\mathcal{C}}$, which itself extends f. For any tuple output by our algorithm, one of the reconstruction functions produces an extension of the original concept. ◀

References

1 Dana Angluin and Tyler Dohrn. The power of random counterexamples. In *International Conference on Algorithmic Learning Theory*, pages 452–465, 2017.

2 Shai Ben-David and Ami Litman. Combinatorial variability of Vapnik-Chervonenkis classes with applications to sample compression schemes. *Discrete Applied Mathematics*, 86(1):3–25, 1998.

3 Hunter Chase and James Freitag. Model theory and machine learning. *Bulletin of Symbolic Logic*, 25(3):319–332, 2019.

4 Sally Floyd and Manfred Warmuth. Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine learning*, 21(3):269–304, 1995.

5 Vincent Guingona. NIP theories and computational learning theory. URL: https://tigerweb.towson.edu/vguingona/NIPTCLT.pdf.

6 Hunter R Johnson and Michael C Laskowski. Compression schemes, stable definable families, and o-minimal structures. *Discrete & Computational Geometry*, 43(4):914–926, 2010.

7 Akash Kumar, Yuxin Chen, and Adish Singla. Teaching via best-case counterexamples in the learning-with-equivalence-queries paradigm. *Advances in Neural Information Processing Systems*, 34:26897–26910, 2021.

8 Dima Kuzmin and Manfred K Warmuth. Unlabeled compression schemes for maximum classes. *Journal of Machine Learning Research*, 8(9), 2007.

9 Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

10 Nick Littlestone and Manfred Warmuth. Relating data compression and learnability. Technical report, University of California, Santa Cruz, 1986.

11 Shay Moran and Amir Yehudayoff. Sample compression schemes for VC classes. *Journal of the ACM (JACM)*, 63(3):21, 2016.

12 Manfred K. Warmuth. Compressing to vc dimension many points. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 743–744, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

The Even-Path Problem in Directed Single-Crossing-Minor-Free Graphs

Archit Chauhan
Chennai Mathematical Institute, India

Samir Datta
Chennai Mathematical Institute & UMI ReLaX, India

Chetan Gupta
Indian Institute of Technology, Roorkee, India

Vimal Raj Sharma
Indian Institute of Technology, Jodhpur, India

Abstract

Finding a simple path of even length between two designated vertices in a directed graph is a fundamental NP-complete problem [24] known as the EvenPath problem. Nedev [28] proved in 1999, that for directed planar graphs, the problem can be solved in polynomial time. More than two decades since then, we make the first progress in extending the tractable classes of graphs for this problem. We give a polynomial time algorithm to solve the EvenPath problem for classes of H-minor-free directed graphs,[1] where H is a single-crossing graph.

We make two new technical contributions along the way, that might be of independent interest. The first, and perhaps our main, contribution is the construction of small, planar, *parity-mimicking networks*. These are graphs that mimic parities of all possible paths between a designated set of terminals of the original graph.

Finding vertex disjoint paths between given source-destination pairs of vertices is another fundamental problem, known to be NP-complete in directed graphs [14], though known to be tractable in planar directed graphs [34]. We encounter a natural variant of this problem, that of finding disjoint paths between given pairs of vertices, but with constraints on parity of the total length of paths. The other significant contribution of our paper is to give a polynomial time algorithm for the 3-*disjoint paths with total parity problem*, in directed planar graphs with some restrictions (and also in directed graphs of bounded treewidth).

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

Keywords and phrases Graph Algorithms, EvenPath, Polynomial-time Algorithms, Reachability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.43

Related Version *Full Version*: `http://arxiv.org/abs/2407.00237`

Funding *Archit Chauhan*: Partially supported by a grant from Infosys foundation and TCS PhD fellowship.
Samir Datta: Partially supported by a grant from Infosys foundation.

Acknowledgements We would like to thank anonymous reviewers for their useful comments and corrections in past submissions. We would also like to thank Martin Grohe for pointing out some references. We are also grateful to Geevarghese Philip and Vishwa Prakash H .V. for their help in reading some sections and improving the presentation of this paper.

[1] Throughout this paper, when referring to concepts like treewidth or minors of directed graphs, we intend them to apply to the underlying undirected graph.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 43; pp. 43:1–43:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a directed graph G, and two vertices s and t in it, checking for the existence of a a simple directed path from s to t is a fundamental problem in graph theory, known as the Reachability problem. The EvenPath problem is a variant of Reachability, where given a directed graph G and two vertices s and t we need to answer whether there exists a simple path of even length from s to t. EvenPath was shown to be NP-complete by LaPaugh and Papadimitriou [24] via a reduction from an NP-complete problem, the Path-Via-A-Vertex problem. On the other hand, they also show in [24] that its undirected counterpart is solvable in linear time. Several researchers have recently studied both the space, and simultaneous time-space complexity of EvenPath for special classes of graphs [5, 10]. A similar problem, that of finding a simple directed cycle of even length, called EvenCycle (which easily reduces to EvenPath), has also received significant attention. While polynomial-time algorithms have been known since long for the undirected version ([24, 39]), the question of tractablility of the directed version was open for over two decades before polynomial-time algorithms were given by McCuiag, and by Robertson, Seymour and Thomas [27]. More recently, Björklund, Husfeldt and Kaski [3] gave a randomized polynomial-time algorithm for finding a *shortest* even directed cycle in directed graphs.

Although EvenPath is NP-complete for general directed graphs, it is natural and interesting to investigate the classes of graphs for which it can be solved efficiently. In 1994, before the algorithm of [27], Gallucio and Loebl [15] gave a polynomial-time algorithm for EvenCycle in planar directed graphs. They did so by developing a routine for a restricted variant of EvenPath (when s, t lie on a common face, and there are no even directed cycles left on removal of that face). Following that, Nedev in 1999, showed that EvenPath in planar graphs is polynomial-time solvable [28]. Planar graphs are an example of a *minor-closed* family, which are families of graphs that are closed under edge contraction and deletion. Minor-closed families include many more natural classes of graphs, like graphs of bounded genus, graphs of bounded treewidth, apex graphs. A theorem of Robertson-Seymour [33] shows that every minor-closed family can be characterized by a set of finite forbidden minors. Planar graphs, for example, are exactly graphs with $K_{3,3,}, K_5$ as forbidden minors [38]. In this paper, we consider the family of H-minor-free graphs, where H is any fixed single-crossing graph, i.e., H can be drawn on the plane with at most one crossing. Such families are called single-crossing-minor-free graphs. They include well-studied classes of graphs like K_5-minor-free graphs, $K_{3,3}$-minor-free graphs.[2] Robertson and Seymour showed that single-crossing-minor free graphs admit a decomposition by (upto) 3-clique-sums, into pieces that are either of bounded treewidth, or planar [30]. This is a simpler version of their more general theorem regarding decomposition of H-minor free graphs, (where H is any fixed graph) by clique sums, into more complex pieces, involving apices and vortices [32]. Solving EvenPath on single-crossing-minor free graphs would therefore be a natural step to build an attack on more general minor closed familes.

Many results on problems like reachability, matching, coloring, isomorphism, for planar graphs have been extended to $K_{3,3}$-minor-free graphs and K_5-minor-free graphs as a next step (see [36, 21, 22, 37, 35, 11, 2]). Chambers and Eppstein showed in [6] that using the results of [4, 17] for maximum flows in planar and bounded treewidth graphs, respectively, maximum flows in single-crossing-minor-free graphs can be computed efficiently. Following the result of [1] which showed that perfect matching in planar graphs can be found in NC, Eppstein and Vazirani in [13], extended the result to single-crossing-minor-free graphs.

[2] Both $K_5, K_{3,3}$ have crossing number one. Also, note that both the families, K_5-minor-free, and $K_{3,3}$-minor-free, have graphs of $O(n)$ genus.

1.1 Our Contributions

From here onwards, we will drop the term "directed" and assume by default that the graphs we are referring to are directed, unless otherwise stated. Operations like clique sums, decomposing the graphs along separating triples, pairs, etc., will be applied on the underlying undirected graphs. The following is the main theorem we prove in this paper:

▶ **Theorem 1.** *Given an H-minor-free graph G for any fixed single-crossing graph H, the* EvenPath *problem in G can be solved in polynomial time.*

We first apply the theorem of Robertson-Seymour (theorem 3), and decompose G using 3-clique sums into pieces that are either planar or of bounded treewidth. Though EvenPath is tractable in planar graphs, and can also be solved in bounded treewidth graphs by Courcelle's theorem, straightforward dynamic programming does not yield a polynomial-time algorithm for the problem, as we will explain in subsequent sections. One of the technical ingredients that we develop to overcome the issues is that of *parity-mimicking networks*, which are graphs that preserve the parities of various paths between designated terminal vertices of the graph it mimics. We construct them for upto three terminal vertices. The idea of mimicking networks has been used in the past in other problems, like flow computation [6, 7, 17, 20, 23], and in perfect matching [13]. The ideas we use for constructing parity mimicking networks however, do not rely on any existing work that we know of. For technical reasons, we require our parity mimicking networks to be of bounded treewidth *and* planar, with all terminals lying on a common face. These requirements make it more challenging to construct them (or even to check their existence), than might seem at a first glance. One of our main contributions is to show (in lemma 8) the construction of such networks, for upto three terminals. It might be of independent interest to see if a more simpler construction exists (perhaps a constructive argument to route paths, that has eluded us so far), that avoids the hefty case analysis we do, and also if they can be constructed for more than three terminals.

We also come across a natural variant of another famous problem. Suppose we are given a graph G and vertices $s_1, t_1, s_2, t_2 \ldots s_k, t_k$ (we may call them terminals) in it. The problem of finding pairwise vertex disjoint paths, from each s_i to t_i is a well-studied problem called the disjoint paths problem. In undirected graphs, the problem is in P when k is fixed [31, 29], but NP-complete otherwise [26]. For (directed) graphs, the problem is NP-complete even for $k = 2$ [14]. In planar graphs, it is known to be in P for fixed k [34, 9, 25]. We consider this problem, with the additional constraint that the sum of lengths of the s_i-t_i paths must be of specified parity. We hereafter refer to the parity of the sum of lengths as total parity, and refer to the problem as DisjPathsTotalParity. In the undirected setting, a stricter version of this problem has been studied, where each s_i-t_i path must have parity p_i that is specified in input. This problem was shown to be in P for fixed k, by Kawarabayashi et al. [18]. However much less is known in directed setting. While DisjPathsTotalParity can be solved for fixed k in bounded treewidth graphs using Courcelle's theorem [8], we do not yet know if it is tractable in planar graphs, even for $k = 2$. The other main technical contribution of our paper is in lemma 10, where we show that in some special cases, i.e., when there are four terminals, three of which lie on a common face of a planar graph, DisjPathsTotalParity can be solved in polynomial time for $k = 3$. We do this by showing that under the extra constraints, the machinery developed by [28] can be further generalized and applied to find a solution in polynomial time. The question of tractability of DisjPathsTotalParity in planar graphs, without any constraint of some terminals lying on a common face is open, and would be interesting to resolve. A polynomial-time algorithm for it (for fixed k), would yield a polynomial-time algorithm for EvenPath in graphs with upto k crossings, which is currently unknown.

Though the proofs of lemmas 8,10 form the meat of technical contributions of the paper, we give a proof idea, deferring the proofs to the full version of the paper.

2 Preliminaries

From now onwards we will refer to simple, directed paths as just paths. For a path P, and a pair of vertices u and v on P, such that u occurs before v in P, $P[u, v]$ denotes the subpath of P from u to v. If P_1 and P_2 are two paths that are vertex disjoint, except possibly sharing starting or ending vertices, then we say that P_1, P_2 are *internally disjoint* paths. If P_1's ending vertex is same as the starting vertex of P_2, then we denote the concatenation of P_1 and P_2 by $P_1.P_2$. We will use the numbers $0, 1$ to refer to *parities*, 0 for even parity and 1 for odd parity. We say a path P is of *parity* p ($p \in \{0, 1\}$), if its length modulo 2 is p. We will use a well-known structural decomposition of H-minor-free graphs due to [30]. We recall the definition of clique sums:

▶ **Definition 2.** *A k-clique-sum of two graphs G_1, G_2 can be obtained from the disjoint union of G_1, G_2 by identifying a clique in G_1 of at most k vertices with a clique of the same number of vertices in G_2, and then possibly deleting some of the edges of the merged clique.*

Thus when separating G along a separating pair/triplet, we can add virtual edges if needed, to make the separating pair/triplet a clique. The virtual edges will not be used in computation of path parities, they are only used to compute the decomposition. We can keep track of which edges in the graph are virtual edges and which are the real edges throughout the algorithm. We can repeatedly apply this procedure to decompose any graph G into smaller pieces. The following is a theorem from [30].

▶ **Theorem 3** (Robertson-Seymour [30]). *For any single-crossing graph H, there is an integer τ_H such that every graph with no minor isomorphic to H is either*

1. *the proper* 0-, 1-, 2- *or* 3-*clique-sum of two graphs, or*
2. *planar*
3. *of treewidth $\leq \tau_H$.*

Thus, every H-minor-free graph, where H is a single-crossing graph, can be decomposed by 3-clique sums into graphs that are either planar or have treewidth at most τ_H. Polynomial time algorithms are known to compute this decomposition [12, 19, 16] (and also NC algorithms [13]). The decomposition can be thought of as a two colored tree (see [12, 6, 13] for further details on the decomposition), where the blue colored nodes represent *pieces* (subgraphs that are either planar or have bounded treewidth), and the red nodes represent cliques at which two or more pieces are attached. We call these nodes of the tree decomposition as *piece nodes* and *clique nodes*, respectively. The edges of the tree describe the incidence relation between pieces and cliques (see Figure 2). We will denote this decomposition tree by T_G. We will sometimes abuse notation slightly and refer to a piece of T_G (and also phrases like *leaf* piece, *child* piece), when it is clear from the context that we mean the piece represented by the corresponding node of T_G. Note that the bounded treewidth and planarity condition on the pieces we get in the decomposition, is along with their virtual edges. As explained in [6, 13], we can assume that in any planar piece of the decomposition, the vertices of a separating pair or triplet lie on a common face (Else we could decompose the graph further).

Suppose G decomposes via a 3-clique sum at clique c into G_1 and G_2. Then we write G as $G_1 \oplus_c G_2$. More generally, if $G_1, G_2, \ldots, G_\ell$ all share a common clique c, then we use $G_1 \oplus_c G_2 \oplus_c \ldots \oplus_c G_\ell$ to mean $G_1, G_2, \ldots, G_\ell$ are glued together at the shared clique. If it is clear from the context which clique we are referring to, we will sometimes drop the subscript

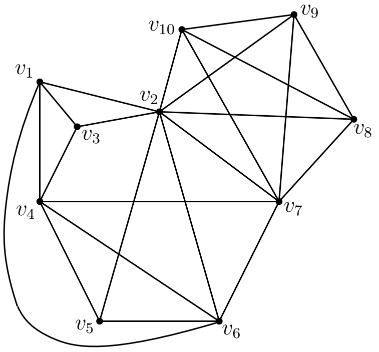


Figure 1 An example of a graph G. We ignore directions here.

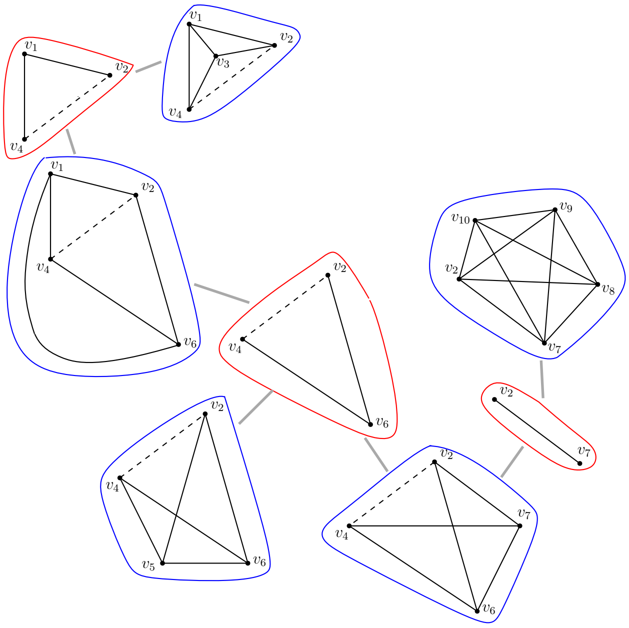


Figure 2 A clique sum decomposition of G. Red nodes are the clique nodes and blue node the piece nodes. Dashed edges denote virtual edges.

and simply use $G_1 \oplus G_2 \oplus \ldots \oplus G_\ell$ instead. Suppose G'_2 is a graph that contains the vertices of the clique c shared by G_1 and G_2. We denote by $G[G_2 \rightarrow G'_2]$, the graph $G_1 \oplus_c G'_2$, i.e., replacing the subgraph G_2 of G, by G'_2, keeping the clique vertices intact. We will also use the notion of *snapshot* of a path in a subgraph. If G can be decomposed into G_1 and G_2 as above, and P is an s-t path in G, its snapshot in G_1 is the set of maximal subpaths of P, restricted to vertices of G_1. Within a piece, we will sometimes refer to the vertices of separating cliques, and s and t, as *terminals*.

In figures, we will generally use the convention that a single arrow denotes a path segment of odd parity and double arrow denotes a path segment of even parity, unless there is an explicit expression for the parity mentioned beside the segment.

3 Overview and Technical Ingredients

We first compute the 3-clique sum decomposition tree of G, T_G. We can assume that s, t, each occur in only one piece of T_G, S and T, respectively.[3] We call the pieces S and T, along with the pieces corresponding to nodes that lie in the unique path in T_G joining S and T, as the *main* pieces of T_G, and the remaining pieces are called the *branch* pieces of T_G. We will assume throughout that T_G is rooted at S.

The high level strategy of our algorithm follows that of [6]. The algorithm has two phases. In the first phase, we simplify the branch pieces of the decomposition tree. Any s-t even path P must start and end inside the main pieces S and T, respectively. However, it may

[3] If they are part of a separating vertex/pair/triplet then they may occur in multiple pieces of T_G. Say s is a part of many pieces in T_G. To handle that case, we can introduce a dummy s' and add an edge from s' to s and reduce the problem to finding an odd length path from s' to t. The vertex s' now will occur in a unique piece in T_G. Vertex t can be handled similarly.

take a detour into the branch pieces. Suppose L is a leaf branch piece of T_G, attached to its parent piece, say G_i, via a 3-clique c. Using Nedev's algorithm or Courcelle's theorem, we can find paths of various parities between vertices of c in L, which constitutes the *parity configuration* of L with respect to c (formally defined in next subsection). We will replace L by a parity mimicking network of L with respect to vertices of c, L'. L' will mimic the parity configuration of L and hence preserve the parities of all s-t paths of original graph. The parity mimicking networks we construct are small and planar, with the terminals (vertices of c) all lying on a common face, as decribed in lemma 8. Therefore, if G_i is of bounded treewidth, then $G_i \oplus L'$ will be of bounded treewidth. And if G_i is planar, then we can plug L' in the face of G_i that is common to vertices of c, and $G_i \oplus L'$ will be planar. This allows us compute the parity configurations of the merged piece, and repeat this step until a single branch, i.e. a path, remains in the decomposition tree, consisting only of the main pieces (connected by cliques), including S and T.

In the second phase, we start simplifying the main pieces, starting with the leaf piece T. Instead of a single mimicking network for T, we will store a set of small networks, each of them mimicking a particular snapshot of a solution. We call them *projection networks.* Since a snapshot of an s-t even path in T can possibly be a set of disjoint paths between the (upto) four terminals in T, we require the DisjPathsTotalParity routine of lemma 10 to compute these projection networks. We combine the parent piece with each possible projection network. The merged piece will again be either planar or of bounded treewidth, allowing us to continue this operation towards the root node until a single piece containing both s and t remains. We query for an s-t even path in this piece and output yes iff there exists one. At each step, the number of projection networks used to replace the leaf piece, and their combinations with its parent piece will remain bounded by a constant number.

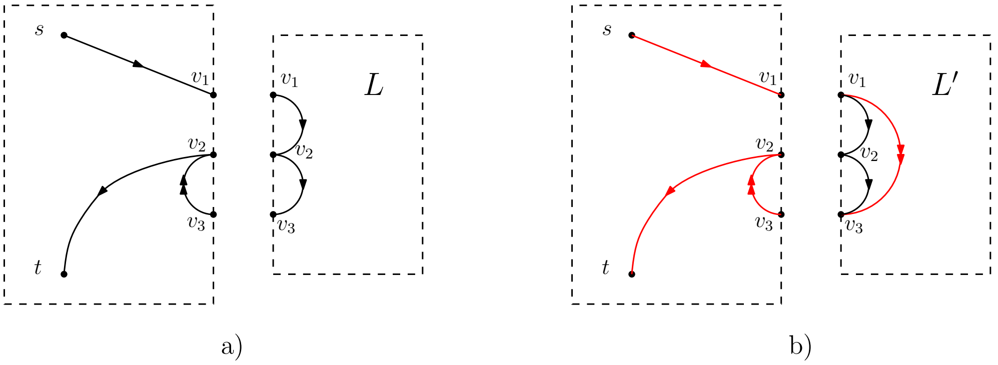
Once we have the decision version of EvenPath, we show a poly-time self-reduction using the decision oracle of EvenPath to construct a solution, in the full version of the paper.

Necessity of a two phased approach

We mention why we have two phases and different technical ingredients for each.

- Instead of a single parity mimicking network, we need a set of projection networks for the leaf piece in the second phase because it can have upto four terminals (three vertices of the separating clique and the vertex t), and we do not yet know how to find (or even the existence of) parity mimicking networks with the constraints we desire, for graphs with four terminals.
- We cannot however use a set of networks for each piece in phase I because of the unbounded degree of T_G. Suppose a branch piece G_i is connected to its parent piece by clique c, and suppose G_i has child pieces $L_1, L_2, \ldots, L_\ell$, attached to G_i via disjoint cliques $c_1, c_2, \ldots, c_\ell$, respectively. An even s-t path can enter G_i via a vertex of c, then visit any of $L_1, L_2, \ldots, L_\ell$ in any order and go back to the parent of G_i via another vertex of c. If we store information regarding parity configurations of $L_1, L_2, \ldots, L_\ell$ as sets of projection networks, we could have to try exponentially many combinations to compute information of parity configurations between vertices of c in the subtree rooted at G_i (note that ℓ could be $O(n)$). Therefore, we compress the information related to parity configurations of L_i into a single parity mimicking network L'_i, while preserving solutions, so that the combined graph $(((G_i \oplus_{c_1} L'_1) \oplus_{c_2} L'_2 \ldots) \oplus_{c_\ell} L'_\ell)$ is either planar or of bounded treewidth.

We will now describe these ingredients formally in the remaining part of this section.



a)

b)

Figure 3 Figure a) shows the input graph and b) shows the graph with L replaced by an erroneous mimicking network L'. Suppose the original graph in a) has no s-t even path but does have an s-t even walk as shown in the figure, using vertex v_2 twice. If we query for a path from v_1 to v_3 in L, and add a direct v_1 to v_3 path of that parity in L', we end up creating a false solution since v_2 is freed up to be used outside L'. Hence there must be equality between corresponding direct sets.

3.1 Parity Mimicking Networks

We first define the parity configuration of a graph, which consists of subsets of $\{0, 1\}$ for each pair, triplet of terminals, depending on whether there exists “direct” or “via” paths of parity even, odd, or both (we use 0 for even parity and 1 for odd). We formalise this below.

▶ **Definition 4.** *Let L be a directed graph and $T(L) = \{v_1, v_2, v_3\}$ be the set of terminal vertices of L. Then, $\forall i, j, k \in \{1, 2, 3\}$, such that i, j, k are distinct, we define the sets* $\mathrm{Dir}_L(v_i, v_j)$*, and* $\mathrm{Via}_L(v_i, v_k, v_j)$ *as:*

- $\mathrm{Dir}_L(v_i, v_j) =$*{p | there exists a path of parity p from v_i to v_j in $L - v_k$ }*
- $\mathrm{Via}_L(v_i, v_k, v_j) =$*{p | there exists a path of parity p from v_i to v_j via v_k* and *there does not exist a path of parity p from v_i to v_j in $L - v_k$}*

We say that the $\mathrm{Dir}_L, \mathrm{Via}_L$ *sets constitute the* parity configuration *of the graph L with respect to $T(L)$. We call the paths corresponding to elements in* $\mathrm{Dir}_L, \mathrm{Via}_L$ *sets as* Direct paths *and* Via paths*, respectively.*

The parity configuration of a graph can be visualised as a table. We have defined it for three terminals, it can be defined in a similar way for two terminals. It is natural to ask the question that given a parity configuration $\mathcal{P}$ independently with respect to some terminal vertices, does there exist a graph with those terminal vertices, realising that parity configuration. If not, we say that $\mathcal{P}$ is unrealisable. It is easy to see that the number of parity configurations for a set of three terminals is bounded by 4^{12}, many of which are unrealizable. We now define parity mimicking networks.

▶ **Definition 5.** *A graph L' is a parity mimicking network of a another graph L (and vica versa), if they share a common set of terminals, and have the same parity configuration, $\mathcal{P}$, w.r.t. the terminals. We also call them parity mimicking networks of parity configuration $\mathcal{P}$.*

The reason we differentiate between direct paths and via paths while defining parity configurations is to ensure that no false solutions are introduced on replacing a leaf piece of T_G by its mimicking network (see Figure 3). Note that in our definition of *Via* sets, we exclude parity entries of via paths between two terminals if that parity is already present in *Dir* set between the same terminals. We do so because this makes the parity configurations easier to enumerate in our construction of parity mimicking networks. In Figure 4, we describe why doing this will still preserve solutions.

We also need to consider the case where multiple leaf pieces, in T_G are attached to a common parent piece via a shared clique (as seen in Figure 2). In this case, we will replace the entire subgraph corresponding to the clique sum of the sibling leaf pieces by one parity mimicking network. To compute the parity configuration of the combined subgraph of leaf pieces, we make the following observation:

▶ **Observation 6.** *Let $L_1, L_2, \ldots, L_\ell$ be leaf branch pieces that are pairwise disjoint except for a common set of terminal vertices, say $\{v_1, v_2, v_3\}$. Let $L = L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$. Then the parity configuration of L with respect to $\{v_1, v_2, v_3\}$ can be computed by:*

$$\mathrm{Dir}_L(v_i, v_j) = \bigcup_{a=1}^{\ell} \mathrm{Dir}_{L_a}(v_i, v_j) \tag{1}$$

$$\mathrm{Via}_L(v_i, v_k, v_j) =$$
$$\left(\bigcup_{a=1}^{\ell} \mathrm{Via}_{L_a}(v_i, v_k, v_j) \cup \bigcup_{a,b=1}^{\ell} (\mathrm{Dir}_{L_a}(v_i, v_k) \boxplus \mathrm{Dir}_{L_b}(v_k, v_j)) \right) \backslash \mathrm{Dir}_L(v_i, v_j) \tag{2}$$

where $A \boxplus B$ denotes the set formed by addition modulo 2 between all pairs of elements in sets A, B, and $i, j, k \in \{1, 2, 3\}$ are distinct.

The intuition behind the observation is simple. Any direct path in L from v_i to v_j must occur as a direct path in one of $L_1, L_2 \ldots L_\ell$ since they are disjoint except for terminal vertices. Any via path in L from v_i to v_j via v_k can occur in two ways, either as a v_i-v_k-v_j via path in one of $L_1, L_2 \ldots L_\ell$, or as a concatenation of two direct paths, one from v_i to v_k in some piece L_i, and another from v_k to v_j in another piece L_j. Note that although the observation is for the case when all $L_1, L_2, \ldots, L_\ell$ share a common 3-clique $\{v_1, v_2, v_3\}$, it is easy to see it can be tweaked easily to handle the cases when some of the $L_i's$ are attached via a 2-clique that is a subset of the 3-clique.

The next lemma states that replacing leaf piece nodes in T_G by parity mimicking networks obeying some planarity conditions, will preserve the existence of s-t paths of any particular parity, and also preserve conditions on treewidth and planarity for the combined piece.

▶ **Lemma 7.** *Let G be a graph with clique sum decomposition tree T_G, and let $L_1, L_2 \ldots, L_\ell$ be set of leaf branch pieces of T_G, attached to their parent piece G_1 via a common clique c. Let L' be a parity mimicking network of $L_1 \oplus L_2 \oplus \ldots L_\ell$ with respect to c, such that L' is planar, and vertices of c lie on a common face in L'. Then:*

1. *There is a path of parity p from s to t in G iff there is a path of parity p from s to t in $G[L_1 \oplus L_2 \oplus \ldots L_\ell \rightarrow L']$.*
2. *If G_1 is planar, then $G_1 \oplus L'$ is also planar.*
3. *If G_1 has treewidth τ_H, and L' has treewidth $\tau_{L'}$, then $G_1 \oplus L'$ has treewidth max$(\tau_H, \tau_{L'})$*

Proof.

1. The proof essentially follows from the definition of parity mimicking networks and observation 6, since we can replace the snapshot of any s-t path P in L_i by a path of corresponding parity in L'_i and vice-versa.
2. This follows since in the decomposition, vertices of separting cliques in every piece lie on the same face, and so is the case for L' by assumption. Therefore we can embed L' inside the face in G_1, on the boundary of which v_1, v_2, v_3 lie.
3. This follows since we can merge tree decompositons of G_1, L' along bags consisting of the common clique. ◀

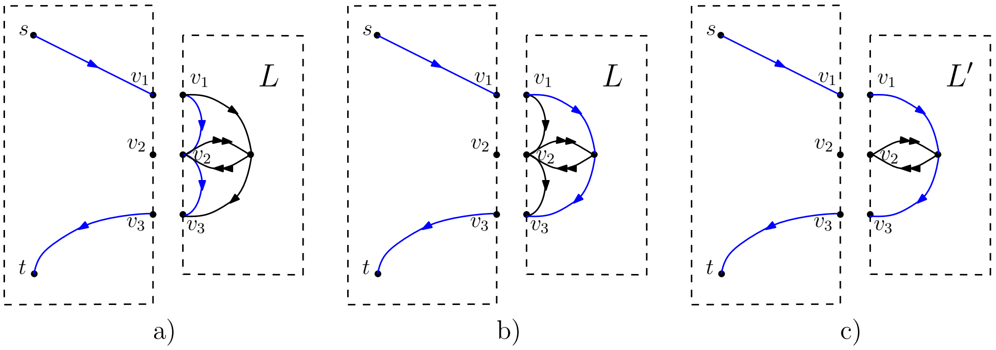


Figure 4 Figure a) denotes the original graph which has both a direct path, as well as a via path of even parity from v_1 to v_3. Suppose the via path is part of an even s-t path solution, as marked by blue. Then in L itself, we could replace the via path by the direct path and it would still be a valid even s-t path, as marked in blue in b). Hence in the mimicking network L', too (shown in c)), we could use the direct v_1 to v_3 path of the same parity. Therefore we do not need to put the parity of the v_1-v_2-v_3 path in $Via_L(v_1, v_2, v_3)$, since the same parity is already present in $Dir_L(v_1, v_3)$, and $Dir_L(v_1, v_3) = Dir_{L'}(v_1, v_3)$.

Now we will show how to compute parity mimicking networks that are small in size (and hence of bounded treewidth), and also planar, with terminal vertices lying on the same face, for a given parity configuration of a graph L.

▶ **Lemma 8.** *Suppose L is a graph with terminals $T(L) = \{v_1, v_2, v_3\}$, and suppose we know the parity configuration of L with respect to $\{v_1, v_2, v_3\}$. We can in polynomial-time, find a parity mimicking network L' of L, with respect to $\{v_1, v_2, v_3\}$ which consists of at most* 18 *vertices, and is also planar, with v_1, v_2, v_3 lying on a common face.*

Proof. We give a brief idea of the proof and defer the full proof to the full version of this paper.As noted above, the number of possible parity configurations are finite (bounded by 4^{12} for three terminals), but the number is too large to enumerate over all of them and individually construct the mimicking networks. We use some observations to make the case analysis tractable. We refer to elements of sets $Dir_L(v_i, v_j)$ as *entries*. But we abuse notation slightly and distinguish them from the boolean values $0, 1$. For example, we always distinguish between an entry of $Dir_L(v_1, v_2)$, and an entry of $Dir_L(v_2, v_3)$, even if they have the same *value* (0 or 1). A natural constructive approach would be to iteratively do the following step for all i, j, p : add a path of length $2 - p$ from v_i to v_j in L', disjoint from existing paths of L', if there is an entry of parity p present in $\mathrm{Dir}_{L'}(v_i, v_j)$. Its easy to check that this will result in a planar L' with terminals on a common face. However, this could lead to wrong parity configurations in L'. For example, L could have a direct paths of parity 1 from v_1 to v_2, and a direct path of parity 0 from v_2 to v_3, but no path of parity 1 from v_1 to v_3, either direct or via v_2 (see Figure 5). We will call pairs of such entries as *bad pairs.* The entries that are part of *any* bad pair are called *bad entries.* Though the example in Figure 5 has a simple fix for the bad pair, it becomes more complicated to maintain the planarity conditions as the number of bad pairs increase. Let $\mathcal{P}_L$ be the parity configuration of L. The idea of the proof is to define a *bad kernel* of $\mathcal{P}_L$, as the *sub-configuration* consisting of all the *bad entries* of $\mathcal{P}_L$. The *closure* of the bad kernel is defined as the parity configuration obtained from it by adding “minimal” number of entiries to make it realizable. We observe that the closure remains a sub-configuration of $\mathcal{P}_L$. Suppose that we can somehow construct a planar mimicking network for the *closure* of the *bad kernel* of $\mathcal{P}_L$, with terminals lying on a

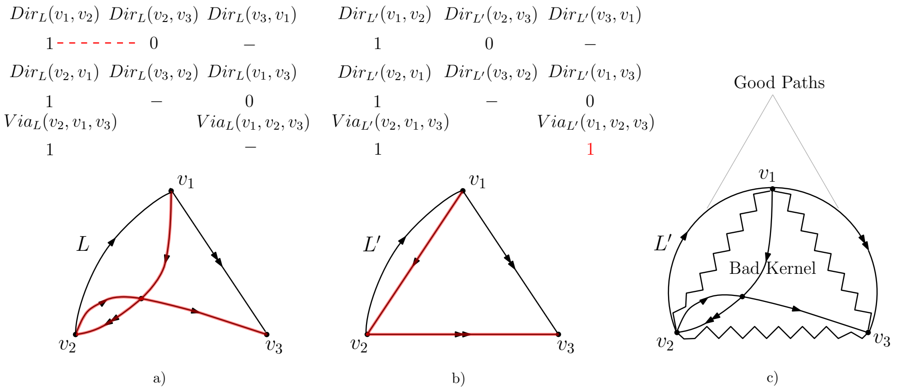

$Dir_L(v_1,v_2)$ $Dir_L(v_2,v_3)$ $Dir_L(v_3,v_1)$
1 0 –
$Dir_L(v_2,v_1)$ $Dir_L(v_3,v_2)$ $Dir_L(v_1,v_3)$
1 – 0
$Via_L(v_2,v_1,v_3)$ $Via_L(v_1,v_2,v_3)$
1 –
$Dir_{L'}(v_1,v_2)$ $Dir_{L'}(v_2,v_3)$ $Dir_{L'}(v_3,v_1)$
1 0 –
$Dir_{L'}(v_2,v_1)$ $Dir_{L'}(v_3,v_2)$ $Dir_{L'}(v_1,v_3)$
1 – 0
$Via_{L'}(v_2,v_1,v_3)$ $Via_{L'}(v_1,v_2,v_3)$
1 1
Good Paths
Bad Kernel
L
L'
L'
v_1
v_2
v_3
a)
b)
c)

Figure 5 Fig a) denotes a graph L with its parity configuration table (only relevant sets). Fig b) denotes a "parity mimicking network", if for each pair of terminals, we just independently put paths of correct parity, disjoint from each other. It leads to an extra path (highlighted in red) from v_1 to v_3 via v_2 in L', of odd parity. Pairs of such entries, for which we cannot add disjoint paths are called bad entries as marked by the dashed red line in the parity configuration table in a). Fig c) outlines the approach used to construct the correct mimicking network. The two paths corresponding to bad pair entries, form the bad kernel, for which we construct a mimicking network by enumerating cases. The remaining paths can be added iteratively, disjoint from all existing paths, on the outer face.

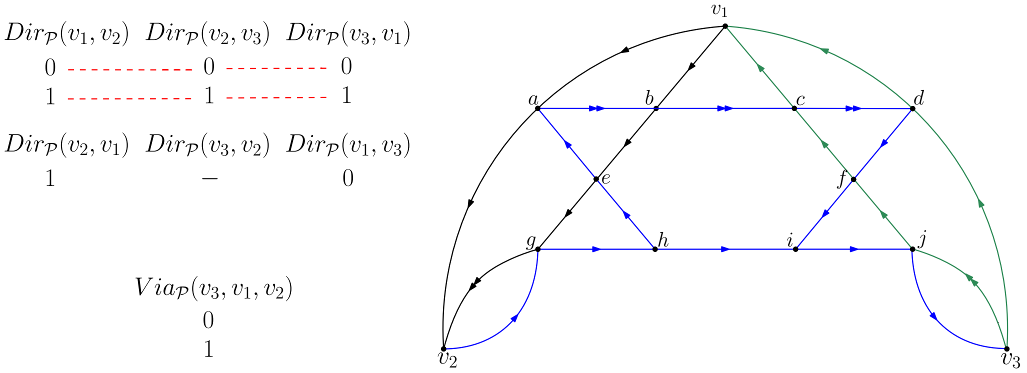

$Dir_{\mathcal{P}}(v_1,v_2)$ $Dir_{\mathcal{P}}(v_2,v_3)$ $Dir_{\mathcal{P}}(v_3,v_1)$
0 0 0
1 1 1
$Dir_{\mathcal{P}}(v_2,v_1)$ $Dir_{\mathcal{P}}(v_3,v_2)$ $Dir_{\mathcal{P}}(v_1,v_3)$
1 – 0
$Via_{\mathcal{P}}(v_3,v_1,v_2)$
0
1
v_1
a
b
c
d
e
f
g
h
i
j
v_2
v_3

Figure 6 An example of a more non-trivial bad kernel, and a mimicking network realising its closure. This is a subcase of case $(4,0)$ described in the full proof. We give a list of paths along with their lengths, for ease of reader to check that the network obeys the parity configuration.

common face. Then we show that paths corresponding to leftover parity entries of L can be safely added using the constructive approach described above. Hence it suffices to construct parity mimicking networks for closures of all possible bad kernels. We use some observations to show that the number of possible types of bad kernels cannot be too large, and enumerate over each type, explicitly constructing the parity mimicking networks of their closures. ◀

3.2 Disjoint Paths with Parity Problem

In this section, we will define and solve the DisjPathsTotalParity problem for some special cases and types of graphs. We define the problem for three paths between four terminals.

▶ **Definition 9.** *Given a graph G and four distinct terminals v_1, v_2, v_3, and v_4 in $V(G)$, the* DisjPathsTotalParity *problem is to find a set of three pairwise disjoint paths, from v_1 to v_2, v_2 to v_3, and from v_3 to v_4, such that the total parity is even, if such a set of paths exist, and output no otherwise.*

The problem where total parity must be odd can be easily reduced to this by adding a dummy neighbour to v_4. The problem is NP-hard in general graphs since the even path problem trivially reduces to this. We show that the above problem can be solved in polynomial time in following two cases:

▶ **Lemma 10.** *Let G be a graph, and v_1, v_2, v_3, v_4 be four vertices of G. Both decision as well as search versions of* DisjPathsTotalParity *for these vertices as defined above can be solved in polynomial time in the following cases:*

1. *If G has constant treewidth.*
2. *If G is planar and v_1, v_2, v_3 lie on a common face of G.*

Proof. Proofs of both parts can be found in the full version of the paper. We give a high level idea of the proof of the second part. The argument of Nedev for EvenPath uses two main lemmas. One lemma states that if there are two paths P_1, P_2, of different parities from s to t, then their union forms a (at least one) structure, which they call an *odd list superface*. It (roughly) consists of two internally disjoint paths of different parities, with a common starting vertex, say b and a common ending vertex, say e. Let F denote such a superface. They show that there exist two disjoint paths in $P_1 \cup P_2 - F$, one from s to b, and one from e to t. This provides a "switch" in $P_1 \cup P_2$, and if we can find this switch efficiently, then we can use existing 2-disjoint path algorithms to connect s and t via this switch. But the number of odd list superfaces in a graph can be exponential. The second lemma of Nedev says that we can exploit the structure of planarity and show that each of the odd list superfaces formed by $P_1 \cup P_2$, "contain" a "minimal" odd list superface, which they call a *simple* odd list superface, that obeys the same conditions. The set of simple odd list superfaces is small and can be enumerated in polynomial time. In our setting, we start from the case that two instances of three disjoint paths between the specified terminals exist, such that they have different total parity. Say the instances are P_1, P_2, P_3, and P'_1, P'_2, P'_3. At least one of P_i, P'_i must be of different parity. We show that using the constraints of three terminals on a face, and using ideas of *leftmost* (and rightmost) paths of $P_i \cup P'_i$, for each case of $i \in \{1, 2, 3\}$, there does exist an analogous structure: a simple odd list super face, and four disjoint path segments connecting the required vertices. A point to note is that in Nedev's argument, *any* odd list superface formed by P_1, P_2 could be trimmed to a simple odd list superface that would give a valid solution. That does not hold true here. We generalise their lemma, and argue that there does exist *at least one* odd list superface between P_i, P'_i that will work in our setting. ◀

4 Main Algorithm

We now explain the two phases of the algorithm.

4.1 Phase 1

1. Find the 3-clique sum decomposition tree T_G. Mark the piece that contains the vertex s as the root of T_G.
2. Pick any maximal set of leaf branch pieces of T_G, say $L_1, L_2, \ldots, L_\ell$, which are attached to a parent piece G_i via a common clique. Compute their parity configurations using Nedev's algorithm, or using Courcelle's theorem.Then compute the parity configuration of $L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$ using observation 6.
3. Compute the parity mimicking network, L', of $L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$ using lemma 8. Replace $L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$ by L' and merge it with G_i.
4. Since $G_i \oplus L'$ is either of bounded treewidth or is planar by lemma 7, we can repeat this step until no branch pieces remain.

4.2 Phase II

Let G' denote the graph after phase I. After phase I, the modified tree $T_{G'}$ looks like a path of pieces, $G_1, G_2, \ldots, G_m$, joined at cliques $c_1, c_2 \ldots c_{m-1}$.[4] The vertex s is in root piece G_1, and t in leaf piece G_m (we use G_1, G_m instead of S, T here for notational convenience). We can write $G' = G_1 \oplus_{c_1} G_2 \oplus_{c_2} \ldots \oplus_{c_{m-1}} G_m$. Since it is clear in this phase that c_i is the clique joining G_i, G_{i+1}, we will omit the subscript for notational convenience and just write $G_1 + G_2 + \ldots + G_m$ instead. Let $c_{m-1} = \{v_1, v_2, v_3\}$ and let $i, j, k \in \{1, 2, 3\}$ be distinct. The snapshot of any even s-t path P in G_m, can be one of the following four types (see figure 7):

- Type 1 : A path from v_i to t without using v_j, v_k.
- Type 2 : A path from v_i to t via v_j, without using v_k.
- Type 3 : A path from v_i to t via v_j, v_k.
- Type 4 : A path from v_i to v_j and a path from v_k to t, both disjoint from each other.

We call any path/set of paths in G_m of one of the above types as a *potential snapshot* of G_m. We now construct the *projection networks* of potential snapshots of G_m.

▶ **Definition 11.** *Let G_m be the leaf piece as described above with clique $c_{m-1} = \{v_1, v_2, v_3\}$, and vertex t present in G_m.*

- *For each of the types described above, for all $i \in \{1, 2, 3\}$, and for all $p \in \{0, 1\}$, find a potential snapshot (if it exists) in G_m from v_i to t, of total parity p, using lemma 10.*
- *Let J be a potential snapshot found in the previous step, Its* projection network*, is defined as the* graph *obtained from J by keeping terminal vertices intact, and replacing every terminal to terminal path in J by a path of length $2 - p$.*

The type of the projection network is the type of the corresponding potential snapshot.
The set of projection networks *of G_m, denoted by $\mathcal{N}(G_m)$, is the set of all projection networks obtained for G_m by the above procedure.*

See Figure 7 for an example. Since the total number of terminals is at most 4 (with one fixed as t), it is easy to see that the number of possible projections networks for G_m is bounded. Therefore $\mathcal{N}(G_m)$ can be computed in polynomial time. Note that $\mathcal{N}(G_m)$ is not uniquely defined. But it is sufficient for our purpose, to compute any one of the various possible choices of the set $\mathcal{N}(G_m)$ as explained in Figure 7. The next lemma shows that the projection networks of G_m preserve solutions, and also maintain invariants on planarity and treewidth, when merged with the parent piece.

▶ **Lemma 12.** *Given $G' = G_1 + \ldots + G_m$ as described above.*

1. *Given a $\mathcal{N}(G_m)$, there is an s-t path in G' of parity p iff $\exists N \in \mathcal{N}(G_m)$ such that $G'[G_m \to N]$ has an s-t path of parity p.*
2. *If G_{m-1} is planar/of bounded treewidth, then for any projection network $N \in \mathcal{N}(G_m)$, $G_{m-1} + N$ is planar/of bounded treewidth, respectively.*

Proof.

1. This follows from the definition of projection networks. The only minor technical point to note is that $\mathcal{N}(G_m)$ is not unique. For example, suppose there are two potential snaphots in G_m of type 4. One is J_1, consisting of a path P_1 from v_1 to v_2 of even parity, and a path P_2 from v_3 to t of odd parity. The other is J_2, consisting of a path P_1' from v_1 to v_2

[4] Note that the vertices of a clique, say c_i need no longer lie on the same face of G_i after phase I, since we might have merged the parity mimicking network of the branch pieces incident at c_i into the face corresponding to c_i.

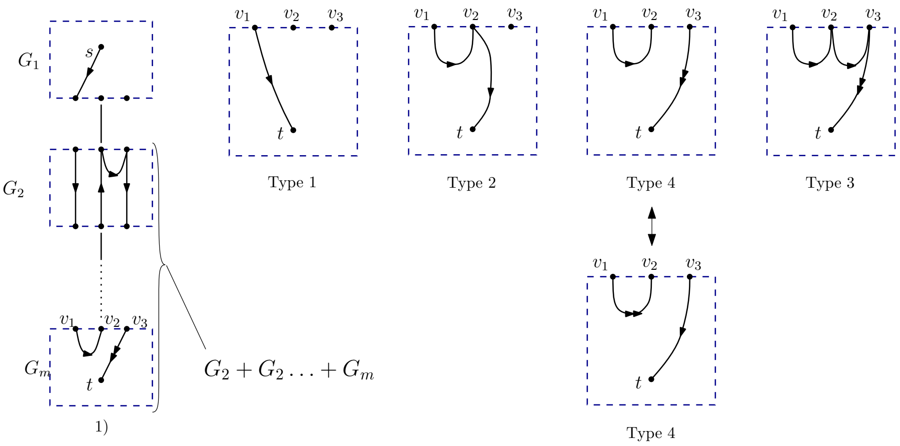


Figure 7 Fig 1) Denotes the decomposition tree after phase 1, with $G_1, G_2 \ldots, G_m$ denoting the pieces. We skip drawing clique nodes here. On the right are examples of projection networks of different types. In fig 1), the snapshot of the s-t path in G_m is of type 4. The two projection networks of type 4 drawn on the right have same total parity, but different parities of individual segments. It is sufficient for our purpose to find any one of them since they are interchangeable.

of odd parity, and a path P_2' from v_3 to t, of even parity. Since the total parity of J_1 and J_2 is same, Lemma 10 could output either one of them. We don't have control over it to find both. But finding any one of them is sufficient for us, since if J_1 is a snapshot of an actual solution, then replacing J_1 by J_2 would also give a valid solution and vice versa.(See Figure 7)

2. Suppose $c_{m_1} = \{v_1, v_2, v_3\}$ is the clique where G_{m-1}, G_m are attached. The argument of treewidth bound is same as that of Lemma 7 in previous phase, when we attached mimicking networks to parent pieces. However if G_{m-1} is planar, there could have been a parity mimicking network L' attached to G_{m-1} via c_{m-1} during phase I. Hence v_1, v_2, v_3 might not lie on a common face in G_{m-1} after phase I. We observe however, since L' was attached at a 3-clique, c_{m-1}, every pair v_i, v_j of vertices of c_{m-1}, must share a common face in G_{m-1}. Now, the projection networks consist of at most three paths, two between v_1, v_2, v_3, and one from them to t. For any v_i, v_j, we can embed the path between v_i, v_j in N, in the face in G_{m-1} shared by v_i, v_j, and finally just add the path leading to t. Therefore if G_{m-1} is planar, all projection networks of G_m can be embedded in their parent nodes. ◀

We make the following observation to compute a $\mathcal{N}(G_i + \ldots G_m)$ recursively:

$$\mathcal{N}(G_i + \ldots G_m) = \bigcup_{N \in \mathcal{N}(G_{i+1} + \ldots G_m)} \mathcal{N}(G_i + N) \tag{3}$$

Thus we can proceed as follows:

1. Compute $\mathcal{N}(G_m)$ using lemma 10
2. For all $N \in \mathcal{N}(G_m)$, compute $G_{m-1} + N$, and hence compute $\mathcal{N}(G_m + G_{m-1})$ using the observation above.
3. For all $N \in \mathcal{N}(G_m + G_{m-1})$, compute $G_{m-2} + N$, and hence compute $\mathcal{N}(G_m + G_{m-1} + G_{m-2})$. Repeat until we reach G_1.

References

1 Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in nc. *J. ACM*, 67(4), May 2020. doi:10.1145/3397504.

2 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $K_{3,3}$-free and K_5-free bipartite graphs. In *Symposium on Theoretical Aspects of Computer Science*, 2014. URL: https://api.semanticscholar.org/CorpusID:1484963.

3 Andreas Björklund, Thore Husfeldt, and Petteri Kaski. The shortest even cycle problem is tractable. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 117–130, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520030.

4 Glencora Borradaile and Philip Klein. An o(n log n) algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2), April 2009. doi:10.1145/1502793.1502798.

5 Diptarka Chakraborty and Raghunath Tewari. Simultaneous time-space upper bounds for red-blue path problem in planar dags. In M. Sohel Rahman and Etsuji Tomita, editors, *WALCOM: Algorithms and Computation - 9th International Workshop, WALCOM 2015, Dhaka, Bangladesh, February 26-28, 2015. Proceedings*, volume 8973 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2015. doi:10.1007/978-3-319-15612-5_23.

6 Erin Wolf Chambers and David Eppstein. Flows in one-crossing-minor-free graphs. *Journal of Graph Algorithms and Applications*, 17(3):201–220, 2013. doi:10.7155/jgaa.00291.

7 Shiva Chaudhuri, K. Subrahmanyam, Frank Wagner, and Christos Zaroliagis. Computing mimicking networks. *Algorithmica*, 26:31–49, January 2000. doi:10.1007/s004539910003.

8 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Marek Cygan, Daniel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 197–206, 2013. doi:10.1109/FOCS.2013.29.

10 Samir Datta, Arjun Gopalan, Raghav Kulkarni, and Raghunath Tewari. Improved bounds for bipartite matching on surfaces. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 254–265. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.254.

11 Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for k_{3, 3}-free and k_5-free graphs is in log-space. *Electron. Colloquium Comput. Complex.*, TR10, 2009. URL: https://api.semanticscholar.org/CorpusID:7978883.

12 Erik D Demaine, MohammadTaghi Hajiaghayi, Naomi Nishimura, Prabhakar Ragde, and Dimitrios M Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *Journal of Computer and System Sciences*, 69(2):166–195, 2004. doi:10.1016/j.jcss.2003.12.001.

13 David Eppstein and Vijay V. Vazirani. Nc algorithms for computing a perfect matching and a maximum flow in one-crossing-minor-free graphs. *SIAM Journal on Computing*, 50(3):1014–1033, 2021. doi:10.1137/19M1256221.

14 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

15 Anna Galluccio and Martin Loebl. Even/odd dipaths in planar digraphs. *Optimization Methods and Software*, 3(1-3):225–236, 1994. doi:10.1080/10556789408805566.

16 Martin Grohe, Ken-ichi Kawarabayashi, and Bruce Reed. A simple algorithm for the graph minor decomposition: logic meets structural graph theory. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 414–431, USA, 2013. Society for Industrial and Applied Mathematics.

17 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998. doi:10.1006/jcss.1998.1592.

18 Ken-ichi Kawarabayashi, Bruce Reed, and Paul Wollan. The graph minor algorithm with parity conditions. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 27–36, 2011. doi:10.1109/FOCS.2011.52.

19 Ken-ichi Kawarabayashi and Paul Wollan. A simpler algorithm and shorter proof for the graph minor decomposition. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 451–458, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1993636.1993697.

20 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014. doi:10.1016/j.ipl.2014.02.011.

21 Samir Khuller. Coloring algorithms for k5-minor free graphs. *Information Processing Letters*, 34(4):203–208, 1990. doi:10.1016/0020-0190(90)90161-P.

22 Samir Khuller. Extending planar graph algorithms to k3,3-free graphs. *Information and Computation*, 84(1):13–25, 1990. doi:10.1016/0890-5401(90)90031-C.

23 Robert Krauthgamer and Inbal Rika. *Mimicking Networks and Succinct Representations of Terminal Cuts*, pages 1789–1799. SIAM, 2013. doi:10.1137/1.9781611973105.128.

24 Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984. doi:10.1002/net.3230140403.

25 Daniel Lokshtanov, Pranabendu Misra, Michał Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1307–1316, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384250.

26 James F. Lynch. The equivalence of theorem proving and the interconnection problem. *SIGDA Newsl.*, 5(3):31–36, September 1975. doi:10.1145/1061425.1061430.

27 William McCuaig, Neil Robertson, P. D. Seymour, and Robin Thomas. Permanents, pfaffian orientations, and even directed circuits (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 402–405, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258625.

28 Zhivko Prodanov Nedev. Finding an even simple path in a directed planar graph. *SIAM J. Comput.*, 29(2):685–695, 1999. doi:10.1137/S0097539797330343.

29 Bruce Reed, Neil Robertson, Alexander Schrijver, and Paul Seymour. Finding dsjoint trees in planar graphs in linear time. In *Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washington, Seattle, USA*, volume 147 of *Contemporary Mathematics*, pages 295–302, January 1991. doi:10.1090/conm/147/01180.

30 N. Robertson and P.D. Seymour. Excluding a graph with one crossing. *Graph struc- ture theory (Seattle, WA, 1991)*, 1993. doi:10.1090/conm/147.

31 N. Robertson and P.D. Seymour. Graph minors .xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

32 Neil Robertson and P.D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. doi:10.1016/S0095-8956(03)00042-X.

33 Neil Robertson and P.D. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. Special Issue Dedicated to Professor W.T. Tutte. doi:10.1016/j.jctb.2004.08.001.

34 Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM J. Comput.*, 23(4):780–788, 1994. doi:10.1137/S0097539792224061.

35 Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in k5-free graphs. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 66–77, 2014. doi:10.1109/CCC.2014.15.

36 Thomas Thierauf and Fabian Wagner. Reachability in k3,3-free graphs and k5-free graphs is in unambiguous log-space. In Mirosław Kutyłowski, Witold Charatonik, and Maciej Gębala, editors, *Fundamentals of Computation Theory*, pages 323–334, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

37 Vijay V. Vazirani. Nc algorithms for computing the number of perfect matchings in k3,3-free graphs and related problems. *Information and Computation*, 80(2):152–164, 1989. doi:10.1016/0890-5401(89)90017-5.

38 Klaus Von Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937. URL: https://api.semanticscholar.org/CorpusID:123534907.

39 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997.

The Freeness Problem for Automaton Semigroups

Daniele D'Angeli
Università degli Studi Niccolò Cusano, Roma, Italy

Emanuele Rodaro
Department of Mathematics, Politecnico di Milano, Italy

Jan Philipp Wächter
Fachrichtung Mathematik, Universität des Saarlandes, Saarbrücken, Germany

Abstract

We present a new technique to encode Post's Correspondence Problem into automaton semigroups and monoids. The encoding allows us to precisely control whether there exists a relation in the generated semigroup/monoid and thus show that the freeness problems for automaton semigroups and for automaton monoids (listed as open problems by Grigorchuk, Nekrashevych and Sushchanskĭi) are undecidable. The construction seems to be quite versatile and we obtain the undecidability of further problems: Is a given automaton semigroup (monoid) (left) cancellative? Is it equidivisible (which – together with the existence of a (proper) length function – characterizes free semigroups and monoids)? Does a given map extend into a homomorphism between given automaton semigroups? Finally, our construction can be adapted to show that it is undecidable whether a given automaton generates a free monoid whose basis is given by the states (but where we allow one state to act as the identity). In the semigroup case, we show a weaker version of this.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases Automaton Monoid, Automaton Semigroup, Freeness Problem, Free Presentation

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.44

Related Version *Full Version*: https://arxiv.org/abs/2402.01372

Funding *Jan Philipp Wächter*: funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 492814705 – while visiting the Department of Mathematics at Politecnico di Milano. The listed affiliation is his current one, partly funded by ERC grant 101097307.

1 Introduction

In the 1980s, Grigorchuk solved a famous question by Milnor (see [20] for a nice introduction) by presenting the first group with intermediate growth: the number of elements that can be written as a word of length at most n over the generators grows sub-exponentially but super-polynomially. The group has even more noteworthy properties. It is amenable but not elementary amenable (e. g. [24]) and an infinite 2-group (giving a counter-example to Burnside's problem, e. g. [33, 3]). Its peculiar properties stirred interest in Grigorchuk's group and groups of similar form where it soon became important that Grigorchuk's group has a nice description using what is simply called an *automaton* in this context (e. g. [33] or [3]). The simplicity of this presentation (the automaton only uses a binary alphabet and four states – with an additional identity state) contrasts the complex nature of the group. An "automaton" here is what more precisely is called a finite-state letter-to-letter transducer (i. e. an automaton with input and output). The idea is that in such an automaton every state induces a mapping of input to output words and the closure of these functions under composition forms a semigroup. If the automaton is additionally invertible, the functions are bijections and we may consider the generated group. This leads to the classes of *automaton semigroups* and *groups*, which contain further noteworthy examples (e. g. Gupta-Siki p-groups [22], the lamplighter group [21] and more general lamplighter-like groups [37, 38]).

Being able to finitely describe groups without classical finite presentations (consisting of generators and relations) additionally highlights the usefulness of considering (semi)groups generated by automata. Starting from Grigorchuk's group, the study of automaton groups and semigroups is nowadays a thriving research field with important connections to many neighboring areas (such as geometry, dynamical systems and symbolic dynamics; see e. g. [33, 3] for more background information). The extensive research in Mathematics and Computer Science on the semigroup (and monoid) case (e. g. [9, 26, 7, 34, 1, 15]) arises naturally from the group case for example via the *dual automaton* where states and input/output letters swap places. The connection between an automaton and its dual has been exploited algebraically and algorithmically (e. g. [18, 41, 42, 26, 27, 11]).

In this work, we look further at the algorithmic aspects of this interesting class by showing that its *freeness problem* is undecidable. This problem asks whether a given automaton generates a free semigroup (or monoid). It has been studied extensively for other classes of groups and semigroups. Since freeness is a Markov property, the problem is undecidable for classical finite group (and, thus, semigroup) presentations (see e. g. [29]). Further important results include the undecidability of the freeness problem for matrix semigroups, originally shown using a reduction from Post's Correspondence Problem [25], which has been improved and contrasted in many further publications (e. g. [31, 10, 4]). Interestingly, matrix (semi)groups and automaton (semi)groups are connected in the sense that the former can be presented as subgroups of the latter [8] (see also [40, 12, 43]) but this does not help to prove the freeness problem undecidable for automaton (semi)groups [13].

With our result, we continue this line of research but also further contribute to the study of freeness in self-similar (i. e. generated by infinite automata) and automaton structures as well as their algorithmic aspects. For the former, we refer the reader to the survey [36] and only point out that, while it is known that free groups are automaton groups [41, 42, 39], these constructions are usually deemed rather difficult. For automaton semigroups and monoids, the situation seems to be simpler: every free semigroup of (finite) rank at least two can be generated by an automaton (see [9] or Example 2.5) but the free semigroup of rank one cannot [9]. All free monoids of finite rank are automaton semigroups, though.

Regarding algorithmic questions for automaton (semi)groups, we point out that, while one may easily be misled into believing that using a finite automaton as the generating combinatorial object should be rather simple, the situation is actually quite complex and only a few natural algorithmic problems are known to be undecidable while many others notoriously remain open problems. An exception here seems to be that the word problem for automaton (semi)groups is PSPACE-complete. Interestingly, this was first known for semigroups [14] and was later extended to groups [43]. Some subclasses have simpler word problems. For example, using finitary automata to present finite groups results in a coNP-complete word problem [28] and the word problem of an automaton group of polynomial activity is in polylogarithmic space [5] (see [44] for more information). On the other hand, there is an automaton group with an undecidable conjugacy problem [40] ("are two given group elements conjugate in the group?"). The construction used there also shows that the isomorphism problem for automaton groups ("are the groups generated by two given automata isomorphic?") and, thus, automaton semigroups is undecidable.[1] There are two constructions for an automaton group with undecidable order problem ("has a given group

[1] Unfortunately, this does not seem to be written down explicitly anywhere.

element finite or infinite order?") [17, 2]. The latter of the two even yields a contracting automaton. The undecidability was also first known for automaton semigroups [16] and the problem is decidable for bounded automaton groups [6] and monoids [1].

All these constructions encoding Turing machines in automaton (semi)groups make a statement about individual (semi)group elements. Since the interaction between the generating automaton and generated algebraic structure is often surprising and still not well understood, it is much more challenging to construct reductions where the entire generated (semi)group (or monoid) has a certain property (based on whether we input a positive or negative problem instance). The only known result of this kind seems to be that the finiteness problem for automaton semigroups ("Is the semigroup generated by a given automaton finite?") is undecidable [16]. The corresponding group problem is still open [19].

Our reduction from Post's Correspondence Problem [35] to the freeness problems for automaton semigroups and for monoids in this paper is a second result of this form. It solves the corresponding open problem by Grigorchuk, Nekrashevych and Sushchanskiĭ [19, 7.2 b)] and, despite previous attempts [12, 13] and a positive result for semigroups generated by invertible and reversible automata with two states [26] as well as a negative result on testing for relations of the form $w = \mathbb{1}$ [12], the problem had remained open quite a while for groups and for semigroups. The main challenge seems to be that we need very precise control over the relations in the generated semigroup (which seems to be much more difficult than, e. g., ensuring that the semigroup is finite or infinite) while the interaction between the structure of the generating automaton and the semigroup/monoid relations is highly non-obvious.

Our construction yields further results beyond the freeness problem(s). Namely, testing whether a given automaton generates a (left) cancellative semigroup/monoid and whether the semigroup/monoid generated by a given automaton is equidivisible (a notion strongly related to freeness by Levi's lemma, see Fact 2.2) are undecidable. We also obtain that it is undecidable whether a given automaton generates a free semigroup with a given basis and whether a given map between the state sets of two given automata can be extended into an iso- or homomorphism. The latter problem is connected to the (undecidable, see above) isomorphism problem for automaton semigroups in the sense that it asks whether all relations of the first automaton semigroup also hold in the second one.

Finally, the construction seems to be flexible enough to be adapted to similar problems, which gives us hope that our results could also contribute towards showing that the freeness problem is undecidable in the group case. For example, it can be adapted to show that the free presentation problem for automaton monoids is undecidable: does a given automaton generate a free monoid whose rank is equal to the number of its states (minus an identity state)? In other words, we cannot test whether a given automaton monoid contains any relations (although this is semi-decidable as the word problem is decidable, see above).

Adapting our construction for this is necessary because the construction in the semigroup case always yields semigroup relations since we need to use a result on the closure of the class of automaton semigroups under (certain) free products [30] in order to construct some kind of "partial" powers of the generating automaton. However, no details of this construction will be required to understand our results. More generally, the presentation in this work is meant to be self-contained (although the construction may be considered to be rather technical).

2 Preliminaries

Fundamentals, Semigroups and Monoids. We write $A \uplus B$ for the disjoint union of sets and consider the set of natural numbers $\mathbb{N}$ to contain 0. We assume the reader to be familiar with fundamental notions of semigroup theory (see e. g. [23]). We write $\mathbb{1}_M$ for the neutral element of a monoid M or, if M is clear from the context, simply $\mathbb{1}$. For a monoid M, we let $M^{\mathbb{1}} = M$ and, if S is a semigroup but not a monoid, we may adjoin a neutral element $\mathbb{1} \notin S$ to S by letting $\mathbb{1}\mathbb{1} = \mathbb{1}$ and $\mathbb{1}s = s = s\mathbb{1}$ for all S and denote the resulting monoid by $S^{\mathbb{1}}$.

Words, Free Semigroups and Free Monoids. Let B be a finite, non-empty set, which we call an *alphabet.* A *word* w over B is a finite sequence $a_1 \dots a_n$ with $a_1, \dots, a_n \in B$, whose *length* is $|w| = n$. We denote the unique word of length 0 (i. e. the *empty word*) by ε. The set of all words over B is denoted by B^*. Words have the natural operation of juxtaposition (where we let $uv = a_1 \dots a_m b_1 \dots b_n$ for $u = a_1 \dots a_m$ and $v = b_1 \dots b_n$ with $a_1, \dots, a_m, b_1, \dots, b_n \in B$), which turns B^* into a monoid with the neutral element ε. This monoid B^* is *the free monoid* with *basis* B (or *over* B) and a monoid M is *free* (with basis B) if it is isomorphic to B^* (for some alphabet B). Closely related to the free monoid is *the free semigroup* B^+, which is formed by the set of all non-empty words (i. e. $B^+ = B^* \setminus \{\varepsilon\}$) and (again) juxtaposition as operation. Similarly, a semigroup S is *free* (with basis B) if it is isomorphic to B^+ (for some alphabet B). Note that B^* is (isomorphic to) $(B^+)^{\mathbb{1}}$ and that the basis of a free monoid or semigroup is unique (see e. g. [23, Proposition 7.1.3]). The *rank* of a free monoid or semigroup is the cardinality $|B|$ of its basis B. We will use common conventions from formal language theory and, e. g., write q^+ and q^* for $\{q\}^+$ and $\{q\}^*$.

Properties of Free Semigroups and Monoids. We will need some properties of free semigroups and monoids. A (general) semigroup S is *left cancellative* if $st = st'$ implies $t = t'$ for all $s, t, t' \in S$. Symmetrically, it is *right cancellative* if $st = s't$ implies $s = s'$ for all $s, s', t \in S$ and, finally, it is *cancellative* if it is both left and right cancellative. It is easy to see that B^* and, thus, B^+ are cancellative (see, e. g. [23, Proposition 7.1.1]).

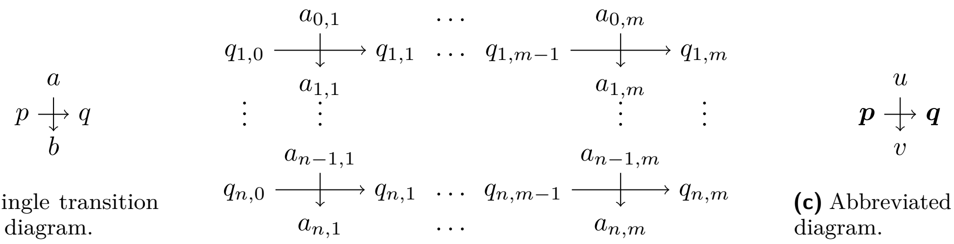
▶ **Fact 2.1.** *Free semigroups and free monoids are cancellative.*

A *length function* of a semigroup S is a homomorphism $S \to \mathbb{N}_{>0}$ where $\mathbb{N}_{>0}$ is the additive semigroup of strictly positive natural numbers. A monoid M has a *proper length function* if there is a monoid homomorphism $M \to \mathbb{N}$ (where $\mathbb{N}$ is the additive monoid of the natural numbers including 0) such that $\mathbb{1}$ is the only pre-image of 0 (i. e. only $\mathbb{1}$ has length 0, all other elements have strictly positive length). A semigroup S that is not a monoid has a length function if and only if $S^{\mathbb{1}}$ has a proper one and free semigroups and monoids do have (proper) length functions (mapping a word to its length).

A semigroup (or monoid) S is *equidivisable* if, for all $s_1, s_2, s_1', s_2' \in S$ with $s_1 s_2 = s_1' s_2'$, there is some $x \in S^{\mathbb{1}}$ with $s_1 = s_1' x$ and $x s_2 = s_2'$ or with $s_1 x = s_1'$ and $s_2 = x s_2'$. It is not difficult to see that free semigroups and monoids are equidivisible (see e. g. [23, Proposition 7.1.2]). Together with having a (proper) length function, this turns out to characterize free semigroups and monoids (see e. g. [23, Proposition 7.1.8]).

▶ **Fact 2.2** (Levi's Lemma). *A semigroup (monoid) S is free if and only if it is equidivisible and has a (proper) length function.*

Free Products of Semigroups. A *semigroup presentation* is a pair $\langle Q \mid \mathcal{R} \rangle_{\mathscr{S}}$ of a set of *generators* Q and a (possibly infinite) set of *relations* $\mathcal{R} \subseteq Q^+ \times Q^+$. We will only consider presentations where Q is finite and non-empty. If we denote by $\mathcal{C}$ the smallest congruence $\mathcal{C} \subseteq Q^+ \times Q^+$ with $\mathcal{R} \subseteq \mathcal{C}$, the semigroup *presented* by such a presentation is $S = Q^+/\mathcal{C}$ formed by the congruence classes $[\cdot]$ of $\mathcal{C}$ with the (well-defined!) operation $[u] \cdot [v] = [uv]$. Every semigroup generated by a finite, non-empty set Q is presented by some semigroup presentation of this form.



(a) Single transition cross diagram.

(b) Multiple crosses combined in one diagram.

(c) Abbreviated cross diagram.

Figure 1 Combined and abbreviated cross diagrams.

The free product of the semigroups $S = \langle Q \mid \mathcal{S} \rangle_{\mathscr{S}}$ and $T = \langle P \mid \mathcal{R} \rangle_{\mathscr{S}}$ is the semigroup $S \star T = \langle Q \uplus P \mid \mathcal{S} \cup \mathcal{R} \rangle_{\mathscr{S}}$. For example, we have $\{p, q\}^+ = p^+ \star q^+$.

▶ Remark. Of course, there is also the free product of monoids (and monoid presentations). However, we will only consider free products of semigroups (in particular: $\{p, q\}^* \not\simeq p^* \star q^*$).

Automata. In the current context, an *automaton* is a triple $\mathcal{T} = (Q, \Sigma, \delta)$ consisting of a non-empty, finite set of *states* Q, an *alphabet* Σ and a set $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$ of *transitions*.

▶ Remark. What we simply call an automaton here would rather be called a finite-state, letter-to-letter transducer in more general automaton-theoretic terms. However, simply using the term "automaton" is standard terminology in the area. We also do not use initial or final states as they do not interact nicely with the self-similar nature of the semigroups and monoids generated by automata we are about to define.

For transitions, we will use the graphical notation $p \xrightarrow{a/b} q$ to denote $(p, a, b, q) \in Q \times \Sigma \times \Sigma \times Q$. Such a transition *starts* in p, *ends* in q, its *input* is a and its *output* is b. This reflects the common way of depicting automata (see e. g. Figure 2). When dealing with an automaton $\mathcal{T} = (Q, \Sigma, \delta)$, we are actually dealing with two alphabets (Q and Σ). In order to avoid confusion, we call the elements of Q *states* and the elements of Q^* *state sequences*, while reserving the terms *letters* and *words* for the elements of Σ and Σ^*, respectively.

Another somewhat graphical tool that we will make heavy use of are *cross diagrams*. Here, a cross diagram as given in Figure 1a indicates the existence of a transition $p \xrightarrow{a/b} q$ in the automaton. Cross diagrams can be stacked together in order to create larger ones. For example, the diagram in Figure 1b indicates the existence of the transition $q_{i,j-1} \xrightarrow{a_{i-1,j}/a_{i,j}} q_{i,j}$ for all $0 < i \leq n$ and $0 < j \leq m$. When combining cross diagrams, we will sometimes omit unnecessary states and letters. Additionally, we will also abbreviate them: for example, if we let $\boldsymbol{p} = q_{n,0} \dots q_{1,0}$, $u = a_{0,1} \dots a_{0,m}$, $v = a_{n,1} \dots a_{n,m}$ and $\boldsymbol{q} = q_{n,m} \dots q_{1,m}$, the cross diagram in Figure 1c is an abbreviation of the cross diagram in Figure 1b. It is important here to note the order we write the state sequences in: in our example, $q_{1,0}$ is the first state in the top left of the cross diagram but it is the rightmost state in the sequence $\boldsymbol{p}$. This order will later be more natural as we will define a left action based on cross diagrams.

An automaton $\mathcal{T} = (Q, \Sigma, \delta)$ is called *complete and deterministic* if, for every $p \in Q$ and every $a \in \Sigma$, there is exactly one $q \in Q$ and exactly one $b \in \Sigma$ such that the cross diagram in Figure 1a holds (i. e. in every state p and for every letter $a \in \Sigma$, there is exactly one transition starting in p with input a). We call such an automaton a *complete $\mathscr{S}$-automaton* (as they naturally generate semigroups).

An automaton $\mathcal{S} = (P, \Sigma, \sigma)$ is a *subautomaton* of another automaton $\mathcal{T} = (Q, \Gamma, \delta)$ if $P \subseteq Q$, $\Sigma \subseteq \Gamma$ and $\sigma \subseteq \delta$. In this case, any cross diagram of $\mathcal{S}$ is also valid for $\mathcal{T}$.

Automaton Semigroups and Monoids. Let $\mathcal{T} = (Q, \Sigma, \delta)$ be a complete $\mathscr{S}$-automaton. By induction, there is exactly one $v \in \Sigma^+$ and exactly one $\boldsymbol{q} \in Q^+$ for every $\boldsymbol{p} \in Q^+$ and $u \in \Sigma^+$ such that the cross diagram in Figure 1c holds. This allows us to define a left action of Q^+ on Σ^+ by letting $\boldsymbol{p} \circ u = v$ and to define a right action of Σ^+ on Q^+, called the *dual action*, by letting $\boldsymbol{p} \cdot u = \boldsymbol{q}$. The reader may verify that this indeed defines well-defined actions by the way cross diagrams work. We may extend these into an action of Q^* on Σ^* and an action of Σ^* on Q^* by letting $\varepsilon \circ u = u$ for all $u \in \Sigma^*$, $\boldsymbol{p} \circ \varepsilon = \varepsilon$ for all $\boldsymbol{p} \in Q^*$, $\varepsilon \cdot u = \varepsilon$ again for all $u \in \Sigma^*$ and, finally, $\boldsymbol{p} \cdot \varepsilon = \boldsymbol{p}$ for (again) all $\boldsymbol{p} \in Q^*$.

By the way cross diagrams work, there is an interaction between the two actions: for all $\boldsymbol{p}, \boldsymbol{q} \in Q^*$ and all $u, v \in \Sigma^*$, we have $\boldsymbol{p} \circ uv = (\boldsymbol{p} \circ u)[(\boldsymbol{p} \cdot u) \circ v]$ and $\boldsymbol{qp} \cdot u = [\boldsymbol{q} \cdot (\boldsymbol{p} \circ u)](\boldsymbol{p} \cdot u)$.

The action $\boldsymbol{p} \circ u$ allows us to define the congruence $=_\mathcal{T} \subseteq Q^* \times Q^*$ by $\boldsymbol{p} =_\mathcal{T} \boldsymbol{q} \iff \forall u \in \Sigma^* : \boldsymbol{p} \circ u = \boldsymbol{q} \circ u$. We denote the class of $\boldsymbol{p} \in Q^*$ with respect to $=_\mathcal{T}$ by $[\boldsymbol{p}]_\mathcal{T}$. The set $\mathscr{M}(\mathcal{T}) = Q^*/{=_\mathcal{T}}$ of these classes forms a monoid, which is called the *monoid generated* by $\mathcal{T}$. In other words, it is the faithful quotient of Q^* with respect to the action $\boldsymbol{q} \circ u$. Note that ε acts like the identity on all $u \in \Sigma^*$ and the class of ε, thus, forms the neutral element of $\mathscr{M}(\mathcal{T})$. A monoid arising in this way is called a *complete automaton monoid*.

Similarly, the *semigroup generated* by $\mathcal{T}$ is the semigroup $\mathscr{S}(\mathcal{T}) = Q^+/{=_\mathcal{T}}$ and any such semigroup is a *complete automaton semigroup*. Note that monoid and semigroup generated by a complete $\mathscr{S}$-automaton coincide if there is a non-empty state sequence acting trivially.

▶ Remark 2.3. We only consider complete $\mathscr{S}$-automata in this work but will make this explicit by talking about complete $\mathscr{S}$-automata and complete automaton semigroups and monoids. In the literature, these objects are often simply called "automaton semigroups" (the term "automaton monoid" is less common). This is a convention that we could also follow here but choose not to since the concepts generalize naturally also to non-complete automata, yielding (partial) automaton semigroups and monoids. It is not known whether the two classes coincide (see [15] for more details).

▶ Remark 2.4. There is a subtle difference between an automaton monoid and an automaton semigroup which happens to be a monoid. In the latter, the neutral element not necessarily acts as the identity map. In fact, it is not known whether the two classes coincide (see [9, Proposition 3.1] for the analogue for groups).

Free Semigroups (Monoids) as Automaton Semigroups (Monoids). As examples of complete automaton semigroups and monoids, we will next look at how to generate free semigroups and monoids. The free monoid of rank one is generated by an automaton known as the *adding machine* (see e. g. [33] or [3]), which turns it into both a complete automaton monoid and a complete automaton semigroup. The free semigroup of rank one, on the other hand, is neither [9, Proposition 4.3] (see also [7, Theorem 15], [15, Theorem 19] and [43, Theorem 1.2.1.4]).

However, free semigroups of higher rank (and their monoid counter-parts) are indeed complete automaton semigroups [9, Proposition 4.1]:

▶ **Example 2.5.** Let R be a finite set with $|R| \geq 2$. Consider the automaton $\mathcal{R} = (R, R, \rho)$ with $\rho = \{a \xrightarrow{b/a} b \mid a, b \in R\}$ (see Figure 2 for the binary case). One easily verifies that $\mathcal{R}$ is a complete $\mathscr{S}$-automaton and we claim that it generates R^+. For this, it suffices to show that, for every $\boldsymbol{p}, \boldsymbol{q} \in R^+$ with $\boldsymbol{p} \neq \boldsymbol{q}$, there is some $u \in R^*$ with $\boldsymbol{p} \circ u \neq \boldsymbol{q} \circ u$. We may assume $|\boldsymbol{p}| \geq |\boldsymbol{q}|$ and there needs to be some $a \in R$ with $\boldsymbol{p} \neq \boldsymbol{q}a^{|\boldsymbol{p}|-|\boldsymbol{q}|}$ (we just need to take a different to the last letter of $\boldsymbol{p}$ if the lengths differ). Now, observe that, for all $n \geq 1$ and

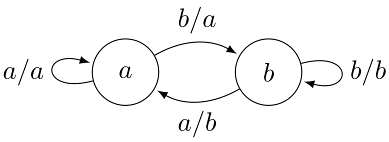


Figure 2 A complete $\mathscr{S}$-automaton generating $\{a, b\}^+$.

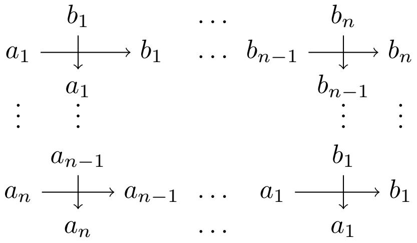


Figure 3 Cross diagram of $\mathcal{R}$.

all $a_1, \dots, a_n, b_1, \dots, b_n \in R$, we have the cross diagram in Figure 3 by the construction of $\mathcal{R}$. This shows, in particular, $\boldsymbol{p} \circ a^{|\boldsymbol{p}|} = \boldsymbol{p}$ and $\boldsymbol{p} \cdot a^{|\boldsymbol{p}|} = a^{|\boldsymbol{p}|}$. By a similar cross diagram, we obtain $\boldsymbol{p} \neq_{\mathcal{R}} \boldsymbol{q}$ (since $\boldsymbol{q} \circ a^{|\boldsymbol{p}|} = (\boldsymbol{q} \circ a^{|\boldsymbol{q}|})(a^{|\boldsymbol{q}|} \circ a^{|\boldsymbol{p}|-|\boldsymbol{q}|}) = \boldsymbol{q}a^{|\boldsymbol{p}|-|\boldsymbol{q}|} \neq \boldsymbol{p} = \boldsymbol{p} \circ a^{|\boldsymbol{p}|}$).

There is no state sequence which acts like the identity and this means that $\mathscr{M}(\mathcal{R})$ is $\mathscr{S}(\mathcal{R})^{\mathbb{1}} \simeq R^*$, which shows that R^* is a complete automaton monoid.

The construction presented in Example 2.5 is clearly computable and we obtain:

▶ **Fact 2.6.** *For every finite set R with $|R| \geq 2$, one can compute an $\mathscr{S}$-automaton $\mathcal{R} = (R, R, \rho)$ with $\mathscr{S}(\mathcal{R}) \simeq R^+$ and $\mathscr{M}(\mathcal{R}) \simeq R^*$.*

Automaton Operations. The *union* of two automata $\mathcal{T}_1 = (Q_1, \Sigma_1, \delta_1)$ and $\mathcal{T}_2 = (Q_2, \Sigma_2, \delta_2)$ is the automaton $\mathcal{T}_1 \cup \mathcal{T}_2 = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2)$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are both complete $\mathscr{S}$-automaton with non-intersecting state sets ($Q_1 \cap Q_2 = \emptyset$) but a common alphabet $\Sigma_1 = \Sigma_2$, their union $\mathcal{T}_1 \cup \mathcal{T}_2$ is also a complete $\mathscr{S}$-automaton (which allows us, for example, to consider the semigroup $\mathscr{S}(\mathcal{T}_1 \cup \mathcal{T}_2)$). Similarly, the union of two complete $\mathscr{S}$-automata with the same state set but disjoint alphabets is again a complete $\mathscr{S}$-automaton. This operation basically adds the transitions of $\mathcal{T}_2$ to the existing transitions of $\mathcal{T}_1$.

The *composition* of two automata $\mathcal{T}_2 = (Q_2, \Sigma, \delta_2)$ and $\mathcal{T}_1 = (Q_1, \Sigma, \delta_1)$ over a common alphabet Σ is the automaton $\mathcal{T}_2 \circ \mathcal{T}_1 = (Q_2Q_1, \Sigma, \delta_2 \circ \delta_1)$ with

$$\delta_2 \circ \delta_1 = \left\{ p_2p_1 \xrightarrow{a/c} q_2q_1 \,\middle|\, \exists b \in \Sigma : p_1 \xrightarrow{a/b} q_1 \in \delta_1 \text{ and } p_2 \xrightarrow{b/c} q_2 \in \delta_2 \right\}$$

(where $Q_2Q_1 = \{q_2q_1 \mid q_1 \in Q_1, q_2 \in Q_2\}$ is the cartesian product of Q_2 and Q_1). If $\mathcal{T}_2$ and $\mathcal{T}_1$ are complete $\mathscr{S}$-automata, also their composition is.

The *k-th power* $\mathcal{T}^k$ of an automaton $\mathcal{T}$ is the k-fold composition of $\mathcal{T}$ with itself. It is computable and, if $\mathcal{T}$ (and, thus, $\mathcal{T}^k$) is a complete $\mathscr{S}$-automaton, the actions of some $\boldsymbol{p} \in Q^*$ of length $|\boldsymbol{p}| = k$ seen as a state of $\mathcal{T}^k$ or seen as a state sequence over $\mathcal{T}$ coincide. Thus (and by an analogue for the dual action), the notations $\boldsymbol{p} \circ u$ and $\boldsymbol{p} \cdot u$ remain unambiguous and we have $\mathscr{S}(\mathcal{T}) = \mathscr{S}(\mathcal{T} \cup \mathcal{T}^k)$ for all $k \geq 1$, which is usually used to ensure that any fixed state sequence $\boldsymbol{p} \in Q^+$ may be assumed to be congruent to a single state under $=_{\mathcal{T}}$ (i. e. equal in the semigroup or monoid).

Finally, the *dual* of an automaton $\mathcal{T} = (Q, \Sigma, \delta)$ is the automaton $\partial\mathcal{T} = (\Sigma, Q, \partial\delta)$ with $\partial\delta = \left\{ a \xrightarrow{p/q} b \,\middle|\, p \xrightarrow{a/b} q \in \delta \right\}$ (i. e. we swap the roles of the states Q and the letters Σ). Clearly, the dual of a complete $\mathscr{S}$-automaton is again a complete $\mathscr{S}$-automaton.

The dual automaton can make it sometimes more accessible to understand how a letter is transformed by a state sequence: we just have to follow a path in the graphical representation of the dual automaton. For example, from Figure 5, it is obvious that the only way for $\boldsymbol{p} \circ \alpha = \boldsymbol{q} \circ \beta$ to hold is for both of them to be equal to f.

Adding Free Generators. For our results, we will need to add new free generators to existing automaton semigroups S computationally (in the sense that we do not change the behavior of existing state sequences but add a new state q such that the new automaton generates the (semigroup) free product $S \star q^+$). More precisely, we will use the following statement, which follows from the construction used for [30, Theorem 13].

▶ **Proposition 2.7.** *On input of a complete $\mathscr{S}$-automaton $\mathcal{S} = (P, \Sigma, \sigma)$, one can compute a complete $\mathscr{S}$-automaton $\mathcal{T} = (Q, \Gamma, \delta)$ with $Q = P \uplus \{q\}$ such that the identity on Q extends into a well-defined isomorphism $\mathscr{S}(\mathcal{T}) \to \mathscr{S}(\mathcal{S}) \star q^+$ (for the free product of semigroups).*

3 The Freeness Problem for Semigroups

We reduce Post's Correspondence Problem[2] PCP

Constant: an alphabet Λ
Input: homomorphisms $\varphi, \psi : I = \{1, \dots, n\} \to \Lambda^+$
Question: $\exists \boldsymbol{i} \in I^+ : \varphi(\boldsymbol{i}) = \psi(\boldsymbol{i})$?

to (the complement of) the freeness problem for automaton semigroups. For this, we fix an instance φ, ψ, I for PCP[3] over an alphabet Λ and describe how to map it to a complete $\mathscr{S}$-automaton $\mathcal{T} = (Q, \Sigma, \delta)$ in such a way that $\mathcal{T}$ can be computed and the PCP instance has a solution if and only if $\mathscr{S}(\mathcal{T})$ is **not** a free semigroup.

Starting from the free semigroup, we will construct $\mathcal{T}$ (in steps) such that the semigroup has a relation $\#_1 \boldsymbol{i} \#_1 =_{\mathcal{T}} \#_1 \boldsymbol{i} \#_2$ for $\boldsymbol{i} \in I^+$ if and only if $\boldsymbol{i}$ belongs to a PCP solution (if there is no solution, $\mathscr{S}(\mathcal{T})$ is free). Throughout this process, the reader may find it convenient to refer to Table 1 for the various symbols we are going to use.

The rough idea is to add an input symbol ι whose dual action turns $\boldsymbol{i}\#_1$ into $\varphi(\boldsymbol{i})$ and $\boldsymbol{i}\#_2$ into $\psi(\boldsymbol{i})$. But we also have to be careful not to introduce any unwanted relations and to keep the underlying free semigroup structure intact.

Without loss of generality, we may assume $|I| = n \geq 1$, $|\Lambda| \geq 2$ and $I \cap \Lambda = \emptyset$. In the following, we let $L = \max\{|\varphi(i)|, |\psi(i)| \mid i \in I\}$, $\hat{\Lambda} = \cup_{\ell=1}^{L} \Lambda^\ell$, $R = \Lambda \cup I$ and $\hat{R} = \hat{\Lambda} \cup I$.

Definition of $\hat{\mathcal{R}}$. First, we compute a complete $\mathscr{S}$-automaton $\hat{\mathcal{R}}$ with state set $\hat{R}$ generating the free semigroup over R:

▶ **Proposition 3.1.** *On input I, Λ and L, one can compute a complete $\mathscr{S}$-automaton $\hat{\mathcal{R}} = (\hat{R}, \Gamma, \rho)$ with state set $\hat{R} = \hat{\Lambda} \cup I$ (for $\hat{\Lambda} = \cup_{\ell=1}^{L} \Lambda^\ell$) and $\mathscr{S}(\hat{\mathcal{R}}) \simeq R^+ = (\Lambda \cup I)^+$ (where the isomorphism is given by $\hat{\lambda} \mapsto \hat{\lambda}$ for all $\hat{\lambda} \in \hat{\Lambda}$ and $i \mapsto i$ for all $i \in I$).*

Proof. Let $\mathcal{R}_1$ be an $\mathscr{S}$-automaton with state set Λ generating the free semigroup Λ^+ (see Fact 2.6) and let $\hat{\mathcal{R}}_1 = \bigcup_{\ell=1}^{L} \mathcal{R}_1^\ell$ be the union of the first L powers of $\mathcal{R}_1$. Note that the state set of $\hat{\mathcal{R}}_1$ is $\hat{\Lambda} = \cup_{\ell=1}^{L} \Lambda^\ell$ and that we still have $\mathscr{S}(\hat{\mathcal{R}}_1) \simeq \Lambda^+$ (where an isomorphism is induced by $\hat{\Lambda} \ni \hat{\lambda} \mapsto \hat{\lambda} \in \Lambda^+$). Now, we may apply Proposition 2.7 sequentially for every element of $I = \{1, \dots, n\}$, which yields the sought automaton $\hat{\mathcal{R}}$ with state set $\hat{R} = \hat{\Lambda} \cup I$ whose generated semigroup is isomorphic to $\Lambda^+ \star \bigstar_{i \in I} i^+ = \Lambda^+ \star I^+ = (\Lambda \cup I)^+$. ◀

[2] Post's statement of the problem [35] is equivalent to ours. In particular, we may assume $\varphi(i), \psi(i) \neq \varepsilon$.

[3] It is worth mentioning that we may assume I to only contain five elements [32] and Λ to be a binary alphabet (using standard encoding techniques). Note that we may only allow non-empty entries, however.

The states in $\hat{R}$ of $\hat{\mathcal{R}}$ do not form a basis of the free semigroup. To simplify working with this fact, we make the following definition(s).

▶ **Definition 3.2** (natural projection). *There is a natural projection $\pi : \hat{\Lambda}^* \to \Lambda^*$ where $\hat{\Lambda} = \bigcup_{\ell=1}^{L} \Lambda^\ell$, which interprets a letter $\hat{\lambda} \in \hat{\Lambda}$ as the corresponding word over Λ. We extend this projection into a homomorphism $\pi : \hat{R}^* \to R^*$ by setting $\pi(i) = i$ for all $i \in I$. Two elements $\hat{\boldsymbol{r}}_1, \hat{\boldsymbol{r}}_2 \in \hat{R}^*$ are* R-equivalent *(written as $\hat{\boldsymbol{r}}_1 =_R \hat{\boldsymbol{r}}_2$) if $\pi(\hat{\boldsymbol{r}}_1) = \pi(\hat{\boldsymbol{r}}_2)$. Finally, $|\hat{\boldsymbol{r}}|_R$ for $\boldsymbol{r} \in \hat{R}^*$ is $|\hat{\boldsymbol{r}}|_R = |\pi(\hat{\boldsymbol{r}})|$.*

Note that we have $\hat{\boldsymbol{r}}_1 =_R \hat{\boldsymbol{r}}_2$ if and only if $\hat{\boldsymbol{r}}_1 =_{\hat{\mathcal{R}}} \hat{\boldsymbol{r}}_2$ for all $\hat{\boldsymbol{r}}_1, \hat{\boldsymbol{r}}_2 \in \hat{R}^*$ as $\mathscr{S}(\hat{\mathcal{R}}) \simeq R^+$.

Definition of $\mathcal{S}$. We use the automaton $\hat{\mathcal{R}} = (\hat{R}, \Gamma, \rho)$ as a building block for our target automaton $\mathcal{T} = (Q, \Sigma, \delta)$ for the reduction. We fix some arbitrary element $\lambda_\# \in \Lambda \subseteq \hat{R}$. To compute $\mathcal{S}$ from $\hat{\mathcal{R}}$, we duplicate the state $\lambda_\#$ twice and call these copies $\#_1$ and $\#_2$. Formally, we have $\mathcal{S} = (Q, \Gamma, \sigma)$ where $Q = \hat{R} \uplus \{\#_1, \#_2\}$ for the new symbols $\#_1$ and $\#_2$ and $\sigma = \rho \cup \{\#_1 \xrightarrow{c/d} q, \#_2 \xrightarrow{c/d} q \mid \lambda_\# \xrightarrow{c/d} q \in \rho\}$. Thus, the new states $\#_1$ and $\#_2$ act in the same way as $\lambda_\#$ and we have $\mathscr{S}(\mathcal{S}) = \mathscr{S}(\hat{\mathcal{R}}) \simeq R^+$.

Definition of $\mathcal{T}$. The next step is to fix another $\lambda_R \in \Lambda \subseteq Q$ arbitrarily but different to $\lambda_\#$ and take $\mathcal{T}_1 = (Q, \Gamma \cup \{a, b\}, \delta_1) = \mathcal{S} \cup \mathcal{T}_1'$ where $\mathcal{T}_1'$ is given via its dual in Figure 4 (i. e. we add two new letters a, b to the alphabet and some additional transitions). Note that we have the transitions $\lambda_\# \xrightarrow{a/a} \lambda_R$ and the self-loops $\lambda_R^\ell \xrightarrow{a/a} \lambda_R^\ell$ for all $1 \leq \ell \leq L$ in $\mathcal{T}_1$.

The idea for this part is that we may factorize a state sequence $\boldsymbol{q} \in Q^*$ into blocks from $\hat{R}^*$ and symbols $\#_1$ and $\#_2$ and then remove the blocks one after another using the letter a. We will explain this precisely later in Fact 3.3.

Finally, we let $\mathcal{T} = (Q, \Sigma, \delta) = \mathcal{T}_1 \cup \mathcal{T}_2$ where $\mathcal{T}_2$ is given via its dual in Figure 5. Note, in particular, that we have $\varphi(i), \psi(i) \in \bigcup_{\ell=1}^{L} \Lambda^\ell = \hat{\Lambda} \subseteq \hat{R}$.

In other words, we obtain $\mathcal{T}$ from $\mathcal{T}_1$ by adding new symbols to the alphabet resulting in $\Sigma = \Gamma \cup \{a, b\} \cup \{\iota, \alpha, \alpha', f_\alpha, \beta, \beta', f_\beta, f\}$ and adding the transitions depicted in Figure 5 for all $i \in I$ and $\hat{\lambda} \in \hat{\Lambda}$. Clearly, $\mathcal{T}$ can be computed and is a complete $\mathscr{S}$-automaton.

The Role of a and b in $\mathcal{T}$. As already mentioned above, we may use the letter a to remove a block from a certain factorization of a state sequence (the proof is by induction on μ):

▶ **Fact 3.3.** *Let $\boldsymbol{p} \in Q^*$ and factorize it as $\boldsymbol{p} = (\boldsymbol{p}_s \#_{x_s}) \dots (\boldsymbol{p}_1 \#_{x_1}) \boldsymbol{p}_0$ for $\boldsymbol{p}_0, \dots, \boldsymbol{p}_s \in \hat{R}^*$ and $x_1, \dots, x_s \in \{1, 2\}$. Then, for any $1 \leq \mu \leq s$, we have (in $\mathcal{T}$):*

$$\boldsymbol{p} \cdot a^\mu = (\boldsymbol{p}_s \#_{x_s}) \dots (\boldsymbol{p}_{\mu+1} \#_{x_{\mu+1}}) \boldsymbol{p}_\mu \, \lambda_\# \lambda_R^{\mu-1+|\boldsymbol{p}_{\mu-1} \dots \boldsymbol{p}_0|_R}$$

Correctness. It remains to show that the `PCP` instance φ, ψ, I has a solution if and only if $\mathscr{S}(\mathcal{T})$ is **not** a free semigroup. We start with the (easier) "only if" direction and show that the additional transitions from $\mathcal{T}_1$ and $\mathcal{T}_2$ do not affect the subautomaton $\hat{\mathcal{R}}$: if two state sequences are R-equivalent, they are also equal with respect to $\mathcal{T}$.

▶ **Lemma 3.4.** *Let $\hat{\boldsymbol{r}}_1, \hat{\boldsymbol{r}}_2 \in \hat{R}^*$ with $\hat{\boldsymbol{r}}_1 =_R \hat{\boldsymbol{r}}_2$. Then, we have $\hat{\boldsymbol{r}}_1 =_{\mathcal{T}} \hat{\boldsymbol{r}}_2$.*

Proof Sketch. We need to show $\hat{\boldsymbol{r}}_1 \circ u = \hat{\boldsymbol{r}}_2 \circ u$ for all $u \in \Sigma^*$ and this can be done by induction on u. Thus, write $u = cu'$ for some $c \in \Sigma = \Gamma \cup \{a, b\} \cup \{\iota, \alpha, \alpha', f_\alpha, \beta, \beta', f_\beta, f\}$ and $u' \in \Sigma^*$.

Most cases for c are straight-forward (for example, for $c \in \Gamma$ – the alphabet of $\hat{\mathcal{R}}$ – we inherit this property from $\hat{\mathcal{R}}$) and we only demonstrate the case $c \in \{\alpha, \alpha', \beta, \beta'\}$. Here, we factorize $\hat{\boldsymbol{r}}_1 = \hat{\boldsymbol{s}}_1 \hat{\lambda}_1 \boldsymbol{i}_1$ with $\boldsymbol{i}_1 \in I^*$ maximal, $\hat{\lambda}_1 \in \hat{\Lambda} \cup \{\varepsilon\}$ and $\hat{\boldsymbol{s}}_1 \in \hat{R}^*$ with

Table 1 Various symbols in the order of their definition.

symbol	usage
Λ	: PCP base alphabet, $\lvert\Lambda\rvert \geq 2$
I	: PCP index set, $\lvert I\rvert \geq 1$, $I \cap \Lambda = \emptyset$
φ, ψ	: $I \to \Lambda^+$ PCP homomorphisms
L	$= \max\{\lvert\varphi(i)\rvert, \lvert\psi(i)\rvert \mid i \in I\}$
$\hat{\Lambda}$	$= \bigcup_{\ell=1}^{L} \Lambda^{\ell}$
R	$= \Lambda \cup I$
$\hat{R}$	$= \hat{\Lambda} \cup I$: state set of $\mathcal{R}$
$\hat{\mathcal{R}}$	$= (\hat{R}, \Gamma, \rho)$: complete $\mathscr{S}$-automaton generating $R^+ = (\Lambda \cup I)^+$
ρ	: transition set of $\hat{\mathcal{R}}$
Γ	: alphabet of $\hat{\mathcal{R}}$ and $\mathcal{S}$
π	: $\hat{\Lambda}^* \to \Lambda$, $\hat{R}^* \to R^*$ natural projection with $\pi(i) = i$ for all $i \in I$
$\lvert\hat{\boldsymbol{r}}\rvert_R$	: length of $\pi(\hat{\boldsymbol{r}})$ for $\hat{\boldsymbol{r}} \in \hat{R}^*$
$\lambda_\# \in \Lambda \subseteq \hat{R}$	: arbitrarily chosen element
$\#_1, \#_2$	: copies of $\lambda_\#$
$\mathcal{S}$	$= (Q, \Gamma, \sigma)$: complete $\mathscr{S}$-automaton, extension of $\hat{\mathcal{R}}$ still generating R^+
Q	$= \hat{R} \uplus \{\#_1, \#_2\}$: state set of $\mathcal{S}$ and $\mathcal{T}$
σ	: transition set of $\mathcal{S}$
$\lambda_R \in \Lambda \subseteq Q$	: arbitrarily chosen element with $\lambda_R \neq \lambda_\#$
$a, b \notin \Gamma$	: new letters for $\mathcal{T}_1$
$\mathcal{T}_1'$	$= (Q, \{a, b\}, \delta_1')$: complete $\mathscr{S}$-automaton, additional transitions for $\mathcal{T}_1$, see Figure 4
$\mathcal{T}_1$	$= (Q, \Gamma \uplus \{a, b\}, \delta_1) = \mathcal{S} \cup \mathcal{T}_1'$: complete $\mathscr{S}$-automaton, extension of $\mathcal{S}$ by $\mathcal{T}_1'$
δ_1	: transition set of $\mathcal{T}_1$
$\mathcal{T}$	$= (Q, \Sigma, \delta) = \mathcal{T}_1 \cup \mathcal{T}_2$: complete $\mathscr{S}$-automaton with $e =_{\mathcal{T}} \varepsilon$, result of the reduction
$\mathcal{T}_2$	: complete $\mathscr{S}$-automaton with new transitions for $\mathcal{T}$, see Figure 5
Σ	$= \Gamma \cup \{a, b\} \cup \{\iota, \alpha, \alpha', f_\alpha, \beta, \beta', f_\beta, f\}$: alphabet of $\mathcal{T}$
$\pi_\#$	: $Q^* \to \{\#_1, \#_2\}^*$ homomorphism with $\pi_\#(\#_x) = \#_x$ but $\pi_\#(\hat{r}) = \varepsilon$ for $\hat{r} \in \hat{R}$
π'	: $Q^* \to (R \cup \{\#_1, \#_2\})^*$ homomorphism extending π with $\pi'(\#_x) = \#_x$ for $x \in \{1, 2\}$

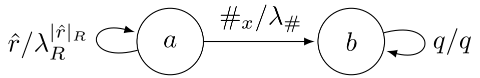


Figure 4 The dual $\partial\mathcal{T}_1'$. The transitions exist for all $\hat{r} \in \hat{R}$, $x \in \{1, 2\}$ and $q \in Q$.

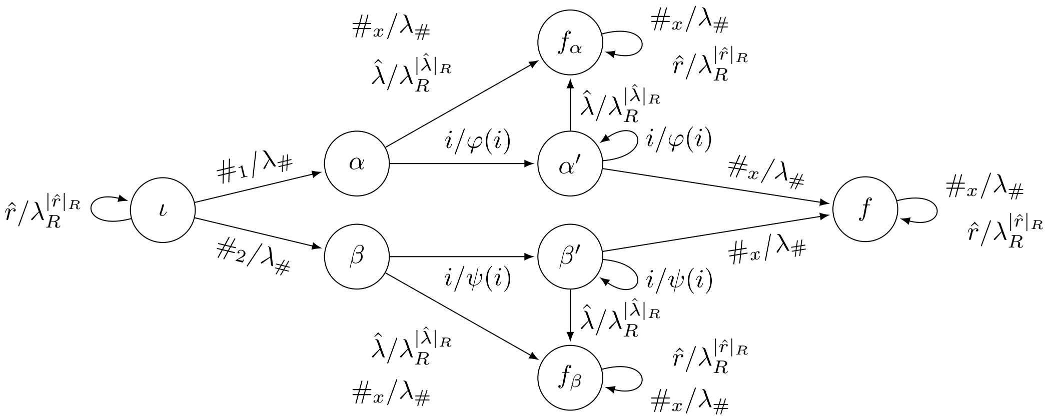

$\#_x/\lambda_\#$
$\hat{\lambda}/\lambda_R^{|\hat{\lambda}|_R}$
f_α
$\#_x/\lambda_\#$
$\hat{r}/\lambda_R^{|\hat{r}|_R}$
$\hat{\lambda}/\lambda_R^{|\hat{\lambda}|_R}$
$i/\varphi(i)$
$i/\varphi(i)$
$\#_1/\lambda_\#$
α
α'
$\#_x/\lambda_\#$
$\hat{r}/\lambda_R^{|\hat{r}|_R}$
ι
f
$\#_x/\lambda_\#$
$\hat{r}/\lambda_R^{|\hat{r}|_R}$
$\#_2/\lambda_\#$
β
β'
$\#_x/\lambda_\#$
$i/\psi(i)$
$i/\psi(i)$
$\hat{\lambda}/\lambda_R^{|\hat{\lambda}|_R}$
$\hat{\lambda}/\lambda_R^{|\hat{\lambda}|_R}$
$\#_x/\lambda_\#$
f_β
$\hat{r}/\lambda_R^{|\hat{r}|_R}$
$\#_x/\lambda_\#$

Figure 5 The dual $\partial\mathcal{T}_2$. The transitions exist for all $i \in I, \hat{r} \in \hat{R}, \hat{\lambda} \in \hat{\Lambda}$ and $x \in \{1, 2\}$.

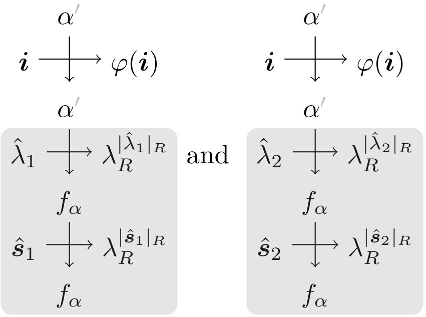

α'
$\boldsymbol{i}$
$\varphi(\boldsymbol{i})$
$\hat{\lambda}_1$
$\lambda_R^{|\hat{\lambda}_1|_R}$
f_α
$\hat{\boldsymbol{s}}_1$
$\lambda_R^{|\hat{\boldsymbol{s}}_1|_R}$
and
$\hat{\lambda}_2$
$\lambda_R^{|\hat{\lambda}_2|_R}$
$\hat{\boldsymbol{s}}_2$
$\lambda_R^{|\hat{\boldsymbol{s}}_2|_R}$

Figure 6 Cross diagrams for Lemma 3.4.

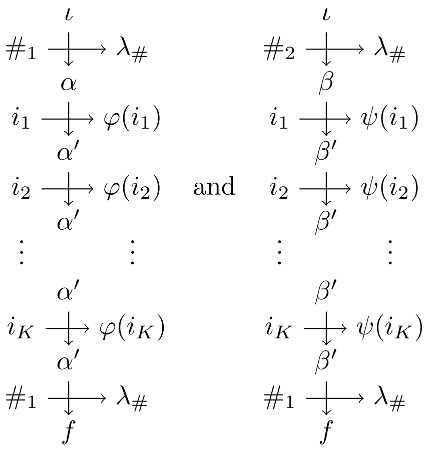


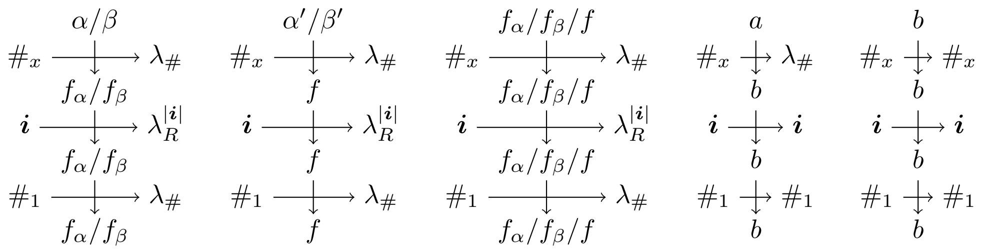
Figure 7 Cross diagrams for Lemma 3.5.

$\lambda_1 = \varepsilon \implies \hat{\boldsymbol{s}}_1 = \varepsilon$. Analogously, we factorize $\hat{\boldsymbol{r}}_2 = \hat{\boldsymbol{s}}_2\hat{\lambda}_2\boldsymbol{i}_2$. Observe that, since we have $\hat{\boldsymbol{r}}_1 =_R \hat{\boldsymbol{r}}_2$, we must have $\boldsymbol{i}_1 = \boldsymbol{i}_2 = \boldsymbol{i}$, $\hat{\boldsymbol{s}}_1\hat{\lambda}_1 =_R \hat{\boldsymbol{s}}_2\hat{\lambda}_2$ and $\hat{\lambda}_1 = \varepsilon \iff \hat{\lambda}_2 = \varepsilon$. This yields the cross diagrams in Figure 6 where the shaded parts only exist if $\hat{\lambda}_1, \hat{\lambda}_2 \neq \varepsilon$ and where we have α' after applying $\boldsymbol{i}$ if $\boldsymbol{i} \neq \varepsilon$. In both diagrams, we have the same state sequence on the right hand side (because of $\hat{\boldsymbol{s}}_1\hat{\lambda}_1 =_R \hat{\boldsymbol{s}}_2\hat{\lambda}_2$) and, thus, are done. The case $c \in \{\beta, \beta'\}$ is analogous (using ψ). ◀

Finally, we show that a solution for the PCP instance implies a proper relation in the semigroup generated by $\mathcal{T}$ and, thus, that it is not free.

▶ **Lemma 3.5.** *If $\boldsymbol{i} \in I^+$ is a solution for the* PCP *instance, then we have $\#_1\boldsymbol{i}\#_1 =_\mathcal{T} \#_1\boldsymbol{i}\#_2$.*

Proof Sketch. We show $\#_1\boldsymbol{i}\#_1 \circ u = \#_1\boldsymbol{i}\#_2 \circ u$ for all $u \in \Sigma^*$. For $u = \varepsilon$, there is nothing to show. So, let $u = cu'$ for some $c \in \Sigma = \Gamma \cup \{a, b\} \cup \{\iota, \alpha, \alpha', f_\alpha, \beta, \beta', f_\beta, f\}$ and $u' \in \Sigma^*$. Again, we only fully demonstrate the most interesting case $c = \iota$ (the other cases may be found in Figure 8 where Figure 8e requires induction). Writing $\boldsymbol{i} = i_K \dots i_2 i_1$ for $i_1, \dots, i_K \in I$, we obtain the diagrams in Figure 7. Since $\boldsymbol{i} = i_K \dots i_2 i_1$ is a solution, we have $\varphi(i_K) \dots \varphi(i_2)\varphi(i_1) =_R \psi(i_K) \dots \psi(i_2)\psi(i_1)$. Thus, Lemma 3.4 implies $\lambda_\#\varphi(i_K) \dots \varphi(i_2)\varphi(i_1)\lambda_\# =_\mathcal{T} \lambda_\#\psi(i_K) \dots \psi(i_2)\psi(i_1)\lambda_\#$. ◀



(a) $c \in \{\alpha, \beta\}$ **(b)** $c \in \{\alpha', \beta'\}$ **(c)** $c \in \{f_\alpha, f_\beta, f\}$ **(d)** $c = a$ **(e)** $c = b$

Figure 8 Various cases for $c \in \Sigma$. The cross diagrams hold for $x \in \{1, 2\}$.

▶ **Proposition 3.6.** *If the* PCP *instance has a solution, $\mathscr{S}(\mathcal{T})$ is not (left) cancellative and, thus, not a free semigroup.*

Converse Direction. To show that the PCP instance has a solution if the semigroup is not free, we introduce the notion of compatibility and observe that every relation is compatible. The proof relies on Fact 3.3 for removing blocks from the factorization used in Definition 3.7 and that $\mathcal{S}$ (generating R^+) survives as a subautomaton of $\mathcal{T}$. The latter allows us to use the cancellativity of R^+ to show that the individual blocks are the same in R^+.

▶ **Definition 3.7** (compatible state sequences). *Factorize $\boldsymbol{p}, \boldsymbol{q} \in Q^*$ (uniquely) as $\boldsymbol{p} = (\boldsymbol{p}_s\#_{x_s}) \dots (\boldsymbol{p}_1\#_{x_1})\, \boldsymbol{p}_0$ and $\boldsymbol{q} = (\boldsymbol{q}_t\#_{y_t}) \dots (\boldsymbol{q}_1\#_{y_1})\, \boldsymbol{q}_0$ with $\boldsymbol{p}_0, \dots, \boldsymbol{p}_s, \boldsymbol{q}_0, \dots, \boldsymbol{q}_t \in \hat{R}^*$ and $x_1, \dots, x_s, y_1, \dots, y_t \in \{1, 2\}$. They are* compatible *if $s = t$ and $\forall\, 0 \leq i \leq s = t : \boldsymbol{p}_i =_R \boldsymbol{q}_i$.*

▶ **Lemma 3.8.** *Let $\boldsymbol{p}, \boldsymbol{q} \in Q^*$ with $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$. Then, we have that $\boldsymbol{p}$ and $\boldsymbol{q}$ are compatible.*

Proof. We factorize $\boldsymbol{p}$ and $\boldsymbol{q}$ in the same way as in Definition 3.7 and show the statement by induction on $s + t$. For $s = t = 0$, we have $\boldsymbol{p}_0 = \boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q} = \boldsymbol{q}_0$. Since $\hat{\mathcal{R}}$ is a subautomaton of $\mathcal{T}$, this implies $\boldsymbol{p}_0 =_{\hat{\mathcal{R}}} \boldsymbol{q}_0$ and, equivalently, $\boldsymbol{p} = \boldsymbol{p}_0 =_R \boldsymbol{q}_0 = \boldsymbol{q}$.

For the inductive step ($s + t > 0$), we may assume $s > 0$ (due to symmetry) or, in other words, that $\boldsymbol{p}$ contains at least one $\#_1$ or $\#_2$. We have $\boldsymbol{p} \circ a = b$ (compare to Figure 4) and, thus, due to $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$, also $\boldsymbol{q} \circ a = \boldsymbol{p} \circ a = b$. This is only possible (again, compare to Figure 4) if $\boldsymbol{q}$ also contains at least one $\#_1$ or $\#_2$, i. e. if $t > 0$.

From Fact 3.3 (with $\mu = 1$), we obtain (for both $\boldsymbol{p}$ and $\boldsymbol{q}$):

$$\begin{aligned} \boldsymbol{p} \cdot a &= \boldsymbol{p}' \lambda_\# \lambda_R^{|\boldsymbol{p}_0|_R} \\ &\qquad \text{for } \boldsymbol{p}' = (\boldsymbol{p}_s\#_{x_s}) \dots (\boldsymbol{p}_2\#_{x_2})\, \boldsymbol{p}_1 \text{ and} \\ \boldsymbol{q} \cdot a &= \boldsymbol{q}' \lambda_\# \lambda_R^{|\boldsymbol{q}_0|_R} \\ &\qquad \text{for } \boldsymbol{q}' = (\boldsymbol{q}_t\#_{x_t}) \dots (\boldsymbol{q}_2\#_{x_2})\, \boldsymbol{q}_1 \end{aligned}$$

Now, $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$ implies $\boldsymbol{p}' \lambda_\# \lambda_R^{|\boldsymbol{p}_0|_R} = \boldsymbol{p} \cdot a =_{\mathcal{T}} \boldsymbol{q} \cdot a = \boldsymbol{q}' \lambda_\# \lambda_R^{|\boldsymbol{q}_0|_R}$ and we may apply the induction hypothesis, which yields that $\boldsymbol{p}' \lambda_\# \lambda_R^{|\boldsymbol{p}_0|_R}$ and $\boldsymbol{q}' \lambda_\# \lambda_R^{|\boldsymbol{q}_0|_R}$ are compatible. This means that we have $s = t$, $\boldsymbol{p}_\mu =_R \boldsymbol{q}_\mu$ for all $2 \leq \mu \leq s = t$ and $\boldsymbol{p}_1 \lambda_\# \lambda_R^{|\boldsymbol{p}_0|_R} =_R \boldsymbol{q}_1 \lambda_\# \lambda_R^{|\boldsymbol{q}_0|_R}$. Observe that the latter implies $\boldsymbol{p}_1 =_R \boldsymbol{q}_1$ (as we have chosen $\lambda_\#$ and λ_R as different elements of Λ). In particular, we also obtain $\boldsymbol{p}_s \lambda_\# \boldsymbol{p}_{s-1} \dots \lambda_\# \boldsymbol{p}_1 =_R \boldsymbol{q}_t \lambda_\# \boldsymbol{q}_{t-1} \dots \lambda_\# \boldsymbol{q}_1$.

Since $\mathcal{S}$ is a subautomaton of $\mathcal{T}$, $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$ implies $\boldsymbol{p} =_{\mathcal{S}} \boldsymbol{q}$. As $\#_1$ and $\#_2$ act in the same way as $\lambda_\#$ in $\mathcal{S}$ by construction, this shows $\boldsymbol{p}_s \lambda_\# \dots \boldsymbol{p}_1 \lambda_\# \boldsymbol{p}_0 =_{\mathcal{S}} \boldsymbol{q}_t \lambda_\# \dots \boldsymbol{q}_1 \lambda_\# \boldsymbol{q}_0$ and, because of $\mathscr{S}(\mathcal{S}) \simeq R^+$, also $\boldsymbol{p}_s \lambda_\# \dots \boldsymbol{p}_1 \lambda_\# \boldsymbol{p}_0 =_R \boldsymbol{q}_t \lambda_\# \dots \boldsymbol{q}_1 \lambda_\# \boldsymbol{q}_0$. Now, because R^* as a free monoid is cancellative (see Fact 2.1) and because we have $\boldsymbol{p}_s \lambda_\# \boldsymbol{p}_{s-1} \dots \lambda_\# \boldsymbol{p}_1 =_R \boldsymbol{q}_t \lambda_\# \boldsymbol{q}_{t-1} \dots \lambda_\# \boldsymbol{q}_1$ (from above), we obtain $\lambda_\# \boldsymbol{p}_0 =_R \lambda_\# \boldsymbol{q}_0$ and, finally, $\boldsymbol{p}_0 =_R \boldsymbol{q}_0$, which concludes the proof that $\boldsymbol{p}$ and $\boldsymbol{q}$ are compatible. ◀

On the other hand, not every compatible pair forms a semigroup relation. However, this is true by Lemma 3.4 if, additionally, the subsequence containing only $\#_1$ and $\#_2$ is the same in both entries. To formalize this, we introduce the following definition.

▶ **Definition 3.9** (projection on $\{\#_1, \#_2\}$). *Let $\pi_\# : Q^* \to \{\#_1, \#_2\}^*$ be the homomorphism given by $\pi_\#(\#_x) = \#_x$ for both $x \in \{1, 2\}$ and $\pi_\#(\hat{r}) = \varepsilon$ for all other $\hat{r} \in Q \setminus \{\#_1, \#_2\} = \hat{R}$.*

▶ **Lemma 3.10.** *Let $\boldsymbol{p}, \boldsymbol{q} \in Q^*$ be compatible with $\pi_\#(\boldsymbol{p}) = \pi_\#(\boldsymbol{q})$. Then, we have $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$.*

Combining the last two lemmas, we obtain that $\mathscr{S}(\mathcal{T})$ is a free semigroup if all its relations have the same projection under $\pi_\#$. Most importantly, we will later on apply the contraposition of the “only if” direction of the following lemma to obtain a relation with different images under the projection if the semigroup is not free.

▶ **Lemma 3.11.** *Let $\pi' : Q^* \to (R \cup \{\#_1, \#_2\})^*$ be the extension of the natural projection π (from Definition 3.2) with $\pi'(\#_x) = \#_x$ for $x \in \{1, 2\}$. The following are equivalent:*

1. *For all $\boldsymbol{p}, \boldsymbol{q} \in Q^+$ with $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$, we have $\pi_\#(\boldsymbol{p}) = \pi_\#(\boldsymbol{q})$.*
2. *The map π' induces a well-defined homomorphism $\mathscr{S}(\mathcal{T}) \to (R \cup \{\#_1, \#_2\})^+$.*
3. *The map π' induces a well-defined isomorphism $\mathscr{S}(\mathcal{T}) \to (R \cup \{\#_1, \#_2\})^+$.*

In particular, $\mathscr{S}(\mathcal{T})$ is isomorphic to $(R \cup \{\#_1, \#_2\})^+$ if we have $\pi_\#(\boldsymbol{p}) = \pi_\#(\boldsymbol{q})$ for all $\boldsymbol{p}, \boldsymbol{q} \in Q^+$ with $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$.

Now a relation whose sides have different images under $\pi_\#$ yields a `PCP` solution.

▶ **Lemma 3.12.** *If there are $\boldsymbol{p}, \boldsymbol{q} \in Q^+$ with $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$ but $\pi_\#(\boldsymbol{p}) \neq \pi_\#(\boldsymbol{q})$, then the* `PCP` *instance has a solution.*

Proof. We factorize these $\boldsymbol{p}$ and $\boldsymbol{q}$ in the same way as in Definition 3.7 and observe that $\boldsymbol{p}$ and $\boldsymbol{q}$ are compatible by Lemma 3.8. We may assume that there is some $1 \leq \mu_0 \leq s = t$ with $\#_{x_{\mu_0}} = \#_1$ but $\#_{y_{\mu_0}} = \#_2$ (due to symmetry).

We may assume $\mu_0 = 1$ without loss of generality. This is because we may substitute $\boldsymbol{p}$ by $\boldsymbol{p}' = \boldsymbol{p} \cdot a^{\mu_0 - 1}$ and $\boldsymbol{q}' = \boldsymbol{q} \cdot a^{\mu_0 - 1}$ (we still have $\boldsymbol{p}' =_{\mathcal{T}} \boldsymbol{q}'$) by Fact 3.3 (for $\mu_0 > 1$).

With these assumptions, we apply $\boldsymbol{p}$ and $\boldsymbol{q}$ to ι and obtain (see Figure 5) the cross diagrams depicted in Figure 9 for $\tilde{\boldsymbol{p}} = \boldsymbol{p}_s \#_{x_s} \dots \boldsymbol{p}_3 \#_{x_3} \boldsymbol{p}_2$, $\tilde{\boldsymbol{q}} = \boldsymbol{q}_t \#_{y_t} \dots \boldsymbol{q}_3 \#_{y_3} \boldsymbol{q}_2$ and some $\boldsymbol{p}_1', \tilde{\boldsymbol{p}}', \boldsymbol{q}_1', \tilde{\boldsymbol{q}}' \in Q^*$, $p_2', q_2' \in Q$ and $c_1, c_2, c, d_1, d_2, d \in \Gamma$. Since we have $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$, we must have $c = d$ and, by the construction of $\mathcal{T}$, this is only possible if $c = f = d$ (see Figure 5). This, in turn, is only possible if we have $\boldsymbol{p}_1 = \boldsymbol{i} \in I^+$ and $\boldsymbol{q}_1 = \boldsymbol{j} \in I^+$. Since $\boldsymbol{p}$ and $\boldsymbol{q}$ are compatible, we must even have $\boldsymbol{i} = \boldsymbol{p}_1 =_R \boldsymbol{q}_1 = \boldsymbol{j}$, which implies $\boldsymbol{i} = \boldsymbol{j}$. Additionally, we also obtain $\boldsymbol{p}_1' =_R \varphi(\boldsymbol{i})$, $c_1 = \alpha'$, $p_2' = \lambda_\#$, $c_2 = f$, $\boldsymbol{q}_1' =_R \psi(\boldsymbol{i})$, $d_1 = \beta'$, $q_2' = \lambda_\#$, $d_2 = f$ and $\tilde{\boldsymbol{p}}' = \lambda_R^{|\boldsymbol{p}_s|_R} \lambda_\# \dots \lambda_R^{|\boldsymbol{p}_3|_R} \lambda_\# \lambda_R^{|\boldsymbol{p}_2|_R}$ as well as $\tilde{\boldsymbol{q}}' = \lambda_R^{|\boldsymbol{q}_t|_R} \lambda_\# \dots \lambda_R^{|\boldsymbol{q}_3|_R} \lambda_\# \lambda_R^{|\boldsymbol{q}_2|_R}$ from the construction of $\mathcal{T}$. This shows

$$\begin{aligned} &\lambda_R^{|\boldsymbol{p}_s|_R} \lambda_\# \dots \lambda_R^{|\boldsymbol{p}_3|_R} \lambda_\# \lambda_R^{|\boldsymbol{p}_2|_R} \, \lambda_\# \varphi(\boldsymbol{i}) \lambda_\# \, \lambda_R^{|\boldsymbol{p}_0|_R} \\ =_{\mathcal{T}} &\lambda_R^{|\boldsymbol{q}_t|_R} \lambda_\# \dots \lambda_R^{|\boldsymbol{q}_3|_R} \lambda_\# \lambda_R^{|\boldsymbol{q}_2|_R} \, \lambda_\# \psi(\boldsymbol{i}) \lambda_\# \, \lambda_R^{|\boldsymbol{q}_0|_R} \end{aligned}$$

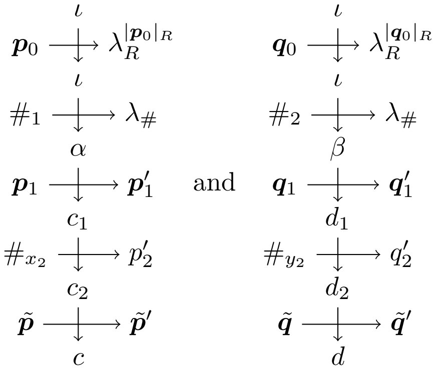


Figure 9 Cross diagrams for Lemma 3.12.

and, by Lemma 3.8, also that both sides are R-equivalent. Since $\boldsymbol{p}$ and $\boldsymbol{q}$ are compatible, we have $\lambda_R^{|\boldsymbol{p}_\mu|_R} =_R \lambda_R^{|\boldsymbol{q}_\mu|_R}$ for all $0 \leq \mu \leq s = t$. Combining this with the cancellativity of R^*, we obtain $\varphi(\boldsymbol{i}) =_R \psi(\boldsymbol{i})$ and, thus, that $\boldsymbol{i}$ is a solution for the PCP instance. ◀

We have now shown that the PCP instance has a solution if the semigroup generated by $\mathcal{T}$ is not free. A careful analysis of the proof yields more, however, which we collect in Proposition 3.14 (which follows from the lemmas and propositions above). For one part of this statement, we will first state another consequence of Lemma 3.8:

▶ **Proposition 3.13.** *Mapping $\hat{r}$ to $|\hat{r}|_R$ for every $\hat{r} \in \hat{R}$ and $\#_x$ to 1 for $x \in \{1, 2\}$ induces a well-defined proper length function of $\mathscr{M}(\mathcal{T})$ (and a well-defined length function of $\mathscr{S}(\mathcal{T})$).*

▶ **Proposition 3.14.** *The following statements are equivalent:*

1. *The* PCP *instance has a solution $\boldsymbol{i} \in I^+$.*
2. *We have $\#_1\boldsymbol{i}\#_1 =_{\mathcal{T}} \#_1\boldsymbol{i}\#_2$ for some $\boldsymbol{i} \in I^+$.*
3. *There are $\boldsymbol{p}, \boldsymbol{q} \in Q^+$ with $\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q}$ but $\pi_\#(\boldsymbol{p}) \neq \pi_\#(\boldsymbol{q})$.*
4. *$\mathscr{S}(\mathcal{T})$ is not a free semigroup.*
5. *$\mathscr{S}(\mathcal{T})$ is not isomorphic to $(R \cup \{\#_1, \#_2\})^+$.*
6. *$\mathscr{S}(\mathcal{T})$ is not (left[4]) cancellative.*
7. *$\mathscr{S}(\mathcal{T})$ is not equidivisible.*

4'. *$\mathscr{M}(\mathcal{T})$ is not a free monoid.*
5'. *$\mathscr{M}(\mathcal{T})$ is not isomorphic to $(R \cup \{\#_1, \#_2\})^*$.*
6'. *$\mathscr{M}(\mathcal{T})$ is not (left) cancellative.*
7'. *$\mathscr{M}(\mathcal{T})$ is not equidivisible.*

Main Theorem and other Consequences. Proposition 3.14 shows that we have reduced PCP to (the complements of) the freeness problem for (complete) automaton semigroups and monoids (as the construction of $\mathcal{T}$ is computable). Since PCP is undecidable [35], we obtain:

▶ **Theorem 3.15.** *The freeness problem for automaton semigroups*

Input: *a (complete) $\mathscr{S}$-automaton $\mathcal{T}$*
Question: *is $\mathscr{S}(\mathcal{T})$ a free semigroup?*

and the freeness problem for automaton monoids

Input: *a (complete) $\mathscr{S}$-automaton $\mathcal{T}$*
Question: *is $\mathscr{M}(\mathcal{T})$ a free monoid?*

are undecidable.

▶ **Theorem 3.16.** *The following problems are undecidable:*

Input: *a complete $\mathscr{S}$-automaton $\mathcal{T}$*
Question: *is $\mathscr{S}(\mathcal{T})$ (left) cancellative/equidivisible?*

Input: *a complete $\mathscr{S}$-automaton $\mathcal{T}$*
Question: *is $\mathscr{M}(\mathcal{T})$ (left) cancellative/equidivisible?*

Finally, we obtain that it is undecidable whether a given map on the generators induces a homomorphism (or an isomorphism) between two automaton semigroups (using $\mathcal{T}$ from above as $\mathcal{T}_1$ and an automaton generating $(R \cup \{\#_1, \#_2\})^+$ for $\mathcal{T}_2$). Note that the isomorphism problem for automaton groups (and, thus, also for automaton semigroups and monoids) is known to be undecidable (as it follows from [40]).

[4] Recall that we defined automaton semigroups by a left action here.

▶ **Theorem 3.17.** *The following two problems are undecidable:*

Input: *two (complete)* $\mathscr{S}$*-automata* $\mathcal{T}_1 = (Q_1, \Sigma_1, \delta_1)$ *and* $\mathcal{T}_2 = (Q_2, \Sigma_2, \delta_2)$ *and a map* $f : Q_1 \to Q_2$
Question: *does* f *extend into a homomorphism* $\mathscr{S}(\mathcal{T}_1) \to \mathscr{S}(\mathcal{T}_2)$*?*

Input: *two (complete)* $\mathscr{S}$*-automata* $\mathcal{T}_1 = (Q_1, \Sigma_1, \delta_1)$ *and* $\mathcal{T}_2 = (Q_2, \Sigma_2, \delta_2)$ *and a map* $f : Q_1 \to Q_2$
Question: *does* f *extend into an isomorphism* $\mathscr{S}(\mathcal{T}_1) \to \mathscr{S}(\mathcal{T}_2)$*?*

For our construction, we need that all $\varphi(i)$ and $\psi(i)$ are states in the automaton. This immediately yields relations of the form $u\,v =_{\hat{\mathcal{R}}} uv$ for $u, v, uv \in \hat{\Lambda}$ that still exist in the eventual automaton $\mathcal{T}$. In the monoid case, however, we may use the neutral element as a "padding symbol" and thus avoid using a power automaton. This then yields:

▶ **Theorem 3.18.** *The free presentation problem for automaton monoids is undecidable:*

Input: *a (complete)* $\mathscr{S}$*-automaton* $\mathcal{T} = (Q, \Sigma, \delta)$ *with a dedicated state* $e \in Q$ *acting as the identity map*
Question: *is* $\mathscr{M}(\mathcal{T}) \simeq (Q \setminus \{e\})^*$*?*

In the semigroup case, we only get a weaker form of this result (using $P = R \cup \{\#_1, \#_2\}$):

▶ **Theorem 3.19.** *The following problem is undecidable:*

Input: *a (complete)* $\mathscr{S}$*-automaton* $\mathcal{T} = (Q, \Sigma, \delta)$ *and a subset* $P \subseteq Q$
Question: *is* $\mathscr{S}(\mathcal{T}) \simeq P^+$*?*

4 Open Problems

Theorem 3.18 immediately raises the question whether the corresponding problem for automaton semigroups is also undecidable:

▶ **Open Problem 4.1.** *Is the following problem decidable?*

Input: *a (complete)* $\mathscr{S}$*-automaton* $\mathcal{T} = (Q, \Sigma, \delta)$
Question: *is* $\mathscr{S}(\mathcal{T}) \simeq Q^+$*?*

In Theorem 3.16, we have also shown that it is not possible to test whether a given automaton semigroup (or monoid) is equidivisible. By Levi's lemma (Fact 2.2) this is one part of a semigroup (monoid) being free while the other one is the existence of a (proper) length function. So, the following question naturally arises.

▶ **Open Problem 4.2.** *Is the following problem decidable?*

Input: *a (complete)* $\mathscr{S}$*-automaton* $\mathcal{T}$
Question: *does* $\mathscr{S}(\mathcal{T})$ *(*$\mathscr{M}(\mathcal{T})$*) admit a (proper) length function?*

We highly suspect this problem to be undecidable and it seems likely that our construction can be adapted to show this.

Of course, it also remains open whether the freeness problem for automaton groups [19, 7.2 b)] is decidable.

References

1 Laurent Bartholdi, Thibault Godin, Ines Klimann, Camille Noûs, and Matthieu Picantin. A new hierarchy for automaton semigroups. *International Journal of Foundations of Computer Science*, 31(08):1069–1089, 2020. doi:10.1142/S0129054120420046.

2 Laurent Bartholdi and Ivan Mitrofanov. The word and order problems for self-similar and automata groups. *Groups, Geometry, and Dynamics*, 14:705–728, 2020. doi:10.4171/GGD/560.

3 Laurent Bartholdi and Pedro Silva. Groups defined by automata. In Jean-Éric Pin, editor, *Handbook of Automata Theory*, volume II, chapter 24, pages 871–911. European Mathematical Society, September 2021.

4 Paul Bell and Igor Potapov. Reachability problems in quaternion matrix and rotation semigroups. *Information and Computation*, 206(11):1353–1361, 2008. doi:10.1016/j.ic.2008.06.004.

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

6 Ievgen V. Bondarenko, Natalia V. Bondarenko, Said N. Sidki, and Flavia R. Zapata. On the conjugacy problem for finite-state automorphisms of regular rooted trees. *Groups, Geometry, and Dynamics*, 7:232–355, 2013. doi:10.4171/GGD/184.

7 Tara Brough and Alan J. Cain. Automaton semigroups: New constructions results and examples of non-automaton semigroups. *Theoretical Computer Science*, 674:1–15, 2017. doi:10.1016/j.tcs.2017.02.003.

8 Andrew M. Brunner and Said Sidki. The generation of $\mathrm{GL}(n, \mathbb{Z})$ by finite state automata. *International Journal of Algebra and Computation*, 08(01):127–139, 1998. doi:10.1142/S0218196798000077.

9 Alan J. Cain. Automaton semigroups. *Theoretical Computer Science*, 410(47):5022–5038, 2009. doi:10.1016/j.tcs.2009.07.054.

10 Julien Cassaigne, Tero Harju, and Juhani Karhumäki. On the undecidability of freeness of matrix semigroups. *International Journal of Algebra and Computation*, 09(03n04):295–305, 1999. doi:10.1142/S0218196799000199.

11 Daniele D'Angeli, Dominik Francoeur, Emanuele Rodaro, and Jan Philipp Wächter. Infinite automaton semigroups and groups have infinite orbits. *Journal of Algebra*, 553:119–137, 2020. doi:10.1016/j.jalgebra.2020.02.014.

12 Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. Automaton semigroups and groups: On the undecidability of problems related to freeness and finiteness. *Israel Journal of Mathematics*, 237:15–52, 2020. doi:10.1007/s11856-020-1972-5.

13 Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. Erratum to "semigroups and groups: On the undecidability of problems related to freeness and finiteness". *Israel Journal of Mathematics*, 245:535–542, 2021. doi:10.1007/s11856-021-2206-1.

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

15 Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. On the structure theory of partial automaton semigroups. *Semigroup Forum*, pages 51–76, 2020. doi:10.1007/s00233-020-10114-5.

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

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

18 Yair Glasner and Shahar Mozes. Automata and square complexes. *Geometriae Dedicata*, 111:43–64, 2005. doi:10.1007/s10711-004-1815-2.

19 Rostislav I. Grigorchuk, Volodymyr V. Nekrashevych, and Vitaly I. Sushchanskĭi. Automata, dynamical systems, and groups. *Proceedings of the Steklov Institute of Mathematics*, 231:128–203, 2000.

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

21 Rostislav I. Grigorchuk and Andrzej Żuk. The lamplighter group as a group generated by a 2-state automaton, and its spectrum. *Geometriae Dedicata*, 87:209–244, 2001. doi:10.1023/A:1012061801279.

22 Narain Gupta and Saïd Sidki. On the burnside problem for periodic groups. *Mathematische Zeitschrift*, 182(3):385–388, 1983.

23 John M. Howie. *Fundamentals of Semigroup Theory.* London Mathematical Society Monographs. Clarendon Press, 1995.

24 Kate Juschenko. *Amenability of discrete groups by examples.* American Mathematical Society, 2022.

25 David A. Klarner, Jean-Camille Birget, and Wade Satterfield. On the undecidability of the freeness of integer matrix semigroups. *International Journal of Algebra and Computation*, 1(2):223–226, 1991.

26 Ines Klimann. Automaton semigroups: The two-state case. *Theory of Computing Systems*, 58:664–680, 2016. doi:10.1007/s00224-014-9594-0.

27 Ines Klimann. To Infinity and Beyond. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 131:1–131:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2018.131.

28 Maximilian Kotowsky and Jan Philipp Wächter. The word problem for finitary automaton groups. In Henning Bordihn, Nicholas Tran, and György Vaszil, editors, *Descriptional Complexity of Formal Systems*, pages 94–108, Cham, 2023. Springer Nature Switzerland.

29 Roger Lyndon and Paul Schupp. *Combinatorial Group Theory.* Classics in Mathematics. Springer, 2001. First edition 1977.

30 Tara Macalister Brough, Jan Philipp Wächter, and Janette Welker. Automaton semigroup free products revisited. *arXiv preprint*, 2023. doi:10.48550/arXiv.2003.12810.

31 Arnaldo Mandel and Imre Simon. On finite semigroups of matrices. *Theoretical Computer Science*, 5(2):101–111, 1977. doi:10.1016/0304-3975(77)90001-9.

32 Turlough Neary. Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 649–661, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2015.649.

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

34 Matthieu Picantin. Automatic Semigroups vs Automaton Semigroups. 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 124:1–124:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2019.124.

35 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–269, 1946. doi:10.1090/s0002-9904-1946-08555-9.

36 Emanuele Rodaro and Jan Philipp Wächter. The self-similarity of free semigroups and groups. In Munehiro Iwami, editor, *Logic, Algebraic system, Language and Related Areas in Computer Science*, volume 2229 of *RIMS Kôkyûroku*, pages 11–20. Research Institute for Mathematical Sciences, Kyoto University, 2022. doi:10.48550/arXiv.2205.10248.

37 Pedro V. Silva and Benjamin Steinberg. On a class of automata groups generalizing lamplighter groups. *International Journal of Algebra and Computation*, 15(05n06):1213–1234, 2005. doi:10.1142/S0218196705002761.

38 Rachel Skipper and Benjamin Steinberg. Lamplighter groups, bireversible automata, and rational series over finite rings. *Groups, Geometry and Dynamics*, 14(2):567–589, 2020. doi:10.4171/GGD/555.

39 Benjamin Steinberg, Mariya Vorobets, and Yaroslav Vorobets. Automata over a binary alphabet generating free groups of even rank. *International Journal of Algebra and Computation*, 21(01n02):329–354, 2011. doi:10.1142/S0218196711006194.

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

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

42 Mariya Vorobets and Yaroslav Vorobets. On a series of finite automata defining free transformation groups. *Groups, Geometry, and Dynamics*, 4:337–405, 2010. doi:10.4171/GGD/87.

43 Jan Philipp Wächter. *Automaton Structures – Decision Problems and Structure Theory*. Doctoral thesis, Institut für Formale Methoden der Informatik, Universität Stuttgart, 2020. doi:10.18419/opus-11267.

44 Jan Philipp Wächter and Armin Weiß. *Automata and Languages – GAGTA Book 3*, chapter “The Word Problem for Automaton Groups”. DeGruyter, 2024. In preparation.

Nearly-Tight Bounds for Flow Sparsifiers in Quasi-Bipartite Graphs

Syamantak Das
IIIT Delhi, India

Nikhil Kumar
University of Waterloo, Canada

Daniel Vaz
LAMSADE, CNRS, Université Paris-Dauphine, Université PSL, France

Abstract

Flow sparsification is a classic graph compression technique which, given a capacitated graph G on k terminals, aims to construct another capacitated graph H, called a *flow sparsifier*, that preserves, either exactly or approximately, every *multicommodity flow* between terminals (ideally, with size as a small function of k). Cut sparsifiers are a restricted variant of flow sparsifiers which are only required to preserve maximum flows between bipartitions of the terminal set. It is known that exact cut sparsifiers require $2^{\Omega(k)}$ many vertices [Krauthgamer and Rika, SODA 2013], with the hard instances being *quasi-bipartite* graphs, where there are no edges between non-terminals. On the other hand, it has been shown recently that exact (or even $(1+\varepsilon)$-approximate) flow sparsifiers on networks with just 6 terminals require unbounded size [Krauthgamer and Mosenzon, SODA 2023, Chen and Tan, SODA 2024].

In this paper, we construct exact flow sparsifiers of size 3^{k^3} and exact cut sparsifiers of size 2^{k^2} for quasi-bipartite graphs. In particular, the flow sparsifiers are contraction-based, that is, they are obtained from the input graph by (vertex) contraction operations. Our main contribution is a new technique to construct sparsifiers that exploits connections to polyhedral geometry, and that can be generalized to graphs with a small separator that separates the graph into small components. We also give an improved reduction theorem for graphs of bounded treewidth [Andoni et al., SODA 2011], implying a flow sparsifier of size $\mathrm{O}(k \cdot w)$ and quality $\mathrm{O}\left(\frac{\log w}{\log\log w}\right)$, where w is the treewidth.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Network flows; Theory of computation → Sparsification and spanners; Theory of computation → Data compression

Keywords and phrases Graph Sparsification, Cut Sparsifiers, Flow Sparsifiers, Quasi-bipartite Graphs, Bounded Treewidth

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.45

Related Version *Full Version*: https://arxiv.org/abs/2407.09433

Funding *Daniel Vaz*: This work was funded by grants ANR-19-CE48-0016 (Algoridam) and ANR-21-CE48-0022 (S-EX-AP-PE-AL) from the French National Research Agency (ANR), and partially done while the author was at École Normale Superieure and IRIF.

1 Introduction

Graph sparsification is a classic and influential technique in algorithm design. The idea behind graph sparsification is to compress a given graph into a "smaller" graph (the notion of small depends on the context) which preserves certain crucial properties of the graph. The notion of edge sparsification dates back to the work of Gomory and Hu [21] and to Nagamochi and Ibaraki [32], who developed techniques to find a sparser graph – that is, with fewer edges – preserving s-t-cuts and k-edge-connectivity, respectively. This work was continued by Benczur and Karger [5] and Spielman and Teng [35], who extended the techniques to preserving cut values, or more generally the Laplacian spectrum, up to a factor of $1+\varepsilon$.

Relatively recent is the study of vertex sparsification. Arguably, the most extensively explored notions here are flow sparsification and cut sparsification. Specifically, suppose we are given an undirected graph $G = (V, E)$ along with a capacity function u on the edges and a subset of vertices T called terminals, with $|T| = k$. A cut sparsifier H of quality $q \geq 1$ is a graph on (potentially) fewer vertices which preserves the minimum cut value between every possible bipartition of the terminal set, up to a factor of q. A more general notion is flow sparsification where the sparsifier must preserve all multicommodity flows between the terminals (formal definitions are introduced in Section 2). Hence, we can see cut sparsification as a special case where it is only required to preserve the single-commodity flows between bipartitions of terminals.

The main focus in cut and flow sparsification research is to strike the ideal trade-off between the size of H and its quality q. In their seminal work, Moitra [31] and later Moitra and Leighton [29] showed that there is a flow sparsifier on just the terminal set (that is with size k) of quality $\mathrm{O}\big(\frac{\log k}{\log \log k}\big)$, and their work was later made constructive by different works [9, 16, 30]. These works also showed that any sparsifier of size k would have a quality loss of at least $\Omega(\sqrt{\log k / \log \log k})$ [30]. Hence, a significant improvement in the quality would either require more vertices in the sparsifier or special properties of the graph.

Considerable research effort has been dedicated to cut and flow sparsification in more restricted settings. For instance, one can construct a flow sparsifier on only the terminals with quality $\mathrm{O}(r)$ for graphs that exclude $K_{r,r}$ as a minor by exploiting connections between flow sparsification and the 0-extension problem [8, 31]. For general graphs, Chuzhoy [12] gave a construction with quality $O(1)$ and size $C^{\mathrm{O}(\log \log C)}$, where C is the total capacity of edges incident on terminals (and hence might be as large is $\Omega(nk)$). On the other hand, one can construct quality-1 cut sparsifiers of size $\mathrm{O}(2^{2^k})$ for general graphs [22, 24], of size $\mathrm{O}(k^2 \cdot 2^k)$ for planar graphs [27] and $\mathrm{O}(k) \cdot 2^{2^{\mathrm{O}(w)}}$ for graphs with treewidth w [10] (such sparsifiers are also known as mimicking networks or exact sparsifiers in the literature). The scenario is drastically different for exact flow sparsifiers. Recent breakthroughs have ruled out the existence of exact flow sparsifiers [26], as well as contraction-based quality-$(1 + \varepsilon)$ flow sparsifiers [11], with size as a function of k, which is achieved by demonstrating hard instances on 6 terminal networks. On the other hand, contraction-based flow sparsifiers of quality $1 + \varepsilon$ and size $2^{\mathtt{poly}(1/\varepsilon)}$ exist for every 5-terminal network [11].

For the special case of *quasi-bipartite graphs* [34], where non-terminals form an independent set, Andoni et al. [2] improved the bound of Chuzhoy significantly: they give a quality-$(1 + \varepsilon)$ flow sparsifier of size $\mathrm{poly}(k/\varepsilon)$, recently improved to size $k \cdot \mathrm{poly}(\log k, \varepsilon^{-1})$ [1, 23]. The significance of these result lies in the fact that for the simpler case of cut sparsification, these graphs present some of the hardest known instances [24, 27], where a quality-1 cut sparsifier requires $2^{\Omega(k)}$ vertices [24, 27]. Thus, their result raises hope that we can overcome the lower bounds by designing quality-$(1 + \varepsilon)$ sparsifiers, even in the flow sparsification setting.

Andoni et al. also prove a second result where they show a generic reduction from graphs of bounded treewidth to general graphs in the following sense: They give a construction whereby the existence of any quality-$q(k)$ sparsifier of size $S(k)$ implies a quality-$q(6w)$ sparsifier of size $k^4 \cdot S(6w)$ where w is the treewidth of the graph.

Our Results. We give the following results on quasi-bipartite graphs and their extensions:

1. A cut sparsifier of size 2^{k^2} for quasi-bipartite graphs (Theorem 3.1);
2. A contraction-based flow sparsifier of size 3^{k^3} for quasi-bipartite graphs (Theorem 3.2);
3. A cut sparsifier of size $k + 2^{c^2}$ and a flow sparsifier of size $k + 3^{c^3}$ when the graph has a vertex cover of size c (Theorem 4.1);
4. A cut sparsifier of size $kd + 4^{d^3}$ when G has vertex integrity d [4, 19], that is, a separator $X \subseteq V$ such that $|X| + |C| \leq d$ for every component C of $G - X$ (Corollary 4.3).

Note that our result almost matches the lower bound given by Krauthgamer and Rika [27] (up to a polynomial in the exponent). Further, our result on flow sparsifiers shows that instances on quasi-bipartite graphs are not hard for flow sparsification, as they admit better bounds than general graphs.

Our main contribution lies in developing a novel tool for constructing sparsifiers that is based on connections to polyhedral geometry. We show that this technique can be applied to obtain cut and flow sparsifiers, and even when the terminal set separates the graph into small components. Furthermore, we show that the size of the sparsifier actually grows with the size of the separator whose removal leaves only small components, thus obtaining improved results for bounded vertex cover and vertex integrity, two structural graph parameters that have recently gained popularity in the parameterized community [7, 18, 19, 20, 28, 33]. These have particular relevance when studying problems that are hard for more general parameters, such as treewidth [17] and treedepth [7, 19]; they also allow for stronger meta-theorems [20, 28] compared to the classic theorem of Courcelle for bounded-treewidth graphs [14].

We give an additional result for graphs with treewidth w, improving the results of Andoni et al. [2] and Chaudhuri et al. [10]: we construct a flow sparsifier of size $k \cdot S(2w)$ and quality $g(2w)$ provided that every k-terminal network admits a quality $g(k)$ flow sparsifier with size $S(k)$ (see Section 5). This implies an $\mathrm{O}(\frac{\log w}{\log \log w})$-quality sparsifier with size $\mathrm{O}(k \cdot w)$ for graphs with treewidth w using results from [16, 29].

Due to space constraints, we defer some proofs to the full version; the corresponding lemmas are marked with an asterisk (*).

Techniques. The main idea behind our construction of cut and flow sparsifiers for bipartite graphs is to consider them as a union of stars centered on Steiner vertices, which can be handled independently as they do not share any edges. By showing that the number of different ways that stars can participate in cuts is bounded, we get a sparsifier with the same bound on the size, as equivalent stars can be contracted together. Thus, it is sufficient to show how to put stars into a bounded number of classes.

Our approach is based on polyhedral theory. We represent each star by a vector in $\mathbb{R}^k_{\geq 0}$ where each coordinate is the capacity of an edge between the center of the star and a terminal. We show that there are 2^{k^2} stars, which we refer to as *basic stars*, such that any star is the conic combination of at most k of them. Using this idea, we obtain two constructions: the first is to construct H from the terminals and the set of *basic stars* with appropriately scaled-up capacities; whereas the second (slightly larger) is to simply contract vertices that are the conic combination of the same set of basic stars. Since the second construction is contraction-based, a consequence of our results is that optimal algorithm for contraction-based sparsifiers presented by Khan et al. [24] obtains a sparsifier of size at most 2^{k^3} for bipartite graphs.

The construction of the flow sparsifiers is much more involved. A first attempt would be to use our result for cut sparsifier along with a result from Andoni et al. [2, Theorem 7.1] which roughly implies that if the flow-cut gap for the given graph is γ, then an exact contraction-based cut sparsifier is also a flow sparsifier of quality γ. Unfortunately, bipartite networks can have a flow-cut gap of $\Omega(\log k)$ and hence this approach fails.

We rather take a more direct approach: by relying on the above mentioned polyhedral tool to define equivalence classes on the stars, we show that we can contract the stars in each equivalence class into a single one. Applying this technique is much more challenging in the case of flow sparsifiers, since one has to ensure that every multicommodity flow between terminals (and not just bipartitions) must be preserved.

The main technical difficulty is to show that merging two stars preserves the routing of multicommodity flows, and in particular, that if a demand can be routed in the merged star, then it can also be routed in the original stars (if the original stars are in the same equivalence class). We achieve this by splitting the demand so that each part can be routed in a different star, by a process which iteratively adds demand to one of the two stars, and if that is no longer possible, refines the partition by globally switching demands between the two stars. We show that when the process can no longer introduce new demand, then there is a saturated cut in the merged star, and thus all of the demand must be already routed.

2 Preliminaries

A network $G = (V, E, u)$ is a graph (V, E) with edge capacities $u\colon E(G) \to \mathbb{R}_{\geq 0}$. It is usually associated with a set of *terminals* $K \subseteq V(G)$, whose size we denote by k. We refer to vertices in $V(G) \setminus K$ as *non-terminal* or *Steiner* vertices, and say that two networks G_1, G_2 are *Steiner-disjoint* if $V(G_1) \cap V(G_2) \subseteq K$.

We consider a cut to be a subset of vertices $X \subset V$, with cut edges $\delta(X) = E(X, V - X)$. For convenience, we usually write $u(X)$ to mean $u(\delta(X))$ for any $X \subseteq V$. A cut X *separates* $A \subseteq K$ if $A \subseteq X$ and $K - A \subseteq V - X$, and it is a min-cut for A if it minimizes the capacity among all cuts separating A. We denote by $\mathrm{mc}_G(A)$ the smallest (minimum $|X|$) min-cut (in G) that separates A, and $\kappa_G(A)$ its capacity. If the network is clear from context, we drop the subscript in $\mathrm{mc}(A)$, $\kappa(A)$.

Cut sparsifiers. A *cut sparsifier* of quality $q \geq 1$ for a network G with terminals K is a network H such that $K \subseteq V(H)$ and for every subset $A \subset K$, the capacity of the min-cut separating A is q-approximated, that is,

$$\kappa_G(A) \leq \kappa_H(A) \leq q \cdot \kappa_G(A).$$

Unless specified, a cut sparsifier is of quality 1.

Flow sparsifiers. A *flow sparsifier* of quality $q \geq 1$ for a network G with terminals K is a network H that q-approximately preserves the multi-commodity flows for any demand. We use the formal description in the work of Andoni et al. [2].

We say that a demand $\mathbf{d} \in \mathbb{R}^{K \times K}_{\geq 0}$ *is routed* in G by flow $f \geq 0$ if $\sum_{P \in \mathcal{P}_{s,t}} f_P = d(s,t)$ and $\sum_{P \ni e} f_P \leq u(e)$, where f is defined over paths and $\mathcal{P}_{s,t}$ is the set of all s-t-paths. We consider both demands and paths as symmetric, that is, $d(s,t) = d(t,s)$, but $\mathcal{P}_{s,t} = \mathcal{P}_{t,s}$ so the same flows can satisfy both demands.

The demand polytope [2] for a network is the set all demands that can be routed in G, $\mathcal{D}(G) = \{\mathbf{d} : \mathbf{d} \text{ can be routed in } G\}$. Given a demand vector $\mathbf{d}$, its *flow factor* is the value $\lambda_G(\mathbf{d}) = \sup\{\lambda \geq 0 : \lambda\mathbf{d} \in \mathcal{D}(G)\}$.

We formally define *flow sparsifiers* as follows: H is a quality-q flow sparsifier for G with terminals K if for all demand vectors $\mathbf{d}$,

$$\lambda_G(\mathbf{d}) \leq \lambda_H(\mathbf{d}) \leq q \cdot \lambda_G(\mathbf{d}).$$

In particular for $q = 1$, we have that $\mathbf{d} \in \mathcal{D}(G)$ if and only if $\mathbf{d} \in \mathcal{D}(H)$.

Treewidth. A tree decomposition $(T, \mathcal{B})$ of a graph G is a tree T together with a collection of subsets of vertices $\mathcal{B} = \{B_i\}_{i \in V(T)}$ called *bags*, such that:

- For each edge $uv \in E(G)$, there is a bag B_i containing both u and v.
- For each vertex $v \in V(G)$, the collection of bags containing v induces a non-empty subtree of T.

The *width* of $(T, \mathcal{B})$ is $w(T, \mathcal{B}) = \max_{i \in V(T)}(|B_i| - 1)$. The *treewidth* of G is the minimum width achievable by any tree decomposition.

We assume that there are no two identical bags in the decomposition, as otherwise we can simply contract the edge connecting the corresponding nodes. We consider that each edge uv is associated with a single node of T, namely the node closest to the root whose bag contains both u and v. The collection of edges associated with a node $i \in T$ is denoted by E_i, and the subgraph induced by a bag is $G[i] = (B_i, E_i)$. This notation is particularly useful when talking about the graph induced by collections of bags, and thus for a subset $R \subseteq V(T)$ we write $B(R) = \bigcup_{i \in R} B_i$, $E(R) = \bigcup_{i \in R} E_i$, and $G[R] = (B(R), E(R))$. We remark that $G[R]$ and $G[B(R)]$ are subgraphs on the same subset of vertices but with different sets of edges.

Computing the treewidth of a graph, together with the corresponding tree decomposition, is an NP-hard problem [3], but can be computed in time $w^{O(w^3)} \cdot n$ [6] for a graph of treewidth w. For a faster running time, we can get a tree decomposition with width $2w + 1$ in time $2^{O(w)} \cdot n$ due to the recent work by Korhonen [25]. For a more detailed introduction to treewidth, see e.g. the book by Cygan et al. [15].

2.1 Basic Tools

▶ **Lemma 2.1** **(*).** *Let G be a network. If H is a quality-q sparsifier for G with terminals K, and L is a quality-r sparsifier for H with terminals K', $K' \subseteq K$, then L is a quality-qr sparsifier for G with terminals K', where the statement works if H, L are cut sparsifiers or flow sparsifiers.*

We recall the splicing lemma of Andoni et al. [2] for flow sparsifiers, which shows that it is sufficient for a sparsifier to preserve routings along terminal-free paths.

▶ **Lemma 2.2** ([2, Lemma 5.1])**.** *Let G and H be two networks with the same set of terminals K, and fix $\rho \geq 1$. Suppose that whenever a demand $\mathbf{d}$ between terminals in K can be routed in G using terminal-free flow paths, demand $\mathbf{d}/\rho$ can be routed in H (by arbitrary flow paths).*

Then for every demand $\mathbf{d}$ between terminals in K that can be routed in G, demand $\mathbf{d}/\rho$ can be routed in H.

Finally, we show the following generalization of the composition lemma to both cut and flow sparsifiers.

▶ **Lemma 2.3** **(*).** *Let G_1 and G_2 be Steiner-disjoint networks for terminal set K.*

If H_1 and H_2 are quality-q (cut or flow) sparsifiers for G_1, and G_2 with terminal set $K \cap V(G_1)$, $K \cap V(G_2)$, respectively, then $H := H_1 \uplus H_2$ is a quality-q (cut or flow, resp.) sparsifier for $G := G_1 \uplus G_2$ (parallel edges in K are joined and their capacities summed).

The proof for flow sparsifiers is given by Andoni et al. [2, Lemma 5.2]; the proof for cut sparsifiers follows using similar arguments.

Let G/vw be the network obtained from G by contracting v and w into a vertex denoted vw, that is, removing v and w, adding a vertex vw with edges to vertices $(N(v) \cup N(w)) \setminus \{v, w\}$, and setting the capacity of each new edge $\{vw, x\}$ to $w_{G/vw}(\{vw, x\}) = u_G(vx) + u_G(wx)$, where $u_G(vx) = 0$ if the edge does not exist.

▶ **Lemma 2.4** **(*).** *Let v, w be vertices such that for every $A \subseteq K$, there is a min-cut X separating A that either contains both v and w or neither of them.*

Then G/vw is an exact cut sparsifier for G with terminals K.

3 Sparsifiers for Quasi-Bipartite Graphs

In this section we show how to compute cut and flow sparsifiers for quasi-bipartite graphs, where the left side is the set of terminals. In a later section we show how to improve this to a more general case of bounded vertex cover. Our results are formalized in the following theorems:

▶ **Theorem 3.1.** *Let $G = (V, E, u)$ be a network with terminal set K of size k.*
If G is bipartite with partition $V = K \uplus (V \setminus K)$, then G has a cut sparsifier of size 2^{k^2} and a contraction-based cut sparsifier of size 2^{k^3}.

▶ **Theorem 3.2.** *Let $G = (V, E, u)$ be a network with terminal set K of size k.*
If G is bipartite with partition $V = K \uplus (V \setminus K)$, then G has a contraction-based flow sparsifier of size 3^{k^3}.

We remark that quasi-bipartite and bipartite graphs are equally hard to handle, as we can simply consider a quasi-bipartite graph as the Steiner-disjoint union of $G[K]$ and $G - E(K)$.

▶ **Corollary 3.3.** *Quasi-bipartite networks (with no edges between Steiner vertices) have cut sparsifiers of size 2^{k^2} and flow sparsifiers of size 3^{k^3}.*

3.1 Cut Sparsifiers

This section is dedicated to proving Theorem 3.1.

Let v be the center of a star, and $(c(1), c(2), \dots, c(k))$ be the capacity vector of the edges to the terminals (with some ordering $K = \{t_1, \dots, t_k\}$). For each cut $S \subseteq K$, either $c(S) \leq c(K - S)$, $c(S) = c(K - S)$, or $c(S) \geq c(K - S)$. Using the inequalities for c and each subset $S \subseteq K$, we can define a polyhedron P_c of all the capacity vectors that cut the star in the same way as c. Thus, the capacity vector c is a conic combination of the extreme rays of P_c, and therefore we can replace it in the graph by a conic combination of the stars corresponding to the extreme rays, which we call *basic stars*. Finally, we show that since c agrees on every inequality with the basic stars in its conic combination, the replacement preserves the value of min-cuts, and so by replacing every star we obtain a sparsifier of G.

Basic stars. Let $c \in \mathbb{R}^k$ be a capacity vector. We define $\mathcal{S}_c$ to be the collection of subsets $S \subseteq K$ such that $c(S) \leq c(K - S)$, i.e. $\mathcal{S}_c = \{S \subseteq K : c(S) \leq c(K - S)\}$, and the *star cone* of c to be:

$$P_c := \left\{ x \in \mathbb{R}^k_{\geq 0} : \ x(S) \leq x(K - S) \ \forall S \in \mathcal{S}_c \right\}$$

We say that a vector x *agrees with* c (on every cut) if $x(S) \leq x(K - S)$ for all $S \in \mathcal{S}_c$, and thus P_c is the polyhedron containing all of the capacity vectors that agree with c. It is a cone since it is defined by constraints of the form $\alpha^T x \leq 0$. We remark that for every $S \subseteq K$, $S \in \mathcal{S}_c$ or $K - S \in \mathcal{S}_c$, and both are present if and only if $x(S) = x(K - S)$.

We define the set of basic stars as stars constructed from extreme rays of any such cone. For a given c, the extreme rays of the cone P_c are found at the intersection of $k - 1$ tight inequalities: $x_i \geq 0$ for some indices $i \in I$, and $x(S) = x(K - S)$ for some $S \in J$, with $|I| + |J| = k - 1$ (see e.g. [13, Sec. 3.12]). Notice that, regardless of the capacity vector c, the extreme rays of P_c are all found using a tight subset of the same collection of inequalities. However, not every extreme ray belongs to every P_c, as they might disagree on some inequalities outside of J. For convenience, we represent each ray by a vector with coordinates summing to 1.

Let Q be the set of extreme rays of any cone as obtained above, that is, the set of extreme rays obtained from intersection of $k-1$ independent tight constraints with $x(K)=1$. Formally, let $\mathcal{I}_k$ be the collection of pairs (I,J), $|I|+|J|=k-1$, such that the constraints $x(i)=0$ for $i\in I$, $x(S)=x(K-S)$ for $S\in J$ and $x(K)=1$ are all independent. Then

$$Q=\big\{q_{IJ}\in\mathbb{R}^k_{\geq 0}: q_{IJ}(K)=1; q_{IJ}(I)=0; \\ q_{IJ}(S)=q_{IJ}(K-S)\ \forall S\in J; (I,J)\in\mathcal{I}_k\big\}.$$

For each $q\in Q$, we can construct a star with center denoted v_q and an edge to each terminal $t_i\in K$ with capacity q_i. These stars are denoted *basic stars* and are referred to by their center v_q.

The size of Q is determined by the possible sets of inequalities that define each of its elements. As the tight inequalities for S and $K\setminus S$ are the same, there are effectively at most $2^{k-1}+k\leq 2^k$ inequalities to choose from. Each element of Q is defined by $k-1$ of these, and thus $|Q|\leq\binom{2^k}{k-1}\leq 2^{k^2}$.

▶ **Lemma 3.4.** *Any capacity vector c can be written as the conic combination of at most k points in Q, all of which agree with c.*

Formally, there are $q_1,\ldots,q_k\in Q$, $\lambda(q_1),\ldots,\lambda(q_k)\geq 0$ such that :

$$\sum_{i=1}^{k}\lambda(q_i)\cdot q_i=c \qquad \textit{and} \qquad q_i(S)\leq q_i(K-S) \qquad \forall i\in[k], S\in\mathcal{S}_c$$

Proof. Let P_c be the star cone corresponding to c. By Carathéodory's theorem (see e.g. [13, Sec. 3.14]), c is a conic combination of at most k extreme rays of P_c (which are contained in Q). In other words, there are $q_1,\ldots,q_k\in P_c\cap Q$ and $\lambda(q_1),\ldots,\lambda(q_k)\geq 0$ such that $c=\sum_{i=1}^{k}\lambda(q_i)\cdot q_i$.

All it remains to show is that every q_i agrees with c on every cut. Indeed, since every q_i is an extreme ray of P_c, it must satisfy the inequalities defining P_c, and thus agree with c. ◀

We obtain a sparsifier for G as follows: for each $v\in V-K$, write the capacity vector c_v of the star centered at v as a conic combination of the points in Q, i.e. as $c_v=\sum_{q\in Q}\lambda_v(q)q$. Then, define $V_Q=\{v_q:q\in Q\}$ and take $H=(K\cup V_Q, K\times V_Q, u')$; the capacity vector for each v_q is q scaled up by the sum of the corresponding values λ_v: $u'(v_q,\cdot)=q\cdot\sum_v\lambda_v(q)$.

To determine the values λ_v, we can use the constructive proof of Carathéodory's theorem, starting by computing the set Q by enumeration in time $2^{k^2}\cdot\text{poly}(k)$. Given a capacity vector c, start by finding $q_1\in Q\cap P_c$ that agrees with c on every cut; then, find the maximum value of $\lambda(q_1)$ such that $c-\lambda(q_1)q_1\in P_c$; finally, set $c'=c-\lambda(q_1)q_1$ and repeat the process for c' to find the remaining $q_2,\ldots$ and $\lambda_2,\ldots$ Notice that at each step, c' has at least one more tight inequality than c (otherwise we could increase $\lambda(q_1)$), and thus the process stops after k iterations. In conclusion, the process of computing the set Q takes time $2^{O(k^2)}$, and the process of finding the coefficients for each star takes time $2^{O(k^2)}$ (per star).

The following lemma allows us to relate a star to its conic combination using basic stars.

▶ **Lemma 3.5.** *Let v be a vertex with degree k and capacity vector c for its incident edges.*

If c can be written as the conic combination of points $q_1,q_2,\ldots,q_\ell\in Q$, all of which agree with c, then G has a sparsifier given by $G-\{v\}+\{w_1,\ldots,w_\ell\}$, where the neighbors of vertices $w_1,\ldots,w_\ell$ are also neighbors of v and the capacities of the edges incident on each vertex w_i are given by $\lambda(q_i)\cdot q_i$.

Proof. We will use the fact that the definition of sparsifier is reflexive, and thus show that G is a sparsifier for $G - \{v\} + \{w_1, \ldots, w_\ell\}$. We will iteratively contract two vertices in $\{w_1, \ldots, w_\ell\}$, until we have the original graph. We assume that each vertex w_i has the same neighbors as v by adding edges with capacity 0.

Let $G' = G - \{v\} + \{w_1, \ldots, w_\ell\}$ so that $u_G(v) = \sum_{i=1}^{\ell} u_{G'}(w_i)$. For any $i \in [\ell]$, q_i agrees with c, so $u_{G'}(w_i) = \lambda(q_i) \cdot q_i$ also agrees with c, as P_c is a cone (and thus scale-invariant). We observe that if some $A \in \mathcal{S}_c$, then $c(A) \leq c(K - A)$, and thus there is a min-cut separating A that does not contain v, as it is at least as cheap to cut the edges to A as to $K - A$; the same argument applies to any vertex whose capacities agree with c, such as any w_i. Therefore, for any $A \subseteq K$, there is a min-cut separating A that contains both $w_{\ell-1}$ and w_ℓ (if $K - A \in \mathcal{S}_c$) or neither (if $A \in \mathcal{S}_c$), and thus we can apply Lemma 2.4 to merge $w_{\ell-1}$ and w_ℓ to obtain a new vertex $w'_{\ell-1}$ with capacities $u_{G'}(w_{\ell-1}) + u_{G'}(w_\ell)$. This ensures that $u_G(v) = \sum_{i=1}^{\ell-2} u_{G'}(w_i) + u_{G'}(w'_{\ell-1})$, and thus we can repeat the process until we only have a single w'_1 left, with capacities $u_{G'}(w'_1) = \sum_{i=1}^{\ell} u_{G'}(w_i) = u_G(v)$, which is equivalent to the original graph. ◀

We can now finish the proof of Theorem 3.1.

Proof of Theorem 3.1. We start by computing the basic stars Q for k terminals, and then computing the coefficients $\lambda_v : Q \to \mathbb{R}_{\geq 0}$ for each $v \in V \setminus K$. We can now obtain two different constructions for a sparsifier, one that replaces all the vertices with basic stars, and another which simply contracts vertices that have the same cut profile.

Basic star sparsifier. We take $V_Q = \{v_q : q \in Q\}$ as the set of all basic stars, and construct our sparsifier by adding edges from every $v_q \in Q$ to every terminal $t \in K$. The capacities are given by summing over the λ_v as follows: $u'(v_q, t_i) = q_i \cdot \sum_v \lambda_v(q)$. The sparsifier is then given by $H = (K \cup V_Q, K \times V_Q, u')$.

This construction is equivalent to decomposing each v into a conic combination of basic stars using Lemma 3.5, and then (iteratively) contracting all of the vertices created for the same $q \in Q$ (and different $v \in V \setminus K$) using Lemma 2.4. This also proves correctness of the sparsifier.

Contraction-based sparsifier. For this variant of the sparsifier, we will only use Lemma 2.4 and Carathéodory's theorem [13, Sec. 3.14], as well as the algorithm introduced by Hagerup et al. [22], in which we compute all 2^k minimum cuts, and contract any two vertices whose capacity vectors agree with each other.

The running time is $O(2^k \cdot n^2)$ to compute the set $\mathcal{S}_c$ for each of the at most n stars, and then comparing the sets for each pair of stars to decide whether to contract them. Alternatively, the sets $\mathcal{S}_c$ can be computed in time $O(2^k \cdot kn)$ and the vertices placed in buckets according to their set $\mathcal{S}_c$, after which the vertices in each bucket can be contracted.

Thus, all that we need to do is to show that if two capacity vectors are the conic combination of the same basic stars, then they agree, and thus can be contracted.

▶ **Lemma 3.6.** *If capacity vectors c and c' can be written as the conic combination of points $q_1, q_2, \ldots, q_\ell \in Q$ and each q_i agrees with both c and c', then c and c' agree with each other.*

Proof. As the lemma is symmetric, we will simply show that if $S \in \mathcal{S}_c$, then $S \in \mathcal{S}_{c'}$. If $S \in \mathcal{S}_c$, then $S \in \mathcal{S}_{q_i}$ and so $q_i(S) \leq q_i(K \setminus S)$ by definition of agreement. Therefore,

$$c'(S) - c'(K \setminus S) = \sum_i \lambda_{c'}(q_i) \cdot \big(q_i(S) - q_i(K \setminus S)\big) \leq 0 \quad \Rightarrow \quad S \in \mathcal{S}_{c'}.$$ ◀

Given Lemma 3.6, it is sufficient to bound the number of possible ways that capacity vectors can be written as conic combinations, as vectors that are written as a conic combination of the same basic stars are always on the same minimal min-cuts, and thus can be contracted by Lemma 2.4. As there are 2^{k^2} basic stars, and by Carathéodory's theorem each capacity vector can be written as the conic combination of at most k of them, there are at most 2^{k^3} such combinations, and thus after contraction, we are left with a sparsifier of size at most 2^{k^3}. ◀

3.2 Flow Sparsifiers

We will show that a slight modification to the contraction-based sparsifier in Section 3.1 increases its size to 3^{k^3} but makes it a flow sparsifier on bipartite graphs, proving Theorem 3.2.

Let c be a capacity vector. The only modification we need is to consider all inequalities of the form $c(A) \leq c(B)$ for $A, B \subseteq K$, $A \cap B = \emptyset$. Surprisingly, the cut sparsifiers of Theorem 3.1 preserve min-cuts separating a subset $A \subseteq K$ from a disjoint subset $B \subseteq K$ without requiring these inequalities. However, for the construction of flow sparsifiers it is necessary that the vertices we contract agree on the inequalities for each pair (A, B).

We define $\mathcal{S}'_c = \{(A, B) \subseteq K : c(A) \leq c(B)\}$ and say that c' *strongly agrees* with c if $c'(A) \leq c'(B)$ for all $(A, B) \in \mathcal{S}'_c$. Notice that $\mathcal{S}'_c$ has at most 3^k sets, as an element can be placed in A, B or neither.

The *strong star cone* of c is defined as $P'_c := \left\{x \in \mathbb{R}^k_{\geq 0} : \; x(A) \leq x(B) \; \forall (A, B) \in \mathcal{S}'_c\right\}$, and the set of extreme rays $Q' := \{q \in \mathbb{R}^k_{\geq 0} : q \text{ is an extreme ray of some } P'_c\}$. The size of Q' is upper-bounded by the possible combinations of $k - 1$ tight inequalities, which implies that $|Q'| \leq 3^{k^2}$.

We use the contraction-based construction of Section 3.1 using the concept of strong agreement. As before, we can show that each capacity vector is the conic combination of k extreme rays.

▶ **Lemma 3.7.** *Any capacity vector c can be written as the conic combination of at most k points in Q', all of which strongly agree with c.*

Formally, there are $q_1, \ldots, q_k \in Q'$, $\lambda(q_1), \ldots, \lambda(q_k) \geq 0$ such that :

$$\sum_{i=1}^{k} \lambda(q_i) \cdot q_i = c \qquad \text{and} \qquad q_i(A) \leq q_i(B) \qquad \forall i \in [k], (A, B) \in \mathcal{S}'_c$$

Proof. Let P'_c be the star cone corresponding to c. By Carathéodory's theorem, c is a conic combination of at most k extreme rays of P'_c (contained in Q'). Each q_i is in P'_c, and thus strongly agrees with c. ◀

As Q' has at most 3^{k^2} extreme rays, this means that there are at most 3^{k^3} classes where capacity vectors can be placed according to which set of extreme rays produce it as a conic combination. As part of our proof, we need to show that if two capacity vectors are in the same class, then they strongly agree.

▶ **Lemma 3.8.** *If capacity vectors c and c' can be written as the conic combination of points $q_1, q_2, \ldots, q_\ell \in Q'$ and both strongly agree with each q_i on every cut, then c and c' strongly agree.*

Proof. As the lemma is symmetric, we will simply show that if $(A, B) \in \mathcal{S}'_c$, then $(A, B) \in \mathcal{S}_{c'}$. If $(A, B) \in \mathcal{S}_c$, then $(A, B) \in \mathcal{S}_{q_i}$ and so $q_i(A) \leq q_i(B)$ by definition of agreement. Therefore,

$$c'(A) - c'(B) = \sum_i \lambda_{c'}(q_i) \cdot \big(q_i(A) - q_i(BS)\big) \leq 0 \quad \Rightarrow \quad (A, B) \in \mathcal{S}_{c'}. \qquad ◀$$

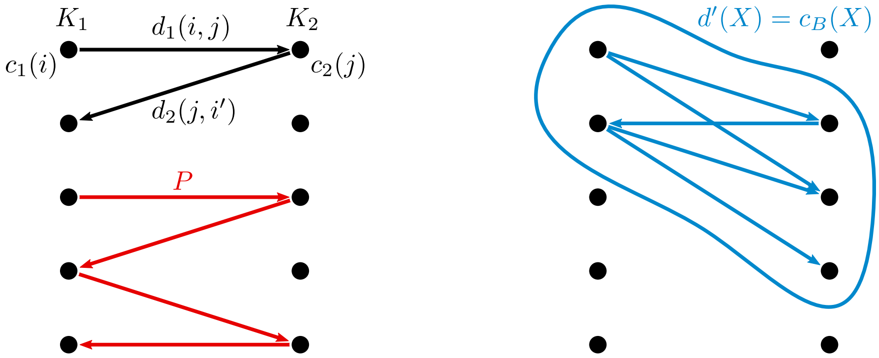


Figure 1 Representation of the graph G_B; a path (left, red) as found in Step 3a; a cut (right, blue) representing the situation in which no path is found.

We then show that if we have two stars where their capacity vectors agree, we can contract them to obtain a flow sparsifier. By repeating the process until we have at most 1 vertex per class, we obtain a flow sparsifier of size 3^{k^3}.

▶ **Lemma 3.9.** *Let v_1, v_2 be centers of stars with leaves K and let $c_1, c_2 \in \mathbb{R}^k_{\geq 0}$ be their capacity vectors (respectively).*

If c_1 and c_2 strongly agree with each other, that is $\mathcal{S}'_{c_1} = \mathcal{S}'_{c_2}$, then G/v_1v_2 is an exact flow sparsifier for G with terminals K.

Proof. Let $c := c_1 + c_2$, and $v := v_1v_2$ be the vertex created in $G' := G/v_1v_2$. We will show that if we can route a demand $\mathbf{d}$ in G then we can route it G' and vice-versa. If we can route a demand in G, then any demand routed on v_1w or v_2w can be routed instead on vw, as $c_1(w) + c_2(w) = c(w)$. We now show that a demand routed in G' can also be routed in G.

By Lemma 2.2, and since the neighbors of v_1 and v_2 are terminals, we can “splice” the flows so that any demand is routed only through (internally) terminal-free paths. Thus, we focus on the demands that are routed through paths t_ivt_j, $t_i, t_j \in K$ in G'. Let $\mathbf{d}$ be any such demand that can be routed in $K \cup \{v\}$, and we will construct vectors $\mathbf{d}_1$, $\mathbf{d}_2$ of the demands that will be routed through v_1, v_2, such that $\mathbf{d}_1 + \mathbf{d}_2 = \mathbf{d}$.

▶ **Observation 3.10.** *In a star $K \cup \{v\}$ with capacities c, a demand $\mathbf{d}$ can be routed if and only if for every $i \in [k]$, $\mathbf{d}(i) := \sum_j d(t_i, t_j) \leq c(i)$.*

We use the following directed bipartite graph G_B to assist us in the algorithm and proof: G_B has as vertices two copies of K, $V(G_B) = K_1 \cup K_2$, and has arcs $A(G_B) = (K_1 \times K_2) \cup (K_2 \times K_1)$. We also impose node capacities for outgoing arcs, with capacity vector c_1 for the vertices in K_1 and c_2 for vertices in K_2; we represent the capacity vector for G_B as c_B (so $c_B(K_1) = c_1$, $c_B(K_2) = c_2$). See Figure 1 for a visual representation of the graph.

Our goal is to add demand to the graph in the form of demands on arcs, with arcs from K_1 to K_2 representing $\mathbf{d}_1$, and arcs from K_2 to K_1 representing $\mathbf{d}_2$. We will use $\mathbf{d}'$ to represent the demand in the graph (so $\mathbf{d}_1 = \mathbf{d}'(K_1, K_2)$, $\mathbf{d}_2 = \mathbf{d}'(K_2, K_1)$), and $\tilde{\mathbf{d}}$ to represent leftover demand (initially $\mathbf{d}' = 0$, $\tilde{\mathbf{d}} = \mathbf{d}$). Whenever we say to add some demand to $d_\iota(i,j)$, it is implied that we add the demand to $d'(i_\iota, j_{3-\iota})$ and remove it from $\tilde{d}(i,j)$, as well as make the same changes to (j,i) (increase $d_\iota(j,i)$ and $d'(j_\iota, i_{3-\iota})$, decrease $\tilde{d}(j,i)$), where $i_1, j_1 \in K_1$, $i_2, j_2 \in K_2$ are the copies of i and j in K_1, K_2, respectively.

We repeat the following steps until $\tilde{\mathbf{d}} = 0$:

1. if there is $\tilde{d}(i,j) > 0$ such that $\mathbf{d}_1(i) < c_1(i)$ and $\mathbf{d}_1(j) < c_1(j)$, add to $d_1(i,j)$ a value of $\varepsilon := \min(\tilde{d}(i,j), c_1(i) - \mathbf{d}_1(i), c_1(j) - \mathbf{d}_1(j))$;
2. similarly for $\mathbf{d}_2$, if there is (i,j) such that $\varepsilon := \min(\tilde{d}(i,j), c_2(i) - \mathbf{d}_2(i), c_2(j) - \mathbf{d}_2(j)) > 0$, add ε to $d_2(i,j)$;
3. let $\tilde{d}(i,j) > 0$, and assume w.l.o.g. that $\mathbf{d}_2(i) = c_2(i)$:
   a. find a path $P = (i_2 = \ell_0, \ell_1, \ell_2, \dots \ell_p)$ such that for all $0 < r < p$, $\mathbf{d}'(\ell_r) = c_B(\ell_r)$, $d'(\ell_r, \ell_{r+1}) > 0$, and for the endpoint, $d'(\ell_p) < c_B(\ell_p)$;
   b. switch demand between $\mathbf{d}_1$ and $\mathbf{d}_2$ as follows: let $\varepsilon := \min\big(c_1(i) - \mathbf{d}_1(i), c_B(\ell_p) - \mathbf{d}'(\ell_p), \min_r d'(\ell_r, \ell_{r+1})\big)$; for any arc $(x_1, y_2) \in K_1 \times K_2$ (resp. $(x_2, y_1) \in K_2 \times K_1$) in P, decrease $d'(x_1, y_2)$ (resp. $d'(x_2, y_1)$) and increase $d'(x_2, y_1)$ (resp. $d'(y_1, x_2)$) by ε;
   c. add $\min(\tilde{d}(i,j), \varepsilon, c_2(j) - \mathbf{d}_2(j))$ to $d_2(i,j)$.

We will show that such a path always exists as long as there is demand to be routed, but first, let us show that these operations maintain the invariants that $\mathbf{d}'(i) \le c_B(i)$, for all $i \in V(G_B)$. For the first two steps, $\mathbf{d}_1(i)$ (resp. $\mathbf{d}_2(i)$) increases by at most $c_1(i) - \mathbf{d}_1(i)$ (resp. $c_2(i) - \mathbf{d}_2(i)$), which maintains the invariant. For the third step, notice that every ℓ_i except the endpoints has two arcs in the path, one corresponding to $\mathbf{d}_1$ and the other to $\mathbf{d}_2$, hence the decrease on $\mathbf{d}_1$ on one edge is compensated by the increase on the other, and the same for $\mathbf{d}_2$. Furthermore, as we choose ε to be the smallest value on the path, no demand can go below 0. Finally, for the endpoints, we know that ℓ_p has spare capacity by definition, and for i we decrease $\mathbf{d}_2(i)$ and increase $\mathbf{d}_1(i)$, but $\mathbf{d}_2(i) = c_2(i)$ implies that $\mathbf{d}_1(i) < c_1(i)$, as $\tilde{\mathbf{d}}(i) + \mathbf{d}_1(i) + \mathbf{d}_2(i) = \mathbf{d}(i) \le c(i) = c_1(i) + c_2(i)$ and $\tilde{\mathbf{d}}(i) > 0$.

We will now show by contradiction that a path must exist. Assume that the process above cannot complete, i.e. there is some demand $\tilde{d}(i,j) > 0$ that cannot be placed in $\mathbf{d}_1$ or $\mathbf{d}_2$, and there is no path P as specified above. Then there is a set $X \subseteq V(G_B)$ of vertices reachable from i by edges with positive demand in $\mathbf{d}'$, all of which have saturated capacity $\mathbf{d}'(\ell_r) = c_B(\ell_r)$. Writing $X_1 = X \cap K_1$, $X_2 = X \cap K_2$, we get that $\mathbf{d}_1(X_1) = \sum_{\ell \in X_1} \mathbf{d}_1(\ell) = c_1(X_1)$ and $\mathbf{d}_2(X_2) = c_2(X_2)$. We can furthermore deduce that $\mathbf{d}_1(X_1) \le \mathbf{d}_1(X_2)$, since all of the demand in $\mathbf{d}_1$ incident on X_1 is represented as demand in an arc (X_1, X_2), which is thus also incident on X_2. Similarly, we know that $\mathbf{d}_2(X_2) \le \mathbf{d}_2(X_1)$. Putting these facts together, we conclude that:

$$c_1(X_1) = \mathbf{d}_1(X_1) \le \mathbf{d}_1(X_2) \le c_1(X_2) \quad \text{and} \quad c_2(X_2) = \mathbf{d}_2(X_2) \le \mathbf{d}_2(X_1) \le c_2(X_1)$$

Since $\mathcal{S}'_{c_1} = \mathcal{S}'_{c_2}$, it must be the case that $c_1(X_1) = c_1(X_2)$ and $c_2(X_2) = c_2(X_1)$, and thus from the inequalities above we can conclude that $\mathbf{d}_1(X_2) = c_1(X_2)$, and in particular, since $i_2 \in X_2$, it must be that $\mathbf{d}_1(i) = c_1(i)$. But this is a contradiction, as $\mathbf{d}(i) = \tilde{\mathbf{d}}(i) + \mathbf{d}_1(i) + \mathbf{d}_2(i) = \tilde{\mathbf{d}}(i) + c_1(i) + c_2(i) > c(i)$, and thus $\mathbf{d}$ would not be routable in G'.

We conclude that, as long as $\tilde{\mathbf{d}} \ne \mathbf{0}$, the process above finds a path and thus makes progress in each iteration. Therefore, $\mathbf{d}$ is routable in G by splitting it into demands $\mathbf{d}_1$ for v_1 and $\mathbf{d}_2$ for v_2 as described above. ◀

4 Sparsifiers for Small Vertex Cover and Integrity

The vertex cover number c of a graph G is the size of the smallest set X such that $G - X$ contains no edges. The vertex integrity d of a graph extends this by allowing $G - X$ to have small components, and it is the smallest number for which there is a set X such that $G - X$ has components of size at most $d - |X|$ (i.e. the size of a component plus X does not exceed

d). We show that if either of these parameters is bounded, the exponential complexity in the size of the sparsifier is limited to an *additive* term depending only on the parameter. The result of Section 3.1 corresponds to the case of $c = k$ (or $d = k + 1$).

Our formal results are the following:

▶ **Theorem 4.1.** *Let $G = (V, E, u)$ be a network with terminal set K of size k.*

If G has a vertex cover of size c, it has a cut sparsifier of size $k + 2^{c^2}$ and a flow sparsifier of size $k + 3^{c^3}$.

Proof. Let X be a vertex cover of size c. We will start by splitting the graph into a subgraph with all of the terminals and another containing only X and non-terminals. Let $G_K = G[K \cup X]$ be the first of these graphs, and $G_S = G[(V \setminus K) \cup X] - E(X)]$ be the second. Notice that the graphs are Steiner-disjoint for terminal set $K \cup X$, and thus we can use Lemma 2.3. Furthermore, if we compute a sparsifier H_S for G_S (with terminal set X) and take the sparsifier $H = G_K \uplus H_S$ for G, the size of H is $k + |V(H)|$. All that remains is to apply Theorem 3.1 to obtain a cut sparsifier H_S of size 2^{c^2} or Theorem 3.2 to obtain a flow sparsifier H_S of size 3^{c^3}, which completes the proof of the theorem. ◀

▶ **Theorem 4.2.** *Let $G = (V, E, u)$ be a network with terminal set K of size k.*

If G has a separator $X \subseteq V$ of size a and $|C| \leq b$ for every component C of $G - X$, then it has a sparsifier of size $kb + 4^{b(a+b)^2}$.

As a corollary, we get that small sparsifiers exist when vertex integrity is bounded.

▶ **Corollary 4.3.** *Let $G = (V, E, u)$ be a network with terminal set K of size k.*

If G has a separator $X \subseteq V$ such that $|X| + |C| \leq d$ for every component C of $G - X$, then it has a cut sparsifier of size $kd + 4^{d^3}$.

The rest of the section is dedicated to proving Theorem 4.2.

We use the same strategy of separating out the terminals, though in this case we need to show a new bound when $G - X$ has small connected components. We will use similar polyhedral techniques adjusted for the case of small components.

Let $X \subseteq V$ be a separator of size a such that $G - X$ has components of size at most b. Let $\mathcal{C} = \mathrm{cc}(G - X)$, and let $\mathcal{C}_K$ be the set of connected components containing any terminals. We define $G_K = G[X \cup \bigcup\{C \in \mathcal{C}_K\}]$ and $G_S = G[(V(G) \setminus V(G_K)) \cup X]$. The graphs are Steiner-disjoint for terminals $X \cup K$, and $|V(G_K) \setminus V(G_S)| \leq kb$, thus by Lemma 2.3, $H = G_K \uplus H_S$ is a sparsifier for G of size $kb + |V(H_S)|$, where H_S is a sparsifier for G_S.

All that is left is then to show that G_S with terminal set X has a sparsifier of size $4^{b(a+b)^2}$. Though on stars (components of size 1), there are only two possible cuts for each subset of terminals, here there are more possibilities for cuts, and thus we need multiple inequalities for each subset of terminals. Therefore, we show that there are at most $4^{b(a+b)^2}$ *basic components* of size at most b, such that any connected component can be written as the conic combination of these. We then replace each component by a conic combination of basic components, and contract all of the components of the new graph corresponding to the same basic component.

Basic components. Let C be a component with vertices $v_1, \ldots, v_b$, and let the terminals be ordered $K = \{t_1, \ldots, t_k\}$. To simplify the analysis, we consider that every component has size b by adding isolated vertices if needed. We represent the edge capacities for C as a vector $c \in \mathbb{R}_{\geq 0}^{C \times (C \cup X)}$ (where $c(v_i, v_j) = c(v_j, v_i)$ for all $v_i, v_j \in C$, $c(v_i, v_i) = 0$ for all $v_i \in C$).

We define $\mathcal{S}_c$ to be the collection of triples (A, B, B'), $A \subseteq K$, $B, B' \subseteq V(C)$, such that $c(A \cup B) \le c(A \cup B')$, i.e. $\mathcal{S}_c = \{(A, B, B') \subseteq K \times V(C)^2 : c(A \cup B) \le c(A \cup B')\}$, and the *cut cone* of c to be:

$$P_c := \Big\{x \in \mathbb{R}_{\ge 0}^{C\times(C\cup X)} : x(A \cup B) \le x(A \cup B') \ \forall (A, B, B') \in \mathcal{S}_c; \\ x(v_i, v_j) = x(v_j, v_i), x(v_i, v_i) = 0 \ \forall v_i, v_j \in V(C)\Big\}$$

We say that a vector x *agrees with* c if $c(A \cup B) \le c(A \cup B')$ for all $(A, B, B') \in \mathcal{S}_c$, and thus P_c is the polyhedron containing all of the capacity vectors that agree with c. It is a cone since it is defined by constraints of the form $\alpha^T x \le 0$.

We can now define the set Q of extreme rays similarly to the star case, as the set of all capacity vectors that satisfy $b(a+(b-1)/2)$ many linearly independent constraints. For each $q \in Q$, we can construct a corresponding *basic component* C_q with vertices $X \cup \{v_{q,1}, v_{q,2}, \dots, v_{q,b}\}$ and capacities defined according to q.

The size of Q is determined by the possible sets of inequalities that define each of its elements. Out of the 2^{a+2b}, we choose at most $b(a + b/2)$, which gives us an upper bound of $|Q| \le 2^{b(a+2b)(a+b)} \le 4^{b(a+b)^2}$.

The following lemma expresses components as conic combinations of basic components:

▶ **Lemma 4.4.** *Any capacity vector c for a component C can be written as the conic combination of at most $\ell = b(a + b/2)$ points in Q, all of which agree with c.*

Formally, there are $q_1, \dots, q_\ell \in Q$, $\lambda(q_1), \dots, \lambda(q_\ell) \ge 0$ such that :

$$\sum_{i=1}^{\ell} \lambda(q_i) \cdot q_i = c \qquad \textit{and} \qquad q_i(A \cup B) \le q_i(A \cup B') \qquad \forall i \in [\ell], (A, B, B') \in \mathcal{S}_c$$

Proof. The proof mimics that of Lemma 3.4, with some slight adjustments to consider the different definition of $\mathcal{S}_c$, P_c and Q.

Let P_c be the cut cone corresponding to c. Notice that P_c always includes equality constraints to ensure symmetry ($x(v_i, v_j) = x(v_j, v_i)$, $x(v_i, v_i) = 0$), and these $\binom{b}{2} + b$ constraints are independent. Thus, by Carathéodory's theorem c is a conic combination of at most $\ell = ab - \binom{b}{2} - b \le b(a + b/2)$ extreme rays of P_c (which are contained in Q). In other words, there are $q_1, \dots, q_k \in P_c \cap Q$ and $\lambda(q_1), \dots, \lambda(q_k) \ge 0$ such that $c = \sum_{i=1}^{k} \lambda(q_i) \cdot q_i$.

Furthermore, each q_i agrees with c on every cut since it is an extreme ray of P_c, and thus satisfies its inequalities. ◀

We obtain a sparsifier H for G as follows: given a separator X, $|X| \le a$ such that $G - X$ only has connected components of size at most b, we start by partitioning G into $G_K = G[X \cup \bigcup\{C \in \mathcal{C}_K\}]$ and $G_S = G[(V(G) \setminus V(G_K)) \cup X]$. We then compute Q and the coefficients λ_C for each component C of $G_S - X$, in time $2^{O(b(a+b)^2)}$. Then, we define the collection of graphs $\mathcal{C}_Q(G_S) = \{C'_q : q \in Q\}$, where $C'_q = C_q \cdot \sum_{C\in\mathcal{C}} \lambda_C(q)$ is C_q with the capacities scaled up by $\sum_{C\in\mathcal{C}} \lambda_C(q)$. The sparsifier is obtained by doing the Steiner-disjoint union of the graphs in $\mathcal{C}_Q(G_S)$ and G_K, that is, $H = G_K \uplus \biguplus_{C'_q\in\mathcal{C}_Q(G_S)} C'_q$.

The analysis follows similarly to Section 3.1, by applying Lemmas 4.4 and 4.5 to G_S.

▶ **Lemma 4.5 (*).** *Let G be a network with a separator $X \subseteq V(G)$ of size a and $|C| \le b$ for every component C of $G - X$.*

Then $H = G_K \uplus \biguplus_{C'_q\in\mathcal{C}_Q(G_S)} C'_q$ is a sparsifier for G of size at most $4^{b(a+b)^2}$.

5 Reduction Theorem for Bounded-Treewidth Graphs

In this section, we show that given a graph G with k terminals and a tree decomposition for G of width w, we can compute in linear time a sparsifier for G with size linear in k. In particular, by using as a black-box algorithm for computing a sparsifier with size $S(k)$ and quality $g(k)$, we obtain a sparsifier of size $O(k) \cdot S(2w)$ and quality $g(2w)$, which can be computed with $O(k)$ calls to the original sparsifier algorithm. The result is formalized in Theorem 5.1.

▶ **Theorem 5.1.** *Let $G = (V, E, u)$ be a network with terminal set K of size k. Let $S : \mathbb{N} \to \mathbb{N}$, $g \colon \mathbb{N} \to \mathbb{R}_{\geq 1}$ be functions such that every network (of treewidth w) has a quality-$g(k)$ cut (resp. flow) sparsifier of size $S(k)$.*

Then any network of treewidth at most w has a cut (resp. flow) sparsifier with quality $g(2w)$ and size $O(k) \cdot S(2w)$.

Furthermore, given a tree decomposition of width w, the sparsifier can be computed in time $O(nw)$ plus $O(k)$ calls to an algorithm computing sparsifiers on edge-disjoint subgraphs of G with at most $2w$ terminals.

As an immediate consequence of Theorem 5.1 and work on flow sparsifiers [9, 16, 30], we obtain the following corollary:

▶ **Corollary 5.2.** *Any network with treewidth at most w has a quality-$O(\log w)$ flow sparsifier of size $O(k \cdot w)$, which can be computed in polynomial time.*

Let $(T, \mathcal{B})$ be a tree decomposition for G of width w. We recall that we associate each edge with a single bag, and that we assume that no two identical bags exist.

Let $Y \subseteq V(T)$ be a subset of bags obtained as follows: first, for each terminal $t \in K$, add to Y a node $u \in V(T)$ containing t, i.e. $t \in B_u$; then add the lowest common ancestors of any two nodes Y to Y as well.

The algorithm first constructs the set Y, and then partitions T into a set of regions $\mathcal{R}(T, Y)$ as follows: consider the components of $T - Y$, and group them into regions according to their neighboring nodes. Finally, it returns the sparsifier $H = G[Y] \uplus \biguplus_{R \in \mathcal{R}(T,Y)} H_R$, where H_R is a sparsifier for the subgraph $G[R]$ with terminal set $G[R] \cap B(Y)$, computed using the black-box algorithm.

Since T has $O(n)$ nodes, constructing Y as well as the graphs $G[R]$ for every region can be done in time $O(nw)$, and computing the sparsifiers simply requires $|\mathcal{R}(T, Y)|$ calls to the given sparsifier algorithm. We remark that the calls to the sparsifier algorithm are run on subgraphs $G[R]$ for disjoint R, and thus induce edge-disjoint subgraphs of G.

We will now show that the size of Y and $\mathcal{R}(T, Y)$ is bounded, before showing that each $G[R]$ has a small sparsifier, and that all of these can be joined into H.

▶ **Lemma 5.3.** *There are at most $2k$ nodes in Y and $2|Y|$ regions in $\mathcal{R}(T, Y)$, with each region neighboring at most two nodes of Y.*

Proof. Let T' be the tree obtained from T by iteratively contracting every edge that does not connect to nodes in Y. T' has (at most) one node for every terminal in K, plus nodes for the lowest common ancestors. In particular, the nodes for the lowest common ancestors have at least 2 children, as they were added to Y because there are terminals in two of its children subtrees. As there are at most k nodes with at most 1 child, T' must have at most $2k - 1$ nodes.

We now show that every region neighbors at most two nodes of Y. Assume that there is a region R neighboring more than 2 nodes in Y. If there are two nodes u, v such that one is not the ancestor of the other, then their lowest common ancestor a is also in Y and in

R, since the u-v-path in T is contained in R. But then a is a cut in T which splits R into smaller regions, and thus R cannot be a region in $\mathcal{R}(T, Y)$. The remaining possibility is that the nodes neighboring R all are ancestors or descendants of one another, and thus are all contained in a root-leaf path in T. However, in that case, the middle points (not the highest or lowest neighbor of R) are all vertex cuts which would split R into smaller regions. We conclude that each region neighbors at most two nodes of Y, and that one is the parent of the other in T'.

As a consequence of the above, there can be at most $|Y|$ regions that have only a single node as neighbor, and $|Y|$ regions that have two neighbors in Y (a node u and its parent). ◀

We now show that we can apply a sparsifier to $G[R]$ to obtain a good sparsifier.

▶ **Lemma 5.4.** *Given $R \in \mathcal{R}(T, Y)$, its induced subgraph $G[R]$ with terminal set $G[R] \cap B(Y)$ has a sparsifier H_R of quality $g(2w)$ and size $S(2w)$.*

Proof. As G has treewidth w, any of its subgraphs has treewidth w as well. Thus, we only need to prove that $|G[R] \cap B(Y)| \le 2w$, as we assume that every graph of treewidth w has a quality-$g(k)$ sparsifier of size $S(k)$, with $k = |G[R] \cap B(Y)|$ for $G[R]$.

If R has only one neighbor $y \in V(T)$, then $|G[R] \cap B(Y)| = |G[R] \cap B_y| \le w + 1 \le 2w$, as the only neighboring bag to R is y and thus $G[R]$ and $G[T - R]$ only intersect in B_y.

Otherwise, R has two neighbors $y_1, y_2 \in V(T)$. Let u_1, u_2 be the nodes of R neighboring y_1, y_2, respectively. Notice that R must correspond to a single connected component of $T - Y$, as there is a single path connecting y_1 and y_2 in T, and thus y_1 and y_2 each have a single neighbor in R.

By the properties of the tree decomposition, any vertex v that is simultaneously in $G[R]$ and $G[T - R]$ must either be contained in B_{y_1} and B_{u_1}, or be contained in B_{y_2} and B_{u_2}, as the subtree induced by the bags containing v must connect R and $T - R$ and thus must contain the edge y_1u_1 or the edge y_2u_2. Since no two bags are the same, $|B_{y_1} \cap B_{u_1}| \le w$, $|B_{y_2} \cap B_{u_2}| \le w$, and thus $|G[R] \cap B(Y)| \le |B_{y_1} \cap B_{u_1}| + |B_{y_2} \cap B_{u_2}| \le 2w$. ◀

We complete the proof by taking the sparsifier $H = G[Y] \uplus \biguplus_{R \in \mathcal{R}(T,Y)} H_R$. H is a sparsifier for G because $G = G[Y] \uplus \biguplus_{R \in \mathcal{R}(T,Y)} G[R]$ and, by Lemma 5.4, each H_R is a sparsifier for $G[R]$, thus by (repeated application of) Lemma 2.3, H is a sparsifier for G with terminal set $B(Y) \supseteq K$. The size of H is at most $2k(w + 1) + 2kS(2w) = O(k) \cdot S(2w)$, by using Lemma 5.3 and it can be computed in running time $O(nw)$ plus $O(k)$ calls to edge-disjoint subgraphs of G.

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. 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 335–344. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.44.

2 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1 + \varepsilon)$-approximate flow sparsifiers. 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 279–293. SIAM, 2014. doi:10.1137/1.9781611973402.20.

3 Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024.

4 C. A. Barefoot, R. C. Entringer, and H. C. Swart. Vulnerability in graphs – A comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1, 1987.

5 András A. Benczúr and David R. Karger. Augmenting undirected edge connectivity in $\tilde{O}(n^2)$ time. *J. Algorithms*, 37(1):2–36, 2000. doi:10.1006/JAGM.2000.1093.

6 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.

7 Hans L. Bodlaender, Carla Groenland, and Michal Pilipczuk. Parameterized complexity of binary CSP: vertex cover, treedepth, and related parameters. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 27:1–27:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.27.

8 Gruia Călinescu, Howard J. Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM J. Comput.*, 34(2):358–372, 2004. doi:10.1137/S0097539701395978.

9 Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 265–274. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.32.

10 Shiva Chaudhuri, K. V. Subrahmanyam, Frank Wagner, and Christos D. Zaroliagis. Computing mimicking networks. *Algorithmica*, 26(1):31–49, 2000. doi:10.1007/S004539910003.

11 Yu Chen and Zihan Tan. On $(1 + \varepsilon)$-approximate flow sparsifiers. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1568–1605. SIAM, 2024. doi:10.1137/1.9781611977912.63.

12 Julia Chuzhoy. On vertex sparsifiers with Steiner nodes. 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 673–688. ACM, 2012. doi:10.1145/2213977.2214039.

13 Michele Conforti, Gerard Cornuejols, and Giacomo Zambelli. *Integer Programming.* Springer, 2014. doi:10.1007/978-3-319-11008-0.

14 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

15 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.

16 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014. doi:10.1137/130908440.

17 Jirí Fiala, Petr A. Golovach, and Jan Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. *Theor. Comput. Sci.*, 412(23):2513–2523, 2011. doi:10.1016/J.TCS.2010.10.043.

18 Fedor V. Fomin, Mathieu Liedloff, Pedro Montealegre, and Ioan Todinca. Algorithms parameterized by vertex cover and modular width, through potential maximal cliques. *Algorithmica*, 80(4):1146–1169, 2018. doi:10.1007/S00453-017-0297-1.

19 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

20 Tatsuya Gima and Yota Otachi. Extended MSO model checking via small vertex integrity. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.20.

21 R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. URL: http://www.jstor.org/stable/2098881.

22 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998. doi:10.1006/JCSS.1998.1592.

23 Arun Jambulapati, James R. Lee, Yang P. Liu, and Aaron Sidford. Sparsifying sums of norms. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1953–1962. IEEE, 2023. doi:10.1109/FOCS57990.2023.00119.

24 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014. doi:10.1016/J.IPL.2014.02.011.

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

26 Robert Krauthgamer and Ron Mosenzon. Exact flow sparsification requires unbounded size. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2354–2367. SIAM, 2023. doi:10.1137/1.9781611977554.CH91.

27 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1789–1799. SIAM, 2013. doi:10.1137/1.9781611973105.128.

28 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ISAAC.2021.34.

29 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. 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 47–56. ACM, 2010. doi:10.1145/1806689.1806698.

30 Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 255–264. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.31.

31 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 3–12. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.28.

32 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

33 Jelle J. Oostveen and Erik Jan van Leeuwen. Streaming deletion problems parameterized by vertex cover. *Theor. Comput. Sci.*, 979:114178, 2023. doi:10.1016/J.TCS.2023.114178.

34 Sridhar Rajagopalan and Vijay V. Vazirani. On the bidirected cut relaxation for the metric Steiner tree problem. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 742–751. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314909.

35 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90. ACM, 2004. doi:10.1145/1007352.1007372.

Query Maintenance Under Batch Changes with Small-Depth Circuits

Samir Datta
Chennai Mathematical Institute & UMI ReLaX, Chennai, India

Asif Khan
Chennai Mathematical Institute, India

Anish Mukherjee
University of Warwick, Coventry, UK

Felix Tschirbs
Ruhr University Bochum, Germany

Nils Vortmeier
Ruhr University Bochum, Germany

Thomas Zeume
Ruhr University Bochum, Germany

Abstract

Which dynamic queries can be maintained efficiently? For constant-size changes, it is known that constant-depth circuits or, equivalently, first-order updates suffice for maintaining many important queries, among them reachability, tree isomorphism, and the word problem for context-free languages. In other words, these queries are in the dynamic complexity class DynFO. We show that most of the existing results for constant-size changes can be recovered for batch changes of polylogarithmic size if one allows circuits of depth $\mathcal{O}(\log \log n)$ or, equivalently, first-order updates that are iterated $\mathcal{O}(\log \log n)$ times.

2012 ACM Subject Classification Theory of computation → Logic and databases; Theory of computation → Complexity theory and logic

Keywords and phrases Dynamic complexity theory, parallel computation, dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.46

Related Version *Full Version*: `https://arxiv.org/abs/2407.20031`

Funding *Samir Datta*: Partially funded by a grant from Infosys foundation.
Asif Khan: Partially funded by a grant from Infosys foundation.
Anish Mukherjee: Research was supported in part by the Centre for Discrete Mathematics and its Applications (DIMAP) and by EPSRC award EP/V01305X/1.
Felix Tschirbs: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.
Nils Vortmeier: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.
Thomas Zeume: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.

1 Introduction

Dynamic descriptive complexity [23, 10] is a framework for studying the amount of resources that are necessary to *maintain* the result of a query when the input changes slightly, possibly using additional auxiliary data (which needs to be maintained as well). Its main class DynFO contains all queries for which the update of the query result (and possibly of further useful

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 46; pp. 46:1–46:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

auxiliary data) can be expressed in first-order logic FO. Equivalently[1], the updates can be computed using (DLOGTIME) uniform circuits with constant-depth and polynomial size that consist of $\neg$- as well as $\wedge$- and $\vee$-gates with unbounded fan-in, that is, within uniform AC^0.

It is known that many important queries can be maintained in DynFO if only one bit of the input changes in every step. This includes reachability for acyclic graphs [11, 23], undirected graphs [23, 12, 17], and general directed graphs [6], tree isomorphism [14] and every problem definable in monadic second-order logic MSO for graphs of bounded treewidth [8], all under insertions and deletions of single edges. Also, membership in context-free languages can be maintained under changes of single positions of the input word [15].

Some of these results have been extended to changes beyond single-bit changes: reachability in undirected graphs is in DynFO if simultaneously $\text{polylog}(n) = (\log n)^{\mathcal{O}(1)}$ edges can be inserted or deleted [7], where n is the size of the graph; regular languages are in DynFO under changes of $\text{polylog}(n)$ positions at once [24]. Reachability in directed graphs can be maintained under insertions and deletions of $\mathcal{O}(\frac{\log n}{\log\log n})$ many edges [9].

Thus, only for few problems it is known that changes of polylogarithmic size (or: even non-constant size) can be handled in DynFO, or, equivalently, by AC^0-updates. Trivially, if a problem can be maintained in DynFO under single-bit changes it can also be maintained under $\text{polylog}(n)$ changes using AC-circuits of $\text{polylog}(n)$-depth. This is achieved by processing the changed bits “sequentially” by “stacking” $\text{polylog}(n)$ copies of the constant-depth circuit for processing single-bit changes.

The starting point for the present paper is the question which problems can be maintained by AC-circuits of less than $\text{polylog}(n)$ depth under $\text{polylog}(n)$-sized changes, in particular which of the problems known to be in DynFO under single-bit changes. The answer is short: for almost all of them circuits of depth $\mathcal{O}(\log\log n)$ suffice.

A first observation is that directed reachability under polylogarithmic changes can be maintained by AC-circuits of depth $\mathcal{O}(\log\log n)$. This can be derived by analyzing the proof from [9] (see Section 3). For this reason, we introduce the dynamic complexity class[2] DynFOLL of problems that can be maintained using circuits with polynomial size and depth $\mathcal{O}(\log\log n)$ or, equivalently, by first-order formulas that are iterated $\mathcal{O}(\log\log n)$ times. We investigate its power when changes affect $\text{polylog}(n)$ input bits and prove that almost all problems known to be maintainable in DynFO for constant-size changes fall into this class for changes of $\text{polylog}(n)$-size, see Table 1. One important problem left open is whether all MSO-definable queries for bounded treewidth graphs can be maintained in DynFOLL under $\text{polylog}(n)$ changes. We present an intermediate result and show that tree decompositions can be maintained within DynFOLL (see Section 5).

This power of depth-$\mathcal{O}(\log\log n)$ update circuits came as a surprise to us. Statically, circuits of this depth and polynomial size still cannot compute the parity of n bits due to Håstad's famous lower bound for parity: depth-$(d+1)$ AC-circuits with alternating $\wedge$- and $\vee$-layers require $2^{\Omega(n^{1/d})}$ gates for computing parity (see, e.g., [20, Theorem 12.3]). Dynamically, while such update circuits are powerful for changes of non-constant size, they seem to provide not much more power for single-bit changes. As an example, the parity-exists query from [26] is conjectured to not be in DynFO, and it also cannot easily be seen to be in DynFOLL.

[1] assuming that first-order formulas have access to numeric predicates $\leq, +, \times$

[2] The class could equally well be called (uniform) $\mathsf{DynAC}[\log\log n]$. We opted for the name DynFOLL as it extends DynFO and its static variant was introduced as FOLL [2].

Table 1 Overview of results for DynFO and DynFOLL. Entries indicate the size of changes that can be handled by DynFO and DynFOLL programs, respectively.

Dynamic query	DynFO		DynFOLL	
reachability				
general graphs	$\mathcal{O}(\frac{\log n}{\log\log n})$	[9]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 3)
undirected graphs	$(\log n)^{\mathcal{O}(1)}$	[7]	$(\log n)^{\mathcal{O}(\log\log n)}$	(Theorem 4)
acyclic graphs	$\mathcal{O}(\frac{\log n}{\log\log n})$	[9]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 3)
distances				
general graphs	open		open	
undirected graphs	$\mathcal{O}(1)$	[17]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 6)
acyclic graphs	$\mathcal{O}(1)$		$(\log n)^{\mathcal{O}(1)}$	(Theorem 6)
bounded tree width				
tree decomposition	open		$(\log n)^{\mathcal{O}(1)}$	(Theorem 16)
MSO properties	$\mathcal{O}(1)$	[8]	$\mathcal{O}(1)$	
other graph problems				
tree isomorphism	$\mathcal{O}(1)$	[14]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 12)
minimum spanning forest	$(\log n)^{\mathcal{O}(1)}$	(Theorem 5)	$(\log n)^{\mathcal{O}(\log\log n)}$	(Theorem 5)
maximal matching	$\mathcal{O}(1)$	[23]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 5)
$(\delta+1)$-colouring	$(\log n)^{\mathcal{O}(1)}$	(Theorem 5)	n^2	(static, [16])
word problem				
regular languages	$(\log n)^{\mathcal{O}(1)}$	[24]	$(\log n)^{\mathcal{O}(\log\log n)}$	(Theorem 4)
context-free languages	$\mathcal{O}(1)$	[15]	$(\log n)^{\mathcal{O}(1)}$	(Theorem 9)

The obtained bounds are almost optimal. For all mentioned problems, DynFO can handle changes of size at most polylog(n) and DynFOLL can handle changes of size at most $(\log n)^{\mathcal{O}(\log\log n)}$. This is an immediate consequence of Håstad's lower bound for parity and standard reductions from parity to these problems. For the queries that are known to be maintainable under polylog(n) changes in DynFO, we show that they can be maintained under $(\log n)^{\mathcal{O}(\log\log n)}$ changes in DynFOLL.

Our results rely on two main techniques for handling changes of polylogarithmic size:

- In the *small-structure technique* (see Section 3), it is exploited that on structures of polylogarithmic size, depth-$\mathcal{O}(\log\log n)$ circuits have the power of NC^2 circuits. Dynamic programs that use this technique first construct a substructure of polylogarithmic size depending on the changes and the current auxiliary data, then perform a NC^2-computation on this structure, and finally combine the result with the rest of the current auxiliary data to obtain the new auxiliary data. This technique is a slight generalization of previously used techniques for DynFO.
- In the *hierarchical technique* (see Section 4), it is exploited that auxiliary data used in dynamic programs is often "composable". Dynamic programs that use this technique first construct polynomially many structures depending on the current auxiliary data, each of them associated with one of the changes (in some cases, known dynamic programs for single changes can be exploited for this step). Then, in $\mathcal{O}(\log\log n)$ rounds, structures are combined hierarchically such that after ℓ rounds the program has computed polynomially many structures, each associated with 2^ℓ changes.

2 Preliminaries and setting

We introduce some notions of finite model theory, circuit complexity and the dynamic complexity framework.

Finite model theory & circuit complexity. A (relational) schema σ is a set of relation symbols and constant symbols. A relational structure $\mathcal{S}$ over a schema σ consists of a finite domain D, relations $R^{\mathcal{S}} \subseteq D^k$ for every k-ary relation symbol $R \in \sigma$, and interpretations $c^{\mathcal{S}} \in D$ of every constant symbol $c \in \sigma$. We assume in this work that every structure has a linear order $\leq$ on its domain. We can therefore identify D with the set $\{0, \ldots, n-1\}$.

First-order logic FO is defined in the usual way. Following [19], we allow first-order formulas to access the linear order on the structures and corresponding relations $+$ and $\times$ encoding addition and multiplication. We write $\mathsf{FO}(\leq, +, \times)$ to make this explicit. $\mathsf{FO}(\leq, +, \times)$ can express iterated addition and iterated multiplication for polylogarithmically many numbers that consist of polylog(n) bits, see [18, Theorem 5.1].

First-order logic with $\leq, +, \times$ is equivalent to (DLOGTIME) uniform AC^0, the class of problems decidable by uniform families of constant-depth circuits with polynomially many "not"-, "and"- and "or"-gates with unbounded fan-in. We write $\mathsf{AC}[f(n)]$ for the class that allows for polynomial-sized circuits of depth $\mathcal{O}(f(n))$, where n is the number of input bits. For polynomially bounded and first-order constructible functions f, the class $\mathsf{AC}[f(n)]$ is equal to $\mathsf{IND}[f(n)]$, the class of problems that can be expressed by inductively applying an $\mathsf{FO}(\leq, +, \times)$ formula $\mathcal{O}(f(n))$ times [19, Theorem 5.22]. So, we can think of an $\mathsf{AC}[f(n)]$ circuit as being a stack of $\mathcal{O}(f(n))$ copies of some AC^0 circuit. The class FOLL, see [2], is defined as $\mathsf{IND}[\log \log n] = \mathsf{AC}[\log \log n]$.

The circuit complexity classes uniform NC^i and SAC^i are defined via uniform circuits of polynomial size and depth $\mathcal{O}((\log n)^i)$; besides "not"-gates, NC circuits use "and"- and "or"-gates with fan-in 2, SAC circuits allow for "or"-gates with unbounded fan-in.

Dynamic complexity. The goal of a *dynamic program* Π is to maintain the result of a query applied to an input structure $\mathcal{I}$ that is subject to changes. In this paper, we consider changes of the form $\textsc{ins}_R(P)$, the insertion of a set P of tuples into the relation R of $\mathcal{I}$, and $\textsc{del}_R(P)$, the deletion of the set P from R. We usually restrict the size of the set P to be bounded by a function $s(n)$, where n is the size of the domain of $\mathcal{I}$. Most of the time, the bound is polylogarithmic in n, so $s(n) = \log(n)^c$ for some constant c. A pair (Q, Δ) of a query Q and a set Δ of (size-bounded) change operation $\textsc{ins}_R, \textsc{del}_R$ is called a *dynamic query.*

To maintain some dynamic query over σ-structures, for some schema σ, Π stores and updates a set $\mathcal{A}$ of auxiliary relations over some schema σ_{aux} and over the same domain as the input structure. For every auxiliary relation symbol $A \in \sigma_{\text{aux}}$ and every change operation δ, Π has an update program $\varphi_\delta^A(\bar{x})$, which can access input and auxiliary relations. Whenever an input structure $\mathcal{I}$ is changed by a change $\delta(P)$, resulting in the structure $\mathcal{I}'$, the new auxiliary relation $A^{\mathcal{A}'}$ in the updated auxiliary structure $\mathcal{A}'$ consists of all tuples $\bar{a}$ such that $\varphi_\delta^A(\bar{a})$ is satisfied in the structure $(\mathcal{I}', \mathcal{A})$.

We say that a dynamic program Π *maintains* a dynamic query (Q, Δ), if after applying a sequence α of changes over Δ to an initial structure $\mathcal{I}_0$ and applying the corresponding update programs to $(\mathcal{I}_0, \mathcal{A}_0)$, where $\mathcal{A}_0$ is an initial auxiliary structure, a dedicated auxiliary relation is always equal to the result of evaluating Q on the current input structure. Following Patnaik and Immerman [23], we demand that the initial input structure $\mathcal{I}_0$ is *empty*, so, has empty relations. The initial auxiliary structure is over the same domain as $\mathcal{I}_0$ and is defined from $\mathcal{I}_0$ by some first-order definable initialization.

The class DynFO is the class of all dynamic queries that are maintained by a dynamic program with $\mathsf{FO}(\leq, +, \times)$ formulas as update programs[3]. Equivalently, we can think of the update programs as being AC^0 circuits. The class $\mathsf{DynFO}[f(n)]$ allows for $\mathsf{AC}[f(n)]$ circuits as update programs. We often use the equivalence $\mathsf{AC}[f(n)] = \mathsf{IND}[f(n)]$ and think of update programs that apply an $\mathsf{FO}(\leq, +, \times)$ update formula $f(n)$ times. In this paper, we are particularly interested in the class $\mathsf{DynFOLL} = \mathsf{DynFO}[\log \log n]$.

3 The small-structure technique

The small-structure technique has been used for obtaining maintenance results for DynFO for non-constant size changes [7, 24]. The idea is simple: for changes of size m, (1) compute a structure with a domain of size roughly m, depending on the changes and the current auxiliary data, then (2) compute information about this structure (as $m \ll n$, this computation can be more powerful than AC^0), and (3) combine the result with the current auxiliary data to obtain the new auxiliary data.

For DynFO and changes of polylogarithmic size, one can use SAC^1-computations in step (2), as formalized in the next lemma.

▶ **Lemma 1** ([24, Corollary 3]). *Let Q be a k-ary query on σ-structures, for some $k \in \mathbb{N}$. If Q is uniform SAC^1-computable, then there is an $\mathsf{FO}(\leq, +, \times)$ formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size* $\text{polylog}(n)$ *and any k-tuple $\bar{a} \in C^k$ it holds that: $\bar{a} \in Q(\mathcal{S}[C])$ if and only if $(\mathcal{S}, C) \models \varphi(\bar{a})$. Here, $\mathcal{S}[C]$ denotes the substructure of $\mathcal{S}$ induced by C.*

For DynFOLL, this generalizes in two directions: (a) for structures of size $\text{polylog}(n)$ one can use NC^2-computations, (b) for structures of size $(\log n)^{\mathcal{O}(\log \log n)}$ one can use SAC^1-computations. This is captured by the following lemma.

▶ **Lemma 2.** *Let Q be a k-ary query on σ-structures, for some $k \in \mathbb{N}$.*

(a) *If Q is uniform NC^2-computable, then there is an FOLL formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size* $\text{polylog}(n)$ *and any k-tuple $\bar{a} \in C^k$ it holds that: $\bar{a} \in Q(\mathcal{S}[C])$ if and only if $(\mathcal{S}, C) \models \varphi(\bar{a})$.*

(b) *If Q is uniform SAC^1-computable, then there is an FOLL formula φ over schema $\sigma \cup \{C\}$ such that for any σ-structure $\mathcal{S}$ with n elements, any subset C of its domain of size $(\log n)^{\mathcal{O}(\log \log n)}$ and any k-tuple $\bar{a} \in C^k$ it holds that: $\bar{a} \in Q(\mathcal{S}[C])$ if and only if $(\mathcal{S}, C) \models \varphi(\bar{a})$.*

Proof.

(a) Let C have size m, which is polylogarithmically bounded in n. The NC^2-circuit for Q has polynomial size in m and depth $\mathcal{O}((\log m)^2)$, so its size is polylogarithmic in n and the depth is $\mathcal{O}((\log \log n)^2)$. It is well-known[4] that for every NC-circuit of depth $f(n)$ there is an equivalent AC-circuit of depth $\mathcal{O}(\frac{f(n)}{\log \log n})$ and size polynomial in the original circuit, so we can obtain an AC-circuit for answering Q on C with depth $\mathcal{O}(\log \log n)$.

[3] Other papers write DynFO for the class that uses FO update formulas without a priori access to the arithmetic relations $\leq, +, \times$ and $\mathsf{DynFO}(\leq, +, \times)$ for the class that uses $\mathsf{FO}(\leq, +, \times)$ update formulas. If changes only affect single tuples, there is no difference for most interesting queries, see [6, Proposition 7]. For changes that affect sets of tuples of non-constant size, all DynFO maintainability results use $\mathsf{FO}(\leq, +, \times)$ update formulas, as FO update formulas without arithmetic are not strong enough to maintain interesting queries. We therefore just write DynFO and omit the suffix $(\leq, +, \times)$ to avoid visual clutter.

[4] Divide the circuit into layers of depth $\log \log n$. Each layer depends only on $\log n$ gates of the previous layer, as each gate has fan-in at most 2, and can be replaced by a constant-depth circuit for the CNF of the layer, which has polynomial size.

(b) The proof of [1, Lemma 8.1] can easily be extended towards the following statement: if a language L is decided by a non-deterministic Turing machine with polynomial time bound m^c and polylogarithmic space bound $(\log m)^d$ then for every positive, non-decreasing and first-order constructible function $t(n)$ there is a uniform AC circuit family for L with depth $\mathcal{O}(t(n))$ and size $2^{\mathcal{O}(m^{\frac{c}{t(n)}}(\log m)^d)}$. For $m = \log n^{e \log\log n}$ and $t(n) = 2ce \log\log n$, the size is exponential in $\sqrt{\log n}\,(\log\log n)^{\mathcal{O}(1)}$, and therefore, as this function grows slower than $\log n$, polynomial in n. The statement follows as all SAC^1 languages can be decided by a non-deterministic Turing machine with polynomial time bound and space bounded by $(\log n)^2$, see [4]. ◀

A straightforward application of the technique to dynamic programs from the literature yields the following DynFOLL-programs. For directed reachability, adapting the DynFO-program for $\mathcal{O}(\frac{\log n}{\log\log n})$ changes from [9] yields (with more proof details in the full version):

▶ **Theorem 3.** *Reachability in directed graphs is in* DynFOLL *under insertions and deletions of* $\text{polylog}(n)$ *edges.*

For undirected reachability and regular languages, replacing Lemma 1 by Lemma 2(b) in the DynFO maintainability proofs for $\text{polylog}(n)$ changes from [7, 24] directly yields:

▶ **Theorem 4.**

(a) *Reachability in undirected graphs is in* DynFOLL *under insertions and deletions of* $(\log n)^{\mathcal{O}(\log\log n)}$ *edges.*

(b) *Membership in regular languages is in* DynFOLL *under symbol changes at* $(\log n)^{\mathcal{O}(\log\log n)}$ *positions.*

The small-structure technique has further applications beyond graph reachability and regular languages. We mention a few here. The proofs are deferred to the full version.

▶ **Theorem 5.**

(a) *A minimum spanning forest for weighted graphs can be maintained*

  (i) *in* DynFO *under changes of* $\text{polylog}(n)$ *edges, and*

  (ii) *in* DynFOLL *under changes of* $(\log n)^{\mathcal{O}(\log\log n)}$ *edges.*

(b) *A maximal matching can be maintained in* DynFOLL *under changes of* $\text{polylog}(n)$ *edges.*

(c) *For graphs with maximum degree bounded by a constant* δ*, a proper* $(\delta+1)$*-colouring can be maintained in* DynFO *under changes of* $\text{polylog}(n)$ *edges.*

4 The hierarchical technique

In this section we describe and use a simple, yet powerful hierarchical technique for handling polylogarithmic changes in DynFOLL. After changing $m \overset{\text{def}}{=} (\log n)^c$ many tuples, auxiliary data $\mathcal{R}^1, \ldots, \mathcal{R}^k$ is built in $k \overset{\text{def}}{=} d \log\log n$ rounds, for suitable d. The auxiliary data $\mathcal{R}^{\ell-1}$ after round $\ell - 1$ encodes information for certain subsets of the changes of size $2^{\ell-1}$. This information is then combined, via first-order formulas, to information on 2^ℓ changes in round ℓ. The challenge for each concrete dynamic query is to find suitable auxiliary data which is defined depending on a current instance as well as on subsets of changes, and can be combined via first-order formulas to yield auxiliary data for larger subsets of changes.

We apply this approach to maintaining distances in acyclic and undirected graphs, context-free language membership, and tree-isomorphism under polylogarithmic changes. In these applications of the hierarchical technique, information is combined along paths, binary trees, and arbitrary trees, respectively.

4.1 Undirected and acyclic reachability and distances

The articles that introduced the class DynFO showed that reachability for undirected and for acyclic graphs is in DynFO under single-edge changes [10, 23]. For these classes of graphs, also distances, that is, the number of edges in a shortest path between two reachable nodes, can be maintained. For undirected graphs, this was proven in [17], for acyclic graphs it is a straightforward extension of the proof for reachability from [10].

While reachability for undirected graphs is in DynFO under polylogarithmically many edge changes [7], we only know the general $\mathcal{O}(\frac{\log n}{\log \log n})$ bound for acyclic graphs [9]. It is unknown whether distances can be maintained in DynFO under changes of non-constant size, both for undirected and for directed, acyclic graphs.

▶ **Theorem 6.** *Distances can be maintained in* DynFOLL *under insertions and deletions of* $\text{polylog}(n)$ *edges for*

(a) *undirected graphs,*

(b) *acyclic directed graphs.*

To maintain distances, a dynamic programs can use a relation of the form $\text{DIST}(u, v, d)$ with the meaning "the shortest path from u to v has length d". The proof of Theorem 6 is then a direct application of the hierarchical technique on paths. After inserting polylogarithmically many edges, distance information for two path fragments can be iteratively combined to distance information for paths fragments that involve more changed edges. Thus $\text{polylog}\, n$ path fragments (coming from so many connected components before the insertion) can be combined in $\log \log n$ many iterations.

To handle edge deletions, we observe that some distance information is still guaranteed to be valid after the deletion: the shortest path from u to v surely has still length d after the deletion of some edge e if there was no path of length d from u to v that used e. These "safe" distances can be identified using the DIST relation. We show that after deleting polylogarithmically many edges, shortest paths can be constructed from polylogarithmically many "safe" shortest paths of the original graph. We make this formal now.

▶ **Lemma 7.** *Let $G = (V, E)$ be an undirected or acyclic graph, $e \in E$ an edge and $u, v \in V$ nodes such that there is a path from u to v in $G' = (V, E - e)$. For every shortest path $u = w_0, w_1, \dots, w_{d-1}, w_d = v$ from u to v in G' there is an edge (w_i, w_{i+1}) such that no shortest path from u to w_i and no shortest path from w_{i+1} to v in the original graph G uses e.*

Proof. For undirected graphs, this was proven in [22, Lemma 3.5c]. We give the similar proof for acyclic graphs. If no node w_{i+1} on a shortest path $u = w_0, w_1, \dots, w_{d-1}, w_d = v$ from u to v in G' exists such that some shortest path from u to w_{i+1} in G uses the edge e, the edge (w_{d-1}, v) satisfies the lemma statement. Otherwise, let w_{i+1} be the first such node on the path. It holds $i + 1 \geq 1$, as the shortest path from u to u trivially does not use e. So, no shortest path from u to w_i in G uses e. There is no shortest path from w_{i+1} to v in G that uses e: otherwise, there would be a path from e to w_{i+1} and a path from w_{i+1} to e in G, contradicting the assumption that G is acyclic. ◀

▶ **Corollary 8.** *Let $G = (V, E)$ be an undirected or acyclic graph and $\Delta E \subseteq E$ with $|\Delta E| = m$. For all nodes u and v such that v is reachable from u in $G' \overset{\text{def}}{=} (V, E \setminus \Delta E)$ there is a shortest path in G' from u to v that is composed of at most m edges and $m + 1$ shortest paths of G, each from some node u_i to some node v_i for $i \leq m + 1$, such that no shortest path from u_i to v_i in G uses an edge from ΔE.*

Proof idea. Via induction over m. For $m = 1$, this follows from Lemma 7. For $m > 1$ this is immediate from the induction hypothesis. ◀

We can now prove that distances in undirected and in acyclic graphs can be maintained in DynFOLL under changes of polylogarithmic size.

Proof of Theorem 6. We construct a DynFOLL program that maintains the auxiliary relation $\textsc{dist}(u, v, d)$ with the meaning "the shortest path from u to v has length d".

Suppose $m \stackrel{\text{def}}{=} (\log n)^c$ edges ΔE are changed in $G = (V, E)$ yielding the graph $G' = (V, E')$. W.l.o.g. all edges in ΔE are either inserted or deleted. In both cases, the program executes a first-order initialization, yielding auxiliary relations $\textsc{dist}^0(u, v, d)$, and afterwards executes a first-order procedure for $k \stackrel{\text{def}}{=} c \log\log n$ rounds, yielding auxiliary relations $\textsc{dist}^1, \ldots, \textsc{dist}^k$. The superscripts on the relations are for convenience, they are all subsequently stored in $\textsc{dist}$.

For insertions, we use the standard inductive definition of reachability and distances. Set $\textsc{dist}^0(u, v, d) \stackrel{\text{def}}{=} \textsc{dist}(u, v, d)$, where $\textsc{dist}(u, v, d)$ is the distance information of the unchanged graph G. Then, for k rounds, the distance information is combined with the new edges ΔE, doubling the amount of used edges from ΔE in each round. Thus $\textsc{dist}^\ell(u, v, d)$ is computed from $\textsc{dist}^{\ell-1}$ by including $\textsc{dist}^{\ell-1}$ and all tuples which satisfy the formula:

$$\varphi_{\text{INS}} \stackrel{\text{def}}{=} \exists z_1 \exists z_2 \exists d_1 \exists d_2 (\Delta E(z_1, z_2) \wedge d_1 + d_2 + 1 = d \wedge \textsc{dist}^{\ell-1}(u, z_1, d_1) \wedge \textsc{dist}^{\ell-1}(z_2, v, d_2))$$

For deletions, the program starts from shortest paths $u, \ldots, v$ in G such that no shortest path from u to v uses edges from ΔE and then combines them for k rounds, which yields the correct distance information for G' according to Corollary 8. Thus, the first-order initialization yields $\textsc{dist}^0(u, v, d)$ via

$$\textsc{dist}(u, v, d) \wedge \neg \exists z_1 \exists z_2 \exists d_1 \exists d_2 (d = d_1 + d_2 + 1 \wedge \Delta E(z_1, z_2) \wedge \textsc{dist}(u, z_1, d_1) \wedge \textsc{dist}(z_2, v, d_2))$$

Then $\textsc{dist}^\ell(u, v, d)$ is computed from $\textsc{dist}^{\ell-1}(u, v, d)$ via a formula similar to φ_{INS}, using E instead of ΔE. ◀

4.2 Context-free language membership

Membership problems for formal languages have been studied in dynamic complexity starting with the work of Gelade, Marquardt, and Schwentick [15]. It is known that context-free languages can be maintained in DynFO under single symbol changes [15] and that regular languages can even be maintained under polylog changes [25, 24].

It is an open problem whether membership in a context-free language can be maintained in DynFO for changes of non-constant size. We show that this problem is in DynFOLL under changes of polylogarithmic size.

▶ **Theorem 9.** *Every context-free language can be maintained in* DynFOLL *under changes of size* polylog n.

Suppose $G = (V, \Sigma, S, \Gamma)$ is a grammar in Chomsky normal form with $L \stackrel{\text{def}}{=} L(G)$. For single changes, 4-ary auxiliary relations $R_{X \to Y}$ are used for all $X, Y \in V$ [15], with the intention that $(i_1, j_1, j_2, i_2) \in R_{X \to Y}$ iff $X \Rightarrow^* w[i_1, j_1) Y w(j_2, i_2]$, where $w \stackrel{\text{def}}{=} w_1 \ldots w_n$ is the current string. Let us call $I = (i_1, j_1, j_2, i_2)$ a *gapped interval*. For a gapped interval I and a set P of changed positions, denote by $\#(I, P)$ the number of changed positions $p \in P$ with $p \in [i_1, j_1) \cup (j_2, i_2]$.

The idea for the DynFOLL program for handling polylog changes is simple and builds on top of the program for single changes. It uses the same auxiliary relations and, after changing a set P of positions, it collects gapped intervals I into the relations $R_{X\to Y}$ for increasing $\#(I,P)$ in at most $\mathcal{O}(\log\log n)$ rounds. Initially, gapped intervals with $\#(I,P)\leq 1$ are collected using the first-order update formulas for single changes. Afterwards, in each round, gapped intervals I with larger $\#(I,P)$ are identified by splitting I into two gapped intervals I_1 and I_2 with $\#(I,P)=\#(I_1,P)+\#(I_2,P)$ such that I can be constructed from I_1 and I_2 with a first-order formula.

To ensure that $\mathcal{O}(\log\log n)$ many rounds suffice, we need that the intervals I_1 and I_2 can always be chosen such that $\#(I_1,P)$ and $\#(I_2,P)$ are of similar size. This will be achieved via the following simple lemma, which will be applied to parse trees. For a binary tree $T=(V,E)$ with red coloured nodes $R\subseteq V$, denote by $\#(T,R)$ the number of red nodes of T. For a tree T and a node v, let T_v be the subtree of T rooted at v.

▶ **Lemma 10.** *For all rooted binary trees $T=(V,E,r)$ with red coloured nodes $R\subseteq V$, there is a node $v\in V$ such that:*

- $\#(T_v,R)\leq\frac{2}{3}\cdot\#(T,R)$ *and*
- $\#(T\setminus T_v,R)\leq\frac{2}{3}\cdot\#(T,R)$

Proof idea. Walk down the tree starting from its root by always choosing the child whose subtree contains more red coloured nodes. Stop as soon as the conditions are satisfied. ◀

We now provide the detailed proof of Theorem 9.

Proof (of Theorem 9). We construct a DynFOLL program that maintains the auxiliary relations $R_{X\to Y}$ for all $X,Y\in V$. Suppose $m\overset{\text{def}}{=}(\log n)^c$ positions P are changed. The program executes a first-order initialization, yielding auxiliary relations $R^0_{X\to Y}$, and afterwards executes a first-order procedure for $k\overset{\text{def}}{=}d\log\log n$ rounds, for $d\in\mathbb{N}$ chosen such that $(\frac{3}{2})^k>m$, yielding auxiliary relations $R^1_{X\to Y},\dots,R^k_{X\to Y}$. The superscripts on the relations are for convenience, they are all subsequently stored in $R_{X\to Y}$.

For initialization, the DynFOLL program includes gapped intervals (i_1,j_1,j_2,i_2) into the relations $R^0_{X\to Y}$ for which

- no position in $[i_1,j_1)\cup(j_2,i_2]$ has changed and (i_1,j_1,j_2,i_2) was previously in $R_{X\to Y}$, or
- exactly one position in $[i_1,j_1)\cup(j_2,i_2]$ has changed and the dynamic program for single changes from [15] includes the tuple (i_1,j_1,j_2,i_2) into $R_{X\to Y}$.

Afterwards, for k rounds, the DynFOLL program applies the following first-order definable procedure to its auxiliary relations. A gapped interval $I=(i_1,j_1,j_2,i_2)$ is included into $R^\ell_{X\to Y}$ in round ℓ if it was included in $R^{\ell-1}_{X\to Y}$ or one of the following conditions hold (see Figure 1 for an illustration):

(a) There are gapped intervals $I_1=(i_1,u_1,u_2,j_2)$ and $I_2=(u_1,j_1,j_2,u_2)$ and a non-terminal $Z\in V$ such that $I_1\in R^{\ell-1}_{X\to Z}$ and $I_2\in R^{\ell-1}_{Z\to Y}$. This can be phrased as first-order formula as follows:

$$\varphi_a\overset{\text{def}}{=}\exists u_1,u_2\Bigg[(i_1\leq u_1\leq j_1\leq j_2\leq u_2\leq i_2)\wedge\bigvee_{Z\in V}\Big(R_{X\to Z}(i_1,u_1,u_2,i_2)\wedge R_{Z\to Y}(u_1,j_1,j_2,u_2)\Big)\Bigg]$$

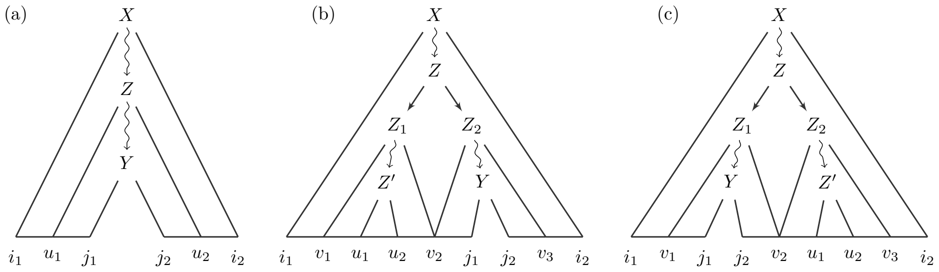

(a)
X
Z
Y
i_1 u_1 j_1 j_2 u_2 i_2
(b)
X
Z
Z_1
Z_2
Z'
Y
i_1 v_1 u_1 u_2 v_2 j_1 j_2 v_3 i_2
(c)
X
Z
Z_1
Z_2
Y
Z'
i_1 v_1 j_1 j_2 v_2 u_1 u_2 v_3 i_2

Figure 1 Illustration of when a gapped interval (i_1, j_1, j_2, i_2) is added to $R_{X\to Y}$ in the proof of Theorem 9.

(b) There are gapped intervals $I_1 = (i_1, v_1, v_3, i_2)$, $I_2 = (v_1, u_1, u_2, v_2)$, and $I_3 = (v_2, j_1, j_2, v_3)$ and non-terminals $Z, Z_1, Z_2, Z' \in V$ such that $Z \to Z_1Z_2 \in \Gamma$ and $I_1 \in R^{\ell-1}_{X\to Z}$, $I_2 \in R^{\ell-1}_{Z_1\to Z'}$, $I_3 \in R^{\ell-1}_{Z_2\to Y}$ and $w[u_1, u_2]$ can be derived from Z'. This can be phrased as first-order formula as follows:

$$\varphi_b \overset{\text{def}}{=} \exists u_1, u_2, v_1, v_2, v_3 \Bigg[(i_1 \leq v_1 \leq u_1 \leq u_2 \leq v_2 \leq j_1 \leq j_2 \leq v_3 \leq i_2) \land \bigvee_{\substack{Z,Z_1,Z_2,Z'\in V\\ Z\to Z_1Z_2\in\Gamma}} \Big(R_{X\to Z}(i_1, v_1, v_3, i_2) \land R_{Z_1\to Z'}(v_1, u_1, u_2, v_2) \land R_{Z'}(u_1, u_2) \land R_{Z_2\to Y}(v_2, j_1, j_2, v_3) \Big) \Bigg]$$

Here, $R_{Z'}(u_1, u_2)$ is an abbreviation for the formula stating that $w[u_1, u_2]$ can be derived from Z', i.e. $R_{Z'}(u_1, u_2) \overset{\text{def}}{=} \exists v \bigvee_{W\to\sigma\in\Gamma} (R_{Z'\to W}(u_1, v, v, u_2) \land \sigma(v))$.

(c) Symmetrical to (b), with gapped intervals $I_1 = (i_1, v_1, v_3, i_2)$, $I_2 = (v_1, j_1, j_2, v_2)$, and $I_3 = (v_2, u_1, u_2, v_3)$ and non-terminals $Z, Z_1, Z_2, Z' \in V$ such that $Z \to Z_1Z_2 \in \Gamma$ and $I_1 \in R^{\ell-1}_{X\to Z}$, $I_2 \in R^{\ell-1}_{Z_1\to Y}$, $I_3 \in R^{\ell-1}_{Z_2\to Z'}$ and $w[u_1, u_2]$ can be derived from Z'.

Note that $\#(I, P) = \#(I_1, P) + \#(I_2, P)$ in case (a) and $\#(I, P) = \#(I_1, P) + \#(I_2, P) + \#(I_3, P)$ in cases (b) and (c). Using Lemma 10, the intervals I_j can be chosen such that $\#(I_j, P) \leq \frac{2}{3} \cdot \#(I, P)$ and thus k rounds suffice. ◀

It is known that for single tuple changes one can maintain for edge-labeled, acyclic graphs whether there is a path between two nodes with a label sequence from a fixed context-free language [21]. The techniques we have seen can be used to also lift this result to changes of polylogarithmic size.

▶ **Proposition 11.** *Context-free path queries can be maintained under changes of polylogarithmic size in* DynFOLL *on acyclic graphs.*

4.3 Tree isomorphism

The dynamic tree isomorphism problem – given a forest $F = (V, E)$ and two nodes $x, x^* \in V$, are the subtrees rooted at x and x^* isomorphic? – has been shown to be maintainable in DynFO under single edge insertions and deletions by Etessami [14].

It is not known whether tree isomorphism can be maintained in DynFO under changes of size $\omega(1)$. We show that it can be maintained in DynFOLL under changes of size $\text{polylog}\, n$:

▶ **Theorem 12.** *Tree isomorphism can be maintained in* DynFOLL *under insertion and deletion of* polylog n *edges.*

Intuitively, we want to use Etessami's dynamic program as the base case for a DynFOLL-program: (1) compute isomorphism information for pairs of subtrees in which only one change happened, then (2) combine this information in $\log\log n$ many rounds. Denote by $\text{subtree}_x(r)$ the subtree rooted at r, in the tree rooted at x, within the forest F. The main ingredient in Etessami's program is a 4-ary auxiliary relation T-ISO for storing tuples (x, r, x^*, r^*) such that $\text{subtree}_x(r)$ and $\text{subtree}_{x^*}(r^*)$ are isomorphic and disjoint in F. It turns out that this information is not "composable" enough for step (2).

We therefore slightly extend the maintained auxiliary information. A *(rooted) context* $C = (V, E, r, h)$ is a tree (V, E) with root $r \in V$ and one distinguished leaf $h \in V$, called the *hole*. Two contexts $C = (V, E, r, h)$, $C^* = (V^*, E^*, r^*, h^*)$ are isomorphic if there is a root- and hole-preserving isomorphism between them, i.e. an isomorphism that maps r to r^* and h to h^*. For a forest F and nodes x, r, h occurring in this order on some path, the context $C(x, r, h)$ is defined as the context we obtain by taking $\text{subtree}_x(r)$, removing all children of h, and taking r as root and h as hole. Our dynamic program uses

- a 6-ary auxiliary relation C-ISO for storing tuples $(C, C^*) \overset{\text{def}}{=} (x, r, h, x^*, r^*, h^*)$ such that the contexts $C \overset{\text{def}}{=} C(x, r, h)$ and by $C^* \overset{\text{def}}{=} C(x^*, r^*, h^*)$ are disjoint and isomorphic,
- a ternary auxiliary relation DIST for storing tuples (x, y, d) such that the distance between nodes x and y is d, and
- a 4-ary auxiliary relation #ISO-SIBLINGS for storing tuples (x, r, y, m) such that y has m isomorphic siblings within $\text{subtree}_x(r)$.

The latter two relations have also been used by Etessami. From distances, a relation PATH(x, y, z) with the meaning "y is on the unique path between x and z" is FO-definable on forests, see [14]. The relation T-ISO(x, r, x^*, r^*) can be FO-defined from C-ISO.

We will now implement the steps (1) and (2) with these adapted auxiliary relations. Suppose a forest $F \overset{\text{def}}{=} (V, E)$ is changed into the forest $F' \overset{\text{def}}{=} (V, E')$ by changing a set ΔE of edges. A node $v \in V$ is *affected* by the change, if v is adjacent to some edge in ΔE. The DynFOLL program iteratively collects isomorphic contexts C and C^* of F' with more and more affected nodes. Denote by $\#(C, C^*, \Delta E)$ the number of nodes in contexts C and C^*, excluding hole nodes, affected by change ΔE.

The following lemma states that C-ISO can be updated for pairs of contexts with at most one affected node each. Its proof is very similar to Etessami's proof.

▶ **Lemma 13.** *Given* C-ISO, DIST, *and* #ISO-SIBLINGS *and a set of changes* ΔE, *the set of pairs* (C, C^*) *of contexts such that* C, C^* *are disjoint and isomorphic and such that both* C *and* C^* *contain at most one node affected by* ΔE *is* FO-*definable.*

The dynamic program will update the auxiliary relation C-ISO for contexts with at most one affected node per context using Lemma 13. Isomorphic pairs (C, C^*) of contexts with larger $\#(C, C^*, \Delta E)$ are identified by splitting both C and C^* into smaller contexts.

The splitting is done such that the smaller contexts have fewer than $\frac{2}{3} \cdot \#(C, C^*, \Delta E)$ affected nodes. To this end, we will use the following simple variation of Lemma 10. For a tree $T = (V, E)$ and a function $p : V \to \{0, 1, 2\}$ which assigns each a node number of pebbles, let $\#(T, p)$ be the total number of pebbles assigned to nodes in T.

▶ **Lemma 14.** *Let T be a tree of unbounded degree and p such that either (i) $\#(T, p) > 2$, or (ii) $\#(T, p) = 2$ and $p(v) \leq 1$ for all $v \in V$. Then there is a node v such that either:*

(1) $\#(T \setminus T_v, p) \leq \frac{2}{3} \cdot \#(T, p)$ *and* $\#(T_v, p) \leq \frac{2}{3} \cdot \#(T, p)$*, or*

(2) $\#(T \setminus T_v, p) \leq \frac{1}{3} \cdot \#(T, p)$ *and* $\#(T_u, p) \leq \frac{1}{3} \cdot \#(T, p)$ *for any child u of v.*

Proof idea. Use the same approach as for Lemma 10. If no node v of type (1) is found, a node of type (2) must exist. ◀

We now prove that tree isomorphism can be maintained in DynFOLL under changes of polylogarithmic size.

Proof (of Theorem 12). We construct a DynFOLL program that maintains the auxiliary relations C-ISO, DIST, and #ISO-SIBLINGS. Suppose $m \overset{\text{def}}{=} (\log n)^c$ edges ΔE are changed. As a preprocessing step, the auxiliary relation DIST is updated in depth $\mathcal{O}(\log\log n)$ via Theorem 6. Then, C-ISO is updated by first executing a first-order initialization for computing an initial version C-ISO^0. Afterwards a first-order procedure is executed for $k \overset{\text{def}}{=} d \log\log n$ rounds, for $d \in \mathbb{N}$ chosen such that $(\frac{3}{2})^k > m$, yielding auxiliary relations $\{\text{C-ISO}^\ell\}_{\ell \le k}$ and $\{\#\text{ISO-SIBLINGS}^\ell\}_{\ell \le k}$. The superscripts on the relations are for convenience, they are all subsequently stored in C-ISO, DIST, and #ISO-SIBLINGS.

The goal is that after the ℓth round

- C-ISO^ℓ contains all pairs C, C^* with $\#(C, C^*, \Delta E) \le (\frac{3}{2})^\ell$ which are isomorphic and disjoint, and
- $\#\text{ISO-SIBLINGS}^\ell$ contains the number of isomorphic siblings identified so far (i.e., with respect to C-ISO^ℓ).

Round ℓ first computes C-ISO^ℓ with a first-order procedure, and afterwards computes $\#\text{ISO-SIBLINGS}^\ell$. For initialization, the DynFOLL program first computes C-ISO^0, using Lemma 13, and $\#\text{ISO-SIBLINGS}^0$. Afterwards, for k rounds, the DynFOLL program combines known pairs of isomorphic contexts into pairs with more affected nodes and adapts C-ISO and #ISO-SIBLINGS accordingly.

Computing C-ISO$^\ell$. In the ℓth round, the program tests whether contexts $C \overset{\text{def}}{=} C(x, r, h)$ and $C^* \overset{\text{def}}{=} C(x^*, r^*, h^*)$ with $\#(C, C^*, \Delta E) \le (\frac{3}{2})^\ell$ are isomorphic by splitting both C and C^* into contexts with fewer affected nodes. The splitting is done by selecting suitable nodes $z \in C$ and $z^* \in C^*$, and splitting the context depending on these nodes.

We first provide some intuition of how z and z^* are intended to be chosen. Suppose C and C^* are isomorphic via isomorphism π. With the goal of applying Lemma 14, let p be the function that assigns to each non-hole node v of C a number of pebbles from $\{0, 1, 2\}$ indicating how many of the two nodes v and $\pi(v)$ have been affected by the change ΔE. Note that if (C, p) does not fulfill the precondition of Lemma 14, then (C, C^*) must have already been included in C-ISO^0 during the initialization. Therefore, assume the precondition holds for (C, p). Let C_z denote the subcontext of C rooted at z. Now, applying Lemma 14 to (C, p) yields a node z such that one of the following cases holds:

(1) $\#(C \setminus C_z, p) \le \frac{2}{3} \cdot \#(C, p)$ and $\#(C_z, p) \le \frac{2}{3} \cdot \#(C, p)$, or

(2) $\#(C \setminus C_z, p) \le \frac{1}{3} \cdot \#(C, p)$ and $\#(C_u, p) \le \frac{1}{3} \cdot \#(C, p)$ for any child u of z.

Intuitively our first-order procedure tries to guess this node z and its image $z^* \overset{\text{def}}{=} \pi(z)$ and split the contexts C and C^* at these nodes.

For testing that C and C^* are isomorphic, the program guesses two nodes z and z^* and (disjunctively) chooses case (1) or (2). Note that the program cannot be sure that it has correctly guessed z and z^* according to the above intuition. For this reason, the program first tests that the size restrictions from the chosen case (1) or (2) are fulfilled, which is easily possible in $\mathsf{FO}(\le, +, \times)$ as there are at most polylogarithmically many affected nodes. Since $\#(C, C^*, \Delta E) \le \left(\frac{3}{2}\right)^\ell$, this ensures that, by induction, $\text{C-ISO}^{\ell-1}$ is fully correct for all pairs of contexts that will be compared when testing isomorphism of C and C^*. Note that in case (2) the subtrees of any pair of children u_1 and u_2 of z have at most $\frac{2}{3} \cdot \#(C, C^*, \Delta E)$ affected nodes.

Next, the procedure tests that there is an isomorphism between C and C^* that maps z to z^*. The following claim is used:

▷ Claim 15. Suppose z and z^* are nodes in C and C^* that satisfy Condition (1) or (2) with children $Z \uplus Y$ and $Z^* \uplus Y^*$, respectively, with $|Y| = |Y^*|$ constant. Then a first-order formula can test whether there is an isomorphism between the forests $\{\text{subtree}_z(u) \mid u \in Z\}$ and $\{\text{subtree}_{z^*}(u^*) \mid u^* \in Z^*\}$ using the relations $\textsc{c-iso}^{\ell-1}$ and $\#\textsc{iso-siblings}^{\ell-1}$.

Proof. The forests are isomorphic iff for each $u \in Z$ there is a $u^* \in Z^*$ such that $\text{subtree}_z(u) \cong \text{subtree}_{z^*}(u^*)$ and such that the number of nodes $v \in Z$ with $\text{subtree}_z(u) \cong \text{subtree}_z(v)$ is the same as the number of nodes v^* with subtrees $\text{subtree}_{z^*}(u^*) \cong \text{subtree}_{z^*}(v^*)$, and vice versa with roles of u and u^* swapped.

Because z and z^* satisfy condition (1) or (2), $\textsc{c-iso}^{\ell-1}$ is correct on the forest $\{\text{subtree}_z(u) \mid u \in Z\} \cup \{\text{subtree}_{z^*}(u^*) \mid u^* \in Z^*\}$ by induction. Additionally, $\#\textsc{iso-siblings}^{\ell-1}$ is consistent with $\textsc{c-iso}^{\ell-1}$ by induction. From $\textsc{c-iso}^{\ell-1}$, the tree isomorphism relation $\textsc{t-iso}^{\ell-1}$ – storing tuples (x, r, x^*, r^*) such that $\text{subtree}_x(r)$ and $\text{subtree}_{x^*}(r^*)$ are isomorphic and disjoint – is FO-definable.

For testing whether a node $u \in Z$ satisfies the above condition, a first-order formula existentially quantifies a node $u^* \in Z^*$ and checks that $\textsc{t-iso}^{\ell-1}(z, u, z^*, u^*)$. The number of isomorphic siblings of u, u^* is compared using $\#\textsc{iso-siblings}^{\ell-1}$ and subtracting any siblings $y \in Y$ for which $\textsc{t-iso}^{\ell-1}(z, u, z, y)$ (and, respectively, $y^* \in Y^*$ for which $\textsc{t-iso}^{\ell-1}(z^*, u^*, z^*, y^*)$). This is possible in FO because (a) $|Y|$ is constant and (b) $\#\textsc{iso-siblings}^{\ell-1}$ is consistent with $\textsc{c-iso}^{\ell-1}$ (even though $\textsc{c-iso}^{\ell-1}$ is not necessarily complete on subtrees in Y, Y^*). ◁

For testing whether there is an isomorphism between C and C^* mapping z to z^*, the program distinguishes the cases (1) and (2) from above. Further, in each of the cases it distinguishes (A) $\textsc{path}(r, z, h)$ and $\textsc{path}(r^*, z^*, h^*)$, or (B) $\neg\textsc{path}(r, z, h)$ and $\neg\textsc{path}(r^*, z^*, h^*)$. For all these first-order definable cases, a first-order formula can test whether there is an isomorphism between C and C^* mapping z to z^*. The detailed case analysis is deferred to the full version of the paper.

Computing #ISO-SIBLINGS$^\ell$. The relation $\#\textsc{iso-siblings}^\ell$ can be first-order defined from $\textsc{c-iso}^\ell$ and the 4-ary relation $\#\textsc{iso-sibling}_{\text{unchanged}}$ containing tuples

- (x, r, y, m) with $m > 0$ if $\text{subtree}_x(y)$ has no affected nodes and the number of isomorphic siblings of y in $\text{subtree}_x(r)$ with no affected nodes is m; and
- $(x, r, y, 0)$ if $\text{subtree}_x(y)$ contains an affected node.

Thus the relation $\#\textsc{iso-sibling}_{\text{unchanged}}$ contains isomorphism counts for "unchanged" siblings. It is FO-definable from the old auxiliary data (from before the change) and the set of changes.

Given $\#\textsc{iso-sibling}_{\text{unchanged}}$ and $\textsc{c-iso}^\ell$, the relation $\#\textsc{iso-sibling}^\ell$ is FO-definable as follows. Include a tuple (x, r, y, m) into $\#\textsc{iso-sibling}^\ell$ if $m = m_1 + m_2$ where m_1 is the number of unchanged, isomorphic siblings of y and m_2 is the number of isomorphic siblings of y affected by the change (but by at most $(\frac{3}{2})^{\ell-1}$ changes).

The number m_1 can be checked via distinguishing whether $\text{subtree}_x(y)$ has changed or not. If $\text{subtree}_x(y)$ has not changed, the formula checks that m_1 is such that $(x, r, y, m_1) \in \#\textsc{iso-sibling}_{\text{unchanged}}$. If $\text{subtree}_x(y)$ has changed, find a sibling y^* of y with an unchanged, isomorphic subtree. If y^* exists then the formula checks that m_1 is such that $(x, r, y^*, m_1) \in \#\textsc{iso-sibling}_{\text{unchanged}}$, and otherwise that m_1 is 0.

For checking m_2, let $S(y)$ be the set of siblings y^* of y in $\text{subtree}_x(r)$ that contain at least one affected node and where $\textsc{t-iso}^\ell(x, y, x, y^*)$. Since there are at most $\text{polylog}\, n$ changes, $|S(y)| = \mathcal{O}(\text{polylog}\, n)$. Therefore, $|S(y)|$ can be counted and compared to m_2 in FO. ◀

5 Tree decompositions of bounded-treewidth graphs

One of the best-known algorithmic meta-theorems is Courcelle's theorem, which states that all graph properties expressible in monadic second-order logic MSO can be decided in linear time for graphs with tree-width bounded by some constant k [5]. The tree-width is a graph parameter and measures how "tree-like" a graph is and is defined via tree decompositions, see below for details. Courcelle's theorem is based on Bodlaender's theorem, stating that in linear time (1) one can decide whether a graph has tree-width at most k and (2) one can compute a corresponding tree decomposition [3].

Elberfeld, Jakoby and Tantau [13] proved variants of these results and showed that "linear time" can be replaced with "logarithmic space" in both theorem statements. A dynamic version of Courcelle's theorem was proven in [8]: every MSO-definable graph property is in DynFO under changes of single edges. The proof of the latter result circumvented providing a dynamic variant of Bodlaender's theorem, by using the result of Elberfeld et al. that tree decompositions can be computed in LOGSPACE, showing that a tree decomposition can be used to decide the graph property if only logarithmically many single-edge changes occurred after its construction, and that this is enough for maintenance in DynFO.

It is an open problem to generalize the DynFO maintenance result of [8] from single-edge changes to changes of polylogarithmically many edges, even for DynFOLL. Here, we provide an intermediate step and show that tree decompositions for graphs of bounded treewidth can be maintained in DynFOLL. This result may lead to a second strategy for maintaining MSO properties dynamically, in addition to the approach of [8].

A *tree decomposition* (T, B) of a graph $G = (V, E)$ consists of a rooted tree T and a mapping B from the nodes of T to subsets of V. For a tree node t, we call the set $B(t)$ the *bag* of t. A tree decomposition needs to satisfy three conditions. First, every vertex $v \in V$ needs to be included in some bag. Second, for every edge $(u, v) \in E$ there needs to be bag that includes both u and v. Third, for each vertex $v \in V$, the nodes t of T such that $v \in B(t)$ form a connected subgraph in T. The *width* of a tree decomposition is the maximal size of a bag $B(t)$, over all tree nodes t, minus 1. The *treewidth* of a graph G is the minimal width of a tree decomposition for G.

In addition to the width, important parameters of a tree decomposition are its *depth*, the maximal distance from the root to a leaf, and its *degree*, the degree of the tree T. Often, a binary tree decomposition of depth $\mathcal{O}(\log |V|)$ is desirable, while width $\mathcal{O}(k)$ for a graph of treewidth k is tolerable. We show that one can maintain in DynFOLL a tree decomposition of logarithmic depth but with unbounded degree. The proof is deferred to the full version of the paper. It does not use the hierarchical technique; a tree decomposition is defined in FOLL from auxiliary information that is maintained in DynFO.

▶ **Theorem 16.** *For every k, there are numbers $c, d \in \mathbb{N}$ such that a tree decomposition of width ck and depth $d \log n$ can be maintained in* DynFOLL *under changes of* $\text{polylog}(n)$ *edges for graphs of treewidth k, where n is the size of the graph.*

6 Conclusion and discussion

We have shown that most existing maintenance results for DynFO under single tuple changes can be lifted to DynFOLL for changes of polylogarithmic size. A notable exception are queries expressible in monadic second-order logic, which can be maintained on graphs of bounded treewidth under single-tuple changes.

Thus it seems very likely that one can find large classes of queries such that: If a query from the class can be maintained in DynFO for changes of size $\mathcal{O}(1)$, then it can be maintained in DynFOLL for polylogarithmic changes. Identifying natural such classes of queries is an interesting question for future research.

References

1 Eric Allender, Lisa Hellerstein, Paul McCabe, Toniann Pitassi, and Michael E. Saks. Minimizing disjunctive normal form formulas and AC^0 circuits given a truth table. *SIAM J. Comput.*, 38(1):63–84, 2008. doi:10.1137/060664537.

2 David A. Mix Barrington, Peter Kadau, Klaus-Jörn Lange, and Pierre McKenzie. On the complexity of some problems on groups input as multiplication tables. *J. Comput. Syst. Sci.*, 63(2):186–200, 2001. doi:10.1006/JCSS.2001.1764.

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 Stephen A. Cook. Path systems and language recognition. In Patrick C. Fischer, Robert Fabian, Jeffrey D. Ullman, and Richard M. Karp, editors, *Proceedings of the 2nd Annual ACM Symposium on Theory of Computing, May 4-6, 1970, Northampton, Massachusetts, USA*, pages 70–72. ACM, 1970. doi:10.1145/800161.805151.

5 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

6 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, 2018. doi:10.1145/3212685.

7 Samir Datta, Pankaj Kumar, Anish Mukherjee, Anuj Tawari, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of reachability: How many changes can we handle? 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 122:1–122:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.122.

8 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Logical Methods in Computer Science*, 15(2), 2019. doi:10.23638/LMCS-15(2:12)2019.

9 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. 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 120:1–120:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.120.

10 Guozhu Dong and Jianwen Su. First-order incremental evaluation of datalog queries. In *Database Programming Languages (DBPL-4), Proceedings of the Fourth International Workshop on Database Programming Languages - Object Models and Languages*, pages 295–308, 1993.

11 Guozhu Dong and Jianwen Su. Incremental and decremental evaluation of transitive closure by first-order queries. *Inf. Comput.*, 120(1):101–106, 1995. doi:10.1006/inco.1995.1102.

12 Guozhu Dong and Jianwen Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998. doi:10.1006/jcss.1998.1565.

13 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

14 Kousha Etessami. Dynamic tree isomorphism via first-order updates. In *PODS*, pages 235–243, 1998. doi:10.1145/275487.275514,db/conf/pods/Etessami98.html.

15 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19:1–19:36, 2012. doi:10.1145/2287718.2287719.

16 Andrew V. Goldberg and Serge A. Plotkin. Parallel $(\Delta + 1)$-coloring of constant-degree graphs. *Information Processing Letters*, 25(4):241–245, 1987. doi:10.1016/0020-0190(87)90169-4.

17 Erich Grädel and Sebastian Siebertz. Dynamic definability. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 236–248. ACM, 2012. doi:10.1145/2274576.2274601.

18 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

19 Neil Immerman. *Descriptive complexity.* Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

20 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012. doi:10.1007/978-3-642-24508-4.

21 Pablo Muñoz, Nils Vortmeier, and Thomas Zeume. Dynamic graph queries. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, volume 48 of *LIPIcs*, pages 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ICDT.2016.14.

22 Chaoyi Pang, Guozhu Dong, and Kotagiri Ramamohanarao. Incremental maintenance of shortest distance and transitive closure in first-order logic and SQL. *ACM Trans. Database Syst.*, 30(3):698–721, 2005. doi:10.1145/1093382.1093384.

23 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

24 Felix Tschirbs, Nils Vortmeier, and Thomas Zeume. Dynamic complexity of regular languages: Big changes, small work. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 35:1–35:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CSL.2023.35.

25 Nils Vortmeier. *Dynamic expressibility under complex changes.* PhD thesis, TU Dortmund University, Germany, 2019. doi:10.17877/DE290R-20434.

26 Nils Vortmeier and Thomas Zeume. Dynamic complexity of parity exists queries. *Log. Methods Comput. Sci.*, 17(4), 2021. doi:10.46298/LMCS-17(4:9)2021.

Preservation Theorems on Sparse Classes Revisited

Anuj Dawar
Department of Computer Science and Technology, University of Cambridge, UK

Ioannis Eleftheriadis
Department of Computer Science and Technology, University of Cambridge, UK

Abstract

We revisit the work studying homomorphism preservation for first-order logic in sparse classes of structures initiated in [Atserias et al., JACM 2006] and [Dawar, JCSS 2010]. These established that first-order logic has the homomorphism preservation property in any sparse class that is monotone and addable. It turns out that the assumption of addability is not strong enough for the proofs given. We demonstrate this by constructing classes of graphs of bounded treewidth which are monotone and addable but fail to have homomorphism preservation. We also show that homomorphism preservation fails on the class of planar graphs. On the other hand, the proofs of homomorphism preservation can be recovered by replacing addability by a stronger condition of amalgamation over bottlenecks. This is analogous to a similar condition formulated for extension preservation in [Atserias et al., SiCOMP 2008].

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Homomorphism preservation, sparsity, finite model theory, planar graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.47

Funding *Anuj Dawar*: Funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee: grant number EP/X028259/1.
Ioannis Eleftheriadis: Funded by a George and Marie Vergottis Scholarship awarded by Cambridge Trust, an Onassis Foundation Scholarship, and a Robert Sansom Studentship.

1 Introduction

Preservation theorems have played an important role in the development of finite model theory. They provide a correspondence between the syntactic structure of first-order sentences and their semantic behaviour. In the early development of finite model theory it was noted that many classical preservation theorems fail when we limit ourselves to finite structures. An important case in point is the Łoś-Tarski or *extension* preservation theorem, which asserts that a first-order formula is preserved by embeddings between all structures if, and only if, it is equivalent to an existential formula. Interestingly, this was shown to fail on finite structures [15] much before the question attracted interest in finite model theory [12]. On the other hand, the *homomorphism* preservation theorem, asserting that formulas preserved by homomorphisms are precisely those equivalent to existential-positive ones, was remarkably shown to hold on finite structures by Rossman [14], spurring applications in constraint satisfaction and database theory.

However, even before Rossman's celebrated result, these preservation properties were investigated on subclasses of the class of finite structures. In the case of both the extension and homomorphism preservation theorems, the direction of the theorem stating that the syntactic restriction implies the semantic closure condition is easy and holds in restriction to any class of structures. It is the other direction that may fail and, restricting to a subclass weakens both the hypothesis and the conclusion, therefore leading to an entirely new question. Thus, while the class of all finite structures is combinatorially wild, it contains *tame* classes which are both algorithmically and model-theoretically better behaved [6]. A study of

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 47; pp. 47:1–47:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

preservation properties for such restricted classes of finite structures was initiated in [3] and [2], which looked at homomorphism preservation and extension preservation respectively. The focus was on tame classes defined by *sparsity* conditions, which allows for methods based on the *locality* of first-order logic. In particular, the sparsity conditions were based on what have come to be called wideness conditions.

We recall the formal definition of wideness in Section 4 below but, informally, a class of structures $\mathcal{C}$ is called *wide* if in any large enough structure in $\mathcal{C}$ we can find a large set of elements that are pairwise far away from each other. The class $\mathcal{C}$ is *almost wide* if there is a constant s so that in any large enough structure in $\mathcal{C}$, removing at most s elements gives a structure in which we can find a large set of elements that are pairwise far away from each other. Finally, $\mathcal{C}$ is said to be *quasi-wide* if there is a function s so that in any large enough structure in $\mathcal{C}$, removing at most $s(d)$ elements gives a structure in which we can find a large set of elements that are pairwise at distance d from each other. In the latter two cases, we refer to a set of elements whose removal yields a large scatterd set as a *bottleneck* set.

The main result asserted in [3] is that homomorphism preservation holds in any class $\mathcal{C}$ which is almost wide and is *monotone* (i.e. closed under substructures) and *addable* (i.e. closed under disjoint unions). From this, it is concluded that homomorphism preservation holds for any class $\mathcal{C}$ whose Gaifman graphs exclude some graph G as a minor, as long as $\mathcal{C}$ is monotone and addable. The result was extended from almost wide to quasi-wide classes in [7], from which homomorphism preservation was deduced for classes that locally exclude minors and classes that have bounded expansion, again subject to the proviso that they are monotone and addable. Quasi-wide classes were later identified with *nowhere dense* classes, which are now central in structural and algorithmic graph theory [13].

The main technical construction in [3] is concerned with showing that classes of graphs which exclude a minor are indeed almost wide. The fact that homomorphism preservation holds in monotone and addable almost wide classes is deduced from a construction of Ajtai and Gurevich [1] which shows the "density" of minimal models of a first-order sentence preserved by homomorphisms, and the fact that in an almost wide class a collection of such dense models must necessarily be finite. While the Ajtai and Gurevich construction is carried out within the class of all finite structures, it is argued in [3] that it can be carried out in any monotone and addable class because of "the fact that disjoint union and taking a substructure are the only constructions used in the proof" [3, p. 216]. This argument is sketched in a bit more detail in [7, Lemma 7]. The starting point of the present paper is that this argument is flawed. The construction requires us to take not just disjoint unions, but unions that identify certain elements: in other words *amalgamations* over sets of points. On the other hand, we can relax the requirement of monotonicity to just hereditariness (i.e. closure under induced substructures). The conclusion is that homomorphism preservation holds in any class $\mathcal{C}$ that is quasi-wide, hereditary and closed under amalgamation over bottleneck points. The precise statement is given in Theorem 9 below. We also show that the requirements formulated in [3] are insufficient by constructing a class that is almost wide (indeed, has bounded treewidth), is monotone and addable, but fails to have the homomorphism preservation property.

Interestingly, the requirement of amalgamations over bottlenecks is similar to that used to define classes on which the extension preservation property holds in [2], even though the construction uses rather different methods. The result there can be understood, in our terms, as showing that the extension preservation theorem holds in any almost wide, hereditary class with amalgamation over bottlenecks. As we observe below (in Corollary 3), this implies that homomorphism preservation holds in all such classes. Our Theorem 9 then strengthens this to quasi-wide classes, where we do not know if extension preservation holds. The class of planar graphs is an interesting case as it is used in [2] as an example of a hereditary, addable class with excluded minors in which extension preservation fails. We show here that homomorphism preservation also fails in this class, strengthening the result of [2].

In the rest of this paper, we introduce notation and necessary background in Section 2. We construct a monotone, addable class of graphs of small treewidth in Section 3, providing the first counterexample to the claims of [3]. Section 4 states and proves the corrected version of the preservation theorems and Section 5 shows the failure of homomorphism preservation in the class of planar graphs.

2 Preliminaries

We assume familiarity with the standard notions of finite model theory and structural graph theory, and refer to [10] and [13] for reference. We henceforth fix a finite relational vocabulary τ; by a structure we implicitly mean a τ-structure. We often abuse notation and do not distinguish between a structure and its domain. Given two structures A, B, a homomorphism $f : A \to B$ is a map such that for all relation symbols R and tuples $\bar{a}$ from A we have $\bar{a} \in R^A \implies f(\bar{a}) \in R^B$. If moreover $f(\bar{a}) \in R^B \implies \bar{a} \in R^A$ then f is said to be *strong*. An injective strong homomorphism is called an *embedding*. We also call a homomorphism $f : A \to B$ *full* if it is surjective and for any relation symbol R and tuple $\bar{b}$ from B we have $\bar{b} \in R^B \implies \exists \bar{a} \in R^A$ with $f(\bar{a}) = \bar{b}$.

A structure B is said to be a *weak substructure* of a structure A if $B \subseteq A$ and the inclusion map $\iota : B \hookrightarrow A$ is a homomorphism. Likewise, B is an *induced substructure* of A if the inclusion map is an embedding. Given a structure A and a subset $S \subseteq A$ we write $A[S]$ for the unique induced substructure of A with underlying set S. An induced substructure B of A is said to be *free in* A if there is some structure C such that A is the disjoint union $B + C$. Finally, a substructure B of A is said to be *proper* if the inclusion map is not full. We say that a class of structures is *monotone* if it is closed under weak substructures, and it is *hereditary* if it is closed under induced substructures. Moreover a class is called *addable* if it is closed under taking disjoint unions. We often consider classes of undirected graphs. Seen as a relational structure, this is a set with an irreflexive symmetric relation E on it. A weak substructure of such a graph need not be a graph. However, when we speak of a monotone class of undirected graphs, we mean it in the usual sense of a class of graphs closed under the operations of removing edges and vertices.

Given a structure A and an equivalence relation $E \subseteq A \times A$ we define the quotient structure A/E as the structure whose domain $A/E = \{[a]_E : a \in A\}$ is the set of E-equivalence classes and such that for every relation symbol R of arity n we have $R^{A/E} = \{([a_1], \dots, [a_n]) \in A/E : (a_1, \dots, a_n) \in R^A\}$. Clearly, the quotient map $\pi_E : A \twoheadrightarrow A/E$ is a full homomorphism which we call the *quotient homomorphism*. Given structures A, B and a set $S \subseteq A \cap B$ such that $A[S] = B[S]$, we write $A \oplus_S B$ for the *free amalgam of A and B over S*. This can be defined as the quotient of the disjoint union $A + B$ by the equivalence relation generated by $\{(\iota_A(s), \iota_B(s)) : s \in S\}$, where $\iota_A : S \to A$ and $\iota_B : S \to B$ are the inclusion maps. Evidently, there is an injective homomorphism $A \to A \oplus_S B$ given by composing the inclusion $A \to A + B$ with the quotient $A + B \to A \oplus_S B$, and a full homomorphism $A + A \to A$ which descends to a full homomorphism $A \oplus_S A \to A$.

By the *Gaifman graph* of a structure A we mean the undirected graph $\mathsf{Gaif}(A)$ with vertex set A such that two elements are adjacent if, and only if, they appear together in some tuple of a relation of A. Given a structure A, $r \in \mathbb{N}$, and $a \in A$, we write $B^r_A(a)$ for the *ball of radius r around a in A*, that is, the set of elements of M whose distance from a

in $\mathsf{Gaif}(A)$ is at most r; we shall often abuse notation and write $B^r_A(a)$ to mean the induced substructure $A[B^r_A(a)]$ of A, possibly with a constant for the element a. A set $S \subseteq A$ is said to be *r-independent* if $b \notin B^r_A(a)$ for any $a, b \in S$.

For $r \in \mathbb{N}$, let $\mathrm{dist}(x, y) \leq r$ be the first-order formula expressing that the distance between x and y in the Gaifman graph is at most r, and $\mathrm{dist}(x, y) > r$ its negation. Evidently, the quantifier rank of $\mathrm{dist}(x, y) \leq r$ less than r. A *basic local sentence* is a sentence

$$\exists x_1, \ldots, x_n(\bigwedge_{i \neq j} \mathrm{dist}(x_i, x_j) > 2r \wedge \bigwedge_{i \in [n]} \psi^{B^r(x_i)}(x_i)),$$

where $\psi^{B^r(x_i)}(x_i)$ denotes the relativisation of ψ to the r-ball around x_i, i.e. the formula obtained from ψ by replacing every quantifier $\exists x\ \theta$ with $\exists x(\mathrm{dist}(x_i, x) \leq r \wedge \theta)$, and likewise every quantifier $\forall x\ \theta$ with $\forall x(\mathrm{dist}(x_i, x) \leq r \to \theta)$. We call r the *locality radius*, n the *width*, and ψ the *local condition* of ϕ. Recall the Gaifman locality theorem [10, Theorem 2.5.1].

▶ **Theorem 1** (Gaifman Locality). *Every first-order sentence of quantifier rank q is equivalent to a Boolean combination of basic local sentences of locality radius 7^q.*

We say that a formula ϕ is preserved by homomorphisms (resp. extensions) over a class of structures $\mathcal{C}$ if for all $A, B \in \mathcal{C}$ such that there is a homomorphism (resp. embedding) from A to B, $A \models \phi$ implies that $B \models \phi$. We say that a class of structures $\mathcal{C}$ has the *homomorphism preservation property* (resp. *extension preservation property*) if for every formula ϕ preserved by homomorphisms (resp. extensions) over $\mathcal{C}$ there is an existential-positive (resp. existential) formula ψ such that $M \models \phi \iff M \models \psi$ for all $M \in \mathcal{C}$.

Given a formula ϕ and a class of structures $\mathcal{C}$, we say that $M \in \mathcal{C}$ is a *minimal induced model* of ϕ in $\mathcal{C}$ if $M \models \phi$ and for any proper induced substructure N of M with $N \in \mathcal{C}$ we have $N \not\models \phi$. The relationship between minimal models and preservation theorems is highlighted by the following lemma, which is standard, and combines [3, Theorem 3.1] and [2, Lemma 2.1]

▶ **Lemma 2.** *Let $\mathcal{C}$ be a hereditary class of finite structures. Then a sentence preserved by homomorphisms (resp. extensions) in $\mathcal{C}$ is equivalent to an existential-positive (resp. existential) sentence over $\mathcal{C}$ if and only if it has finitely many minimal induced models in $\mathcal{C}$.*

▶ **Corollary 3.** *Let $\mathcal{C}$ be a hereditary class of finite structures. If $\mathcal{C}$ has the extension preservation property, then $\mathcal{C}$ has the homomorphism preservation property.*

Proof. If a formula ϕ is preserved by homomorphisms then, in particular, it is preserved by extensions. It follows that ϕ is equivalent to an existential sentence over $\mathcal{C}$, and so by Lemma 2 it has finitely many minimal induced models in $\mathcal{C}$. Consequently, the same lemma implies that ϕ is equivalent to an existential-positive sentence over $\mathcal{C}$ as required. ◀

Another immediate consequence of the above is that both preservation properties hold over any class $\mathcal{C}$ that is *well-quasi-ordered* by the induced substructure relation, i.e. for every infinite subclass $\{M_i : i \in I\} \subseteq \mathcal{C}$ there are $i \neq j \in I$ such that either M_i is an induced substructure of M_j or vice versa. In fact, any property (i.e. not necessarily definable by a first-order formula) preserved by homomorphisms (resp. extensions) is equivalent to an existential positive (resp. existential) formula over a well-quasi-ordered class. In particular, this applies to classes of cliques or more generally classes of bounded shrub-depth [11].

3 Preservation can fail on classes of small treewidth

Theorem 4.4 of [3] can be paraphrased in the language of this paper as saying that *homomorphism preservation holds over any monotone and addable class of bounded treewidth.* In this section we provide a simple counterexample to this, exhibiting a monotone and addable class of graphs of treewidth 3 where homomorphism preservation fails. More generally, this contradicts Corollary 3.3 of [3] and Theorem 9 of [7]. To witness failure of preservation, we must exhibit the relevant class, a formula preserved by homomorphisms over this class, and an infinite collection of minimal induced models in the class. We then conclude by Lemma 2.

▶ **Definition 4.** *Fix $k \in \mathbb{N}$ and $n_i \geq 3$ for every $i \in [k]$. We define the bouquet of cycles of type $(n_1, \dots, n_k)$, denoted by $W_{n_1,\dots,n_k}$, as the graph obtained by taking the disjoint union of k cycles of length $n_1, \dots, n_k$ respectively, and adding an apex vertex, i.e. a vertex adjacent to every vertex in these cycles. When $k = 1$, we refer to the graph W_n as the wheel of order n.*

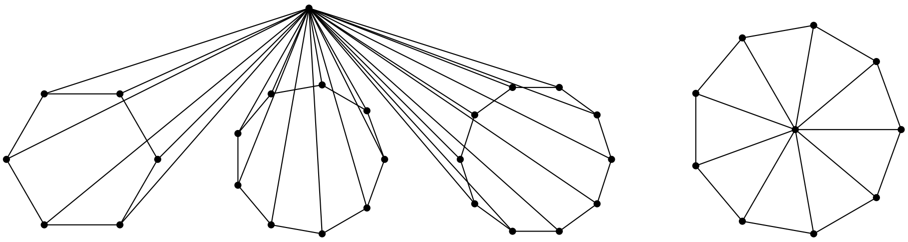


Figure 1 The bouquet of cycles of type $(6, 9, 10)$ and the wheel of order 9 respectively.

▶ **Lemma 5.** *Fix $n, m \in \mathbb{N}$ odd. Then the wheel W_n has chromatic number 4, while its proper subgraphs have chromatic number 3. Consequently, any homomorphism $W_n \to W_m$ is full.*

Proof. Fix $n, m \in \mathbb{N}$ odd. Clearly, any proper colouring of W_n must use a unique colour for the apex as it is adjacent to every other node in W_n. Moreover, we require an additional three colours to colour the vertices in the odd-length cycle of W_n. It follows that $\chi(W_n) = 4$. Now, let W be a proper subgraph of W_n. It follows that there is at least one edge (u, v) present in W_n which is not in W. If u and v are both in the cycle of odd length then we may define a proper 3-colouring of W by giving u and v the same colour, alternating between this and a second colour along the cycle, and using a final third colour for the apex. If one of u or v is the apex of W_n, then we once again colour u and v with the same colour and use an additional two colours to alternate between along the cycle. In particular, it follows that the chromatic number of any homomorphic image of W_n is at least 4, and so it cannot be a proper substructure of W_m. This implies that any homomorphism $f : W_n \to W_m$ is full. ◀

The advantage of working with bouquets of cycles is that, unlike single cycles, there is a formula that asserts the existence of such a structure as a free induced subgraph. Indeed, let

$$\phi := \exists x \exists y [E(x, y) \wedge \forall z (z \neq x \wedge \mathrm{dist}(x, z) \leq 2 \to E(x, z) \wedge \psi(x, z)], \text{ where}$$

$$\psi(x, z) := \exists u \exists v [u \neq v \wedge u \neq x \wedge v \neq x \wedge E(z, u) \wedge E(z, v) \wedge \forall w (E(w, z) \to w = u \vee w = v \vee w = x)]$$

Intuitively, ϕ asserts the following: “there is a vertex x of degree at least one such that every other vertex reachable from x by a path of length two is adjacent to x and has exactly two distinct neighbours which are not x”.

▶ **Lemma 6.** *Let G be an arbitrary finite graph. Then $G \models \phi$ if, and only if, it contains a bouquet of cycles as a free induced subgraph.*

Proof. Suppose that G contains a bouquet W as a free induced subgraph. Then the apex of the bouquet is a vertex of degree at least one, while every vertex reachable from the apex by a path of length two must be in one of the cycles, since W is free in G. Since all vertices in the cycles have degree exactly two, not considering the apex itself, it follows that $G \models \phi$.

Conversely, suppose that $G \models \phi$, and let x be the vertex that is guaranteed to exist by ϕ. Let $S \subseteq V(G)$ be the vertices that are adjacent to x. Since x has degree at least one it follows that S is non-empty. Partition S into k classes $S_1, \ldots, S_k$, by putting two vertices in the same class if, and only if, there is a path between them in $G \setminus \{x\}$. We argue that for each $i \in [k]$, S_i is a free induced cycle in $G \setminus \{x\}$. First, notice that since every vertex reachable from x by a path of length two has degree exactly two in $G \setminus \{x\}$, it follows that S_i induces a cycle in $G \setminus \{x\}$. Moreover, there is no $y \in G \setminus (\{x\} \cup S_i)$ which is adjacent to S_i. Indeed, if $y \neq x$ is adjacent to some $v \in S_i$ then y is reachable from x by a path of length two. It follows that y is itself adjacent to x, and therefore y is in S; in particular, y and v are in the same class and so $y \in S_i$. Consequently each S_i is a free induced cycle in $G \setminus \{x\}$, and so the connected component of x is a free induced bouquet of cycles in G. ◀

It is evident that ϕ is not preserved by homomorphisms in general as every W_n maps homomorphically to the structure $W_n \cup \{c\}$ with an additional vertex adjacent to the apex, and while the latter contains a bouquet as an induced subgraph, it does not contain a bouquet as a free induced subgraph. However, when restricting to subgraphs of disjoint unions of wheels we no longer have non-free-occurring bouquets. This is the core of the next argument.

▶ **Theorem 7.** *Let $\mathcal{C}$ be the monotone and addable closure of $\{W_{2n+1} : n \in \mathbb{N}\}$. Then homomorphism preservation fails over $\mathcal{C}$.*

Proof. Let ϕ be as above; we argue that it is preserved by homomorphisms over $\mathcal{C}$. Indeed, if some $G \in \mathcal{C}$ is such that $G \models \phi$ then by Lemma 6 it contains a bouquet of cycles as a free induced subgraph. By the choice of $\mathcal{C}$, this necessarily implies that G contains W_n as a free induced subgraph for some odd n. Let $H \in \mathcal{C}$ and $f : G \to H$ be a homomorphism. Then f restricts onto a homomorphism $W_n \to H$ which, by the connectivity of W_n and the fact that $H \in \mathcal{C}$, descends to a homomorphism $\hat{f} : W_n \to W_m$ for some odd $m \in \mathbb{N}$. It follows by Lemma 5 that $\hat{f}$ is full, and therefore H contains W_m as a subgraph. The choice of $\mathcal{C}$ once again ensures that W_m is a free induced subgraph of H, and so Lemma 6 implies that $H \models \phi$ as required. Finally, Lemma 6 implies that every W_n is a model of ϕ, while every proper subgraph of W_n cannot possibly contain a bouquet of cycles as a free induced subgraph, and so it cannot model ϕ. Consequently, each W_n is a minimal model of ϕ in $\mathcal{C}$, and so we conclude by Lemma 2 that ϕ is not equivalent to an existential-positive formula over $\mathcal{C}$. ◀

Finally, observe that each graph W_n has treewidth 3 (in fact even pathwidth 3). Indeed, taking a tree decomposition of the cycle C_n of width 2, and adding the apex to every bag in the decomposition gives the required tree decomposition of W_n.

4 Preservation under bottleneck amalgamation

The results of [3] were later extended to classes that are *quasi-wide*. Recall that a class is called quasi-wide if for every $r \in \mathbb{N}$ there exist $k_r \in \mathbb{N}$ and $f_r : \mathbb{N} \to \mathbb{N}$ such that for all $m \in \mathbb{N}$ and $M \in \mathcal{C}$ of size at least $f_r(m)$ there are disjoint sets $A, S \subseteq M$ with $|A| \geq m$, $|S| \leq k_r$ and such that A is r-independent in $M \setminus S$. Intuitively, quasi-wideness ensures that, for every choice of distance, we may find in suitably large structures a large set of elements that are pairwise far away after removing a small set of *bottleneck points*. This notion was introduced for the purpose of extending the arguments of [3] to more general sparse classes, such as classes of bounded expansion, or classes that locally exclude a minor. Paraphrased into the language of this paper, Theorem 9 of [7] asserts that: *homomorphism preservation holds over any monotone and addable quasi-wide class.* Evidently, this is violated by Theorem 7 above. Nonetheless, we may salvage the proof by replacing additivity by the stronger assumption of closure under amalgamation over the bottleneck points that witness quasi-wideness.

The proof proceeds by arguing that any suitably large model M of a sentence ϕ preserved by homomorphisms over a class $\mathcal{C}$ satisfying our assumptions, has a proper induced substructure N which also models ϕ. We thus obtain a concrete bound on the size of minimal models of ϕ, and conclude by Lemma 2. The existence of this bound is guaranteed by quasi-wideness, as any large enough structure contains a large scattered set after removing a small number of bottleneck points. To isolate the bottleneck points $\bar{p}$ of M we consider a structure $\bar{p}M$ in an expanded language which is bi-interpretable with M, and work with the corresponding interpretation ϕ^k of ϕ; in particular $\bar{p}M$ contains a large scattered set itself and it models ϕ^k. Then, by removing a carefully chosen point from the scattered set of $\bar{p}M$, we obtain a proper induced substructure $\bar{p}N$ of $\bar{p}M$ such that $N \in \mathcal{C}$ by hereditariness. To argue that this still models ϕ^k, we use a relativisation of the locality argument of Ajtai and Gurevich from [1]. While in its original version the argument only considers disjoint copies of M, working with the interpretation $\bar{p}M$ of M corresponds to taking free amalgams of M over the set of bottleneck points; this is precisely the subtlety that was missed in [3] and [7].

We now define the structure $\bar{p}M$; in the following we only consider the case of undirected graphs for simplicity. For arbitrary relational structures the idea is analogous, in that we isolate the tuple $\bar{p}$ by forgetting any relation that contains some p_i, and introduce new relation symbols of smaller arities to recover the forgotten relations.

▶ **Definition 8.** *Fix $k \in \mathbb{N}$, and let $\sigma = \{E, P_1, \dots, P_k, Q_1, \dots, Q_k\}$ be the expansion of the language of graphs with $2k$ unary predicates. Given a graph $G = (V, E)$ and a tuple $\bar{p} \in V^k$, define the σ-structure $\bar{p}G$ on the same domain V such that for all $i \in [k]$:*

- $E^{\bar{p}G} = \{(u, v) \in E : u, v \notin \{p_1, \dots, p_k\}\}$;
- $P_i^{\bar{p}G} = \{p_i\}$;
- $Q_i^{\bar{p}G} = \{v \in V : (p_i, v) \in E\}$.

Consider the formula $\epsilon(x, y) := \bigvee_{i\in[k]}(P_i(x) \wedge Q_i(y)) \vee E(x, y)$. Given a sentence ϕ, write ϕ^k for the σ-sentence obtained by ϕ by replacing every atom $E(x, y)$ by $(\epsilon(x, y) \vee \epsilon(y, x))$. It is then clear that for every $G = (V, E)$ and $\bar{p} \in V^k$ as above $G \models \phi \iff \bar{p}G \models \phi^k$.

With this, we turn to our main theorem in this section. Instead of proving a base case and invoking that for the interpretation step as in [2], [3], and [7], we opt for a direct proof to illustrate the relevance of our assumptions on the class.

▶ **Theorem 9.** *Let $\mathcal{C}$ be hereditary such that for every $r \in \mathbb{N}$ there exist $k_r \in \mathbb{N}$ and $f_r : \mathbb{N} \to \mathbb{N}$ so that for every $m \in \mathbb{N}$ and $M \in \mathcal{C}$ of size $\geq f_r(m)$ there are disjoint sets $A, S \subseteq M$ with:*

- *$|A| \geq m$ and $|S| \leq k_r$;*
- *A being r-independent in $M \setminus S$;*
- *$\oplus_S^n M := \underbrace{M \oplus_S M \oplus_S \cdots \oplus_S M}_{n \text{ times}} \in \mathcal{C}$ for every $n \in \mathbb{N}$.*

Then homomorphism preservation holds over $\mathcal{C}$.

Proof. Let $\mathcal{C}$ be as above, and fix ϕ which is preserved by homomorphisms over $\mathcal{C}$. Denote the quantifier rank of ϕ by q, and let $r = 2 \cdot 7^q$. It follows that there is some $k \in \mathbb{N}$ and some $f : \mathbb{N} \to \mathbb{N}$ such that for every $m \in \mathbb{N}$ and every M in $\mathcal{C}$ of size at least $f(m)$, there are disjoint sets $A, S \subseteq M$ such that $|A| \geq m$, $|S| \leq k$, $|A|$ is r-independent in $M \setminus S$, and $\oplus^n_S M \in \mathcal{C}$ for every $n \in \mathbb{N}$. We consider the formula ϕ^k: by Gaifman locality, there is a set $\{\phi_1, \dots, \phi_s\}$ of basic local sentences such that ϕ^k is equivalent to a Boolean combination of these. For $i \in [s]$, let r_i and n_i be the radius and width of locality respectively of ϕ_i, and ψ_i its local condition. Observe that ϕ and ϕ^k have the same quantifier rank, and so $2 \cdot \max_{i \in [s]} r_i \leq r$. Set $n := \max_{i \in [s]} n_i$ and $m := 2^s + 1$. We argue that every minimal model $M \in \mathcal{C}$ of ϕ has size $< f(m)$.

Let M be a minimal model of ϕ, and assume for a contradiction that $|M| \geq f(m)$. It follows that there is a set $S \subseteq M$ of size k such that $M \setminus S$ contains an r-independent set A of size m. Let $\bar{p} \in M^k$ be an enumeration of S; this implies that $\bar{p}M \models \phi^k$ and A is an r-independent set in $\bar{p}M$. For each $i \in [s]$ define

$$\Psi_i(x) := \exists y(\mathrm{dist}(x, y) \leq r_i \wedge \psi_i^{B^{r_i}(y)}(y)).$$

Since $|A| \geq m = 2^s + 1$, it follows that there are at least two vertices $u, v \in A$ satisfying

$$B^{2r_i}_{\bar{p}M}(u) \models \Psi_i(u) \iff B^{2r_i}_{\bar{p}M}(v) \models \Psi_i(v)$$

for all $i \in [s]$. Let N' be the substructure of $\bar{p}M$ induced on $\bar{p}M \setminus \{u\}$. Since A does not intersect the vertices in S, the substructure N of M induced on $M \setminus \{u\}$ satisfies that $N' = \bar{p}N$. We shall argue that $\bar{p}N \models \phi^k$ and so $N \models \phi$, contradicting that M is a minimal induced model of ϕ.

By our closure assumptions on $\mathcal{C}$, $N_n := \oplus^n_S N$ and $M_n := M \oplus_S (\oplus^n_S N)$ are both in $\mathcal{C}$, as the latter is an induced substructure of $\oplus^{n+1}_S M$. Since there is a homomorphism $M \to M_n$ we obtain that $M_n \models \phi$ and thus $\bar{p}M_n \models \phi^k$. We shall argue that

$$\bar{p}M_n \models \phi_i \iff \bar{p}N_n \models \phi_i$$

for all basic local sentences ϕ_i of ϕ^k. In particular, this implies that $\bar{p}N_n \models \phi^k$, and so $N_n \models \phi$. Since there is a homomorphism $N_n \to N$, the preservation of ϕ implies that $N \models \phi$ as claimed.

Clearly, if $\bar{p}N_n \models \phi_i$ then $\bar{p}M_n \models \phi_i$ by the fact that ϕ_i is a local sentence and $\bar{p}N_n$ a free induced substructure of $\bar{p}M_n$. Conversely, if $\bar{p}M_n \models \phi_i$ then there is a $2r_i$-independent subset X of size n_i such that $B^{r_i}_{\bar{p}M_n}(x) \models \psi_i(x)$ for every $x \in X$. Observe that if $X \subseteq S$ then clearly $\bar{p}N_n \models \phi_i$. So, since S is isolated in $\bar{p}M_n$ and $\bar{p}N_n$, we focus on elements of $X \setminus S$, which we may assume to be non-empty. We therefore distinguish two cases.

If $|X \setminus S| > 1$ then, by the $2r_i$-independence of X, there is at least one $x \in X \setminus S$ such that $u \notin B^{r_i}_{\bar{p}M_n}(x)$. It follows that the r_i-ball centered at x is isomorphic to an r_i-ball centered at an element in a disjoint copy of $N \setminus S$. Since $\bar{p}N_n$ contains $n \geq n_i$ such copies, it follows that there is a $2r_i$-independent subset Y of N_n of size n_i such that $B^{r_i}_{\bar{p}N_n}(y) \models \psi_i(y)$ for every $y \in Y$, i.e. $\bar{p}N_n \models \phi_i$.

On the other hand if $|X \setminus S| = 1$, let x be the unique element of $X \setminus S$. Clearly if $u \notin B^{r_i}_{\bar{p}M_n}(x)$ then the r_i-ball centered at x is isomorphic to the r_i-ball centered at an element in $\bar{p}N_n$, and so $\bar{p}N_n \models \phi_i$. If $u \in B^{r_i}_{\bar{p}M_n}(x)$, then $\bar{p}M \models \Psi_i(u)$, and so by the choice of u and v, $\bar{p}M \models \Psi_i(v)$. Consequently, there is some $y \in B^{r_i}_{\bar{p}M}(v)$ such that $B^{r_i}_{\bar{p}M}(v) \models \psi_i(v)$. Observe that because v and u are $2r_i$-independent, $u \notin B^{r_i}_{\bar{p}M_n}(y)$. As before, this implies that $\bar{p}(B_n) \models \phi_i$.

The above implies that there are finitely many minimal induced models of ϕ in $\mathcal{C}$, and so we conclude that ϕ is equivalent to an existential-positive formula over $\mathcal{C}$ by Lemma 2. ◀

Going back to bouquets of cycles, it is easy to see that if a bouquet has more than $m^2 \cdot (r+1)$ vertices then after removing the apex it either contains m disjoint cycles or it contains a cycle of size at least $m \cdot (r+1)$; in either case it contains an r-independent set of size m. In this case the apex is the only bottleneck point, and so amalgamating over this corresponds to adding more cycles to the bouquet. Consequently, homomorphism preservation holds for the hereditary closure of the class of bouquets of cycles by Theorem 9 above.

Closure under amalgamation over bottlenecks is a technical condition and one might consider if it could be replaced by more natural conditions. For example, we could strengthen it by considering closure under arbitrary amalgamation. However, this is a condition that does not sit well with sparsity requirements. Indeed, any hereditary class of undirected graphs that is also closed under arbitrary amalgamation contains arbitrarily large 1-subdivided cliques, and hence, cannot be quasi-wide. Nonetheless, there are naturally defined sparse families of structures that satisfy the conditions of Theorem 9. One such class is known to exist by [2], that is, the class $\mathcal{T}_k$ of all graphs of treewidth bounded by k, for any value of $k \in \mathbb{N}$. Indeed, for any suitably large graph of bounded treewidth we may pick a set of bottleneck points that comes from the same bag in a tree decomposition of the graph, and so amalgamating over this set of points does not increase the treewidth. Another naturally defined such class is the class of outerplanar graphs. For our purposes, we may define outerplanar graphs as those omitting K_4 and $K_{2,3}$ as minors [4]. The quasi-wideness of this class follows by the following fact, which moreover permits some control over the bottleneck points.

▶ **Theorem 10** ([3]). *For every $k, r, m \in \mathbb{N}$ there is an $N = N(k, r, m) \in \mathbb{N}$ such that if G is a graph of size at least N excluding K_k as a minor, then there are disjoint sets $A, S \subseteq V(G)$ with $|A| \geq m$ and $|S| \leq k-2$ such that A is $2r$-independent in $G \setminus S$. Moreover, the bipartite graph $K_{A,S}$ with parts A and S defined by putting an edge between $a \in A$ and $s \in S$ if and only if there is some $u \in B^r_{G\setminus S}(a)$ such that $(u, s) \in E(G)$ is complete.*

▶ **Theorem 11.** *Homomorphism preservation holds for the class of outerplanar graphs.*

Proof. Since outerplanar graphs are K_4-minor-free, it follows by Theorem 10 that for every $r, m \in \mathbb{N}$ there exists an $N = N(r, m) \in \mathbb{N}$ such that if G is an outerplanar graph of size at least N, then there are disjoint sets $A, S \subseteq V(G)$ with $|A| \geq m, |S| \leq 2$, A r-independent in $G \setminus S$, and $K_{A,S}$ complete. Since outerplanar graphs also forbid $K_{2,3}$ as a minor, this implies that $|S| \leq 1$. It is then clear that for any such G and $S \subseteq V(G)$, the graph $\oplus^n_S G$ is still outerplanar for all $n \in \mathbb{N}$, as no 1-point free amalgams can create K_4 or $K_{2,3}$ minors. We thus conclude by Theorem 9. ◀

Interestingly, the examples exhibited above are in fact *almost-wide*, that is, the number of the bottleneck points does not depend on the radius of independence. It would be interesting to find natural quasi-wide classes which are not almost-wide, and which are closed under bottleneck amalgamation. One potential candidate might be the class of all graphs whose local treewidth is bounded by the same constant.

5 Homomorphism preservation fails on planar graphs

In this section we witness that homomorphism preservation fails on the class of planar graphs. Previously, it was established [2] that the extension preservation property fails on planar graphs. Since extension preservation implies homomorphism preservation on hereditary classes by Corollary 3, our result strengthens the above. Recall that by Wagner's theorem [16] a graph is planar if and only if it omits $K_{3,3}$ and K_5 as minors. Our construction, in fact, also reveals that homomorphism preservation fails on the class of K_5-minor-free graphs.

▶ **Definition 12.** *Fix $n \in \mathbb{N}$. Define G_n as the undirected graph on vertex set $V(G_n) = \{v_1, v_2\} \cup \{a_i : i \in [n]\} \cup \{b_i : i \in [n]\}$ and edge set*

$$E(G_n) = \{(v_1, a_i) : i \in [n]\} \cup \{(v_2, b_i) : i \in [n]\} \cup \{(a_i, b_i) : i \in [n]\}$$

$$\cup \{(a_i, a_{i+1}) : i \in [n-1]\} \cup \{(b_i, b_{i+1}) : i \in [n-1]\} \cup \{(a_{i+1}, b_i) : i \in [n-1]\}.$$

We define D_n as the extension of G_n on the same vertex set, with

$$E(D_n) = E(G_n) \cup \{(a_1, a_n), (b_1, b_n), (a_1, b_n)\}.$$

We also define A_n as the graph obtained from G_n by taking the quotient over the equivalence relation generated by (a_1, a_n), and we write $\alpha_n : G_n \to A_n$ for the corresponding quotient homomorphism. Likewise, we let $B_n := G_n/(a_1, b_n)$ and $C_n := G_n/(b_1, b_n)$, and write $\beta_n : G_n \to B_n$ and $\gamma_n : G_n \to C_n$ for the respective quotient homomorphisms.

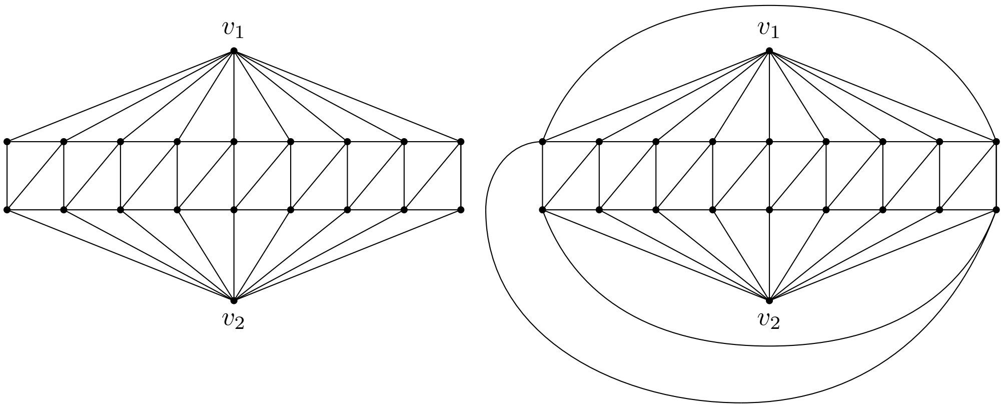


Figure 2 Planar embeddings of G_9 and D_9 respectively.

Consider the following observations. First, for every $n \geq 3$ the graphs G_n, D_n, A_n, B_n, C_n are all planar and 4-chromatic. In particular D_n, B_n, C_n are maximal planar. For $n = 3$ the graphs D_n and B_n contain a copy of K_4, while for $n \geq 4$ they are K_4-free. Likewise, for $n \in \{3, 4\}$ the graphs A_n and C_n contain a copy of K_4, while for $n \geq 5$ they are K_4-free. Finally, for $3 \leq m \leq n$ there is a homomorphism $\delta_{n,m} : G_n \to D_m$ that "wraps" G_n around D_m. Labelling their vertices as above, this satisfies

$$\delta_{n,m}(v_1) = v_1, \delta_{n,m}(v_2) = v_2, \delta_{n,m}(a_i) = a_{i \bmod m} \text{ and } \delta_{n,m}(b_i) = b_{i \bmod m},$$

for all $i \in [n]$.

We proceed to characterise the K_5-minor-free homomorphic images of D_n. We argue that these either contain K_4, or an induced copy of D_m for some m with $m|n$. The proof proceeds by first characterising the K_5-minor-free homomorphic images of G_n by induction, and then using that G_n is a subgraph of D_n. While this requires a fair amount of book-keeping, it is not conceptually difficult. The base case of the induction is the following lemma.

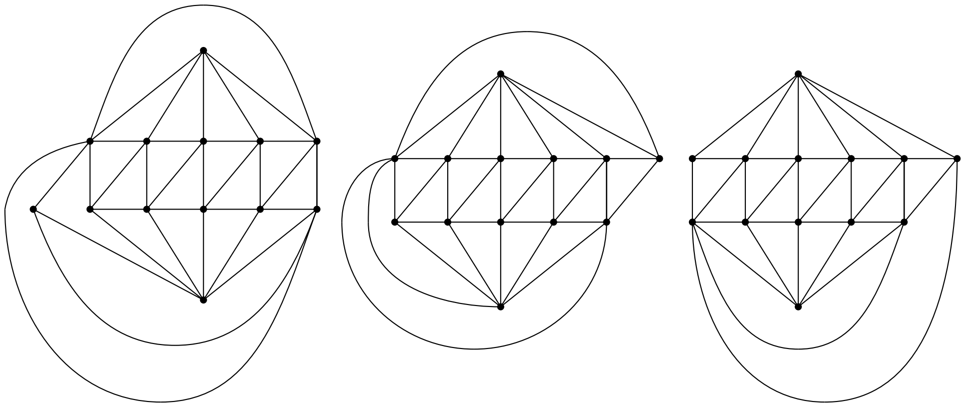
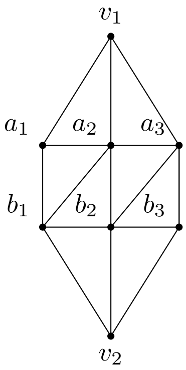


Figure 3 Planar embeddings of A_6, B_6, and C_6 respectively.

▶ **Lemma 13.** *Let $f : G_3 \to H$ be a homomorphism. If H is K_4-free then f is injective.*

Proof. Let H be a K_4-free graph, and $f : G_3 \to H$ a homomorphism. Label the vertices of G_3 as in the picture below; we shall argue that f is injective.



First, notice that f is injective on $\{v_2, b_1, b_2\}$ since they form a triangle in G_3. If $f(a_2) = f(v_2)$ then the set $\{f(v_2), f(b_2), f(a_3), f(b_3)\}$ induces K_4 in H; it follows that $f(a_2) \neq f(v_2)$, and so f is injective on $\{v_2, b_1, b_2, a_2\}$. Likewise, $f(b_3) \neq f(a_2)$ as otherwise $\{f(b_1), f(b_2), f(b_3), f(v_2)\}$ induce K_4 in H. Moreover, $f(b_3) \neq f(b_1)$ as $\{f(b_1), f(b_2), f(a_2), f(a_3)\}$ would otherwise induce K_4 in H. It follows that f is injective on $\{v_2, b_1, b_2, b_3, a_2\}$. From this we deduce that $f(a_3) \neq f(b_1)$ as otherwise $\{f(v_2), f(b_1), f(b_2), f(b_3)\}$ would induce K_4 in H, and that $f(a_3) \neq f(v_2)$ as otherwise $\{f(v_2), f(b_1), f(b_2), f(a_2)\}$ would also induce K_4. Hence, f is injective on $\{v_2, b_1, b_2, b_3, a_2, a_3\}$. By symmetry, it follows that f is also injective on $\{v_1, a_1, a_2, a_3, b_1, b_2\}$. Notice that $f(a_1) \neq f(b_3)$ and $f(v_1) \neq f(b_3)$ since otherwise $\{f(a_2), f(a_3), f(b_2), f(b_3)\}$ would induce K_4 in H. Finally, $f(a_1) \neq f(v_2)$ and $f(v_1) \neq f(v_2)$ as otherwise $\{f(a_2), f(b_1), f(b_2), f(v_2)\}$ would induce K_4 in H. Putting all the above together, we conclude that f is injective on all of G_3 as required. ◀

We now proceed to the general case.

▶ **Lemma 14.** *Fix $n \geq 3$. Let $f : G_n \to H$ be a homomorphism, where H is K_4-free and K_5-minor-free. Then one of the following is true:*

1. *f is injective;*
2. *there is some $m \in [4, n-1]$ and an embedding $\hat{f} : D_m \to H$ such that $f = \hat{f} \circ \delta_{n,m}$;*
3. *$n \geq 5$ and there is an injective homomorphism $\hat{f} : A_n \to H$ such that $f = \hat{f} \circ \alpha_n$;*
4. *$n \geq 4$ and there is an embedding $\hat{f} : B_n \to H$ such that $f = \hat{f} \circ \beta_n$;*
5. *$n \geq 5$ and there is an embedding $\hat{f} : C_n \to H$ such that $f = \hat{f} \circ \gamma_n$;*

Proof. We prove the claim by induction on n. The base case $n = 3$ follows by Lemma 13. So, fix a K_4-free and K_5-minor-free graph H and consider a homomorphism $f : G_{n+1} \to H$. Evidently, this restricts to a homomorphism $f' : G_n \to H$. By the induction hypothesis, we may assume that f' satisfies one of the five conditions of this proposition.

Assume that f' satisfies (1), i.e. f is injective on $G_n = G_{n+1} \setminus \{a_{n+1}, b_{n+1}\}$. We consider the images of the vertices a_{n+1} and b_{n+1} under f. Observe that $(a_{n+1}, b_{n+1}) \in E(G_{n+1})$ so $f(a_{n+1}) \neq f(b_{n+1})$. Clearly, if $f(a_{n+1})$ and $f(b_{n+1})$ are not any of the vertices in $f[G_n]$, then f is itself injective and (1) holds. We hence distinguish three cases.

First, suppose that $f(a_{n+1}) \in f[G_n]$ and $f(b_{n+1}) \notin f[G_n]$. Since there are edges (v_1, a_{n+1}), (a_n, a_{n+1}), and (b_n, a_{n+1}), it follows that $f(a_{n+1}) \notin \{f(v_1), f(a_n), f(b_n)\}$. Moreover, $f(a_{n+1}) \neq f(v_2)$ as otherwise $\{f(b_{n-1}), f(b_n), f(a_n), f(v_2)\}$ would induce K_4 in H. Similarly, $f(a_{n+1}) \neq f(a_{n-1})$, as otherwise $\{f(a_{n-1}), f(b_{n-1}), f(a_n), f(b_n)\}$ would induce K_4, and $f(a_{n+1}) \neq f(b_{n-1})$, as otherwise $\{f(v_1), f(a_{n-1}), f(a_n), f(b_{n-1})\}$ would induce K_4. In addition, $f(a_{n+1}) \notin \{f(a_i) : 2 \leq i \leq n-2\}$ as otherwise an edge $(f(a_i), f(a_n))$ for some $i \in [2, n-2]$ would produce a K_5-minor in H, namely the minor arising from $S_1 = \{f(v_1)\}, S_2 = \{f(a_i)\}, S_3 = \{f(a_n)\}, S_4 = \{f(a_j) : i+1 \leq j \leq n-1\}, S_5 = \{f(v_2), f(b_1), f(a_1), f(b_i), f(b_{i+1}), f(b_n)\}$. A similar argument reveals that $f(a_{n+1}) \notin \{f(b_i) : 2 \leq i \leq n-2\}$. It follows that $f(a_{n+1}) = f(b_1)$ or $f(a_{n+1}) = f(a_1)$. The former case would produce a copy of K_4, namely $\{f(v_1), f(a_1), f(a_2), f(b_1)\}$, leading to a contradiction. Hence $f(a_{n+1}) = f(a_1)$. Since $f(b_{n+1}) \notin f[G_n]$, it follows that f factors through the quotient homomorphism α_n, i.e. case (3) is true.

Next, suppose that $f(a_{n+1}) \in f[G_n]$ and $f(b_{n+1}) \in f[G_n]$. As before, we deduce from the first assumption that $f(a_{n+1}) = f(a_1)$. This implies that there are edges $(f(a_1), f(a_n))$ and $(f(a_1), f(b_n))$ in H. Since there are edges $(a_{n+1}, b_{n+1}), (b_n, b_{n+1}), (v_2, b_{n+1})$ in G_{n+1} we deduce that $f(b_{n+1}) \notin \{f(a_1), f(b_n), f(v_2)\}$. Moreover, $f(b_{n+1}) \neq f(v_1)$ as otherwise the edge $(f(v_1), f(v_2))$ would produce a K_5-minor in H, namely $S_1 = \{f(v_1)\}, S_2 = \{f(a_1)\}, S_3 = \{f(a_n)\}, S_4 = \{f(a_j) : j \in [2, n-1]\}, S_5 = \{f(v_2), f(b_1), f(b_2), f(b_n)\}$. Similarly, a K_5-minor arises in H if $f(b_{n+1}) \in \{f(a_i), f(b_i) : i \in [2, n-1]\}$. We thus deduce that $f(b_{n+1}) = f(b_1)$, from which we conclude that $f[G_{n+1}]$ induces a copy of D_n in H, and more precisely, case (2) holds.

So, suppose that $f(a_{n+1}) \notin f[G_n]$ and $f(b_{n+1}) \in f[G_n]$. We consider the possible images of b_{n+1} under f. Since there are edges $(a_{n+1}, b_{n+1}), (v_2, b_{n+1}), (b_n, b_{n+1})$ in G_{n+1} we deduce that $f(b_{n+1}) \notin \{f(a_{n+1}), f(v_2), f(b_n)\}$. Moreover, $f(b_{n+1}) \neq f(v_1)$ as otherwise $\{f(v_1), f(a_n), f(a_{n+1}), f(b_n)\}$ would induce K_4 in H. Likewise, $f(b_{n+1}) \neq f(a_n)$ as otherwise $\{f(v_2), f(a_{b-1}), f(b_n), f(a_n)\}$ would induce K_4 in H. Moreover $f(b_{n+1}) \notin \{f(a_i) : i \in [2, n-1]\}$ as otherwise the edge $(f(a_i), f(a_{n+1}))$ would produce a K_5-minor in H, namely $S_1 = \{f(v_1)\}, S_2 = \{f(a_i)\}, S_3 = \{f(a_{n+1})\}, S_4 = \{f(a_j) : j \in [i+1, n]\}, S_5 = \{f(v_2), f(b_1), f(a_1), f(b_i), f(b_{i+1}), f(b_n)\}$. Very similarly, we deduce that $f(b_{n+1}) \notin \{f(b_i) : i \in [2, n-1]\}$. It follows that $f(b_{n+1}) = f(a_1)$ or $f(b_{n+1}) = f(b_1)$. In the former case, it follows that f factors through the quotient homomorphism β_n, and so (4) is true, while in the latter case, it follows that f factors through γ_n, and so (5) is true.

Next, assume that f' satisfies (2), i.e. there is some $m \in [4, n-1]$ and and an embedding $\hat{f} : D_m \to H$ such that $f' = \hat{f} \circ \delta_{n,m}$. Arguing as before, it is easy to see that the assumptions on H force $f(a_{n+1})$ to be equal to $\hat{f}(a_{n+1 \bmod m})$, and likewise $f(b_{n+1}) = \hat{f}(b_{n+1 \bmod m})$, implying that $f = \hat{f} \circ \delta_{n+1,m}$. Hence f also satisfies (2).

Finally, we argue that f' cannot satisfy any of (3), (4), and (5). Indeed, assume for a contradiction that f' satisfies (3) and write f' as $\hat{f} \circ \alpha_n$ for some injective homomorphism $\hat{f} : A_n \to H$. In particular, we know that $n \geq 5$. Consider the image of a_{n+1} under f; this is some vertex adjacent to $f(a_n) = f(a_1), f(v_1)$, and $f(b_n)$. If $f(a_{n+1}) \notin f[G_n]$, then we obtain a K_5-minor in H, namely $S_1 = \{f(v_1)\}, S_2 = \{f(a_1)\}, S_3 = \{f(a_{n-1})\}, S_4 = \{f(a_i) : i \in [2, n-1]\}, S_5 = \{f(v_2), f(b_1), f(b_2), f(b_{n-1}), f(b_n), f(a_{n+1})\}$. So $f(a_{n+1}) \in f[G_n]$. If $f(a_{n_1}) = f(a_i)$ for some $i \in [2, n-2]$ then we obtain a K_5-minor in H by picking some $j \in [2, n-2] \setminus \{i\}$ and letting $S_1 = \{f(v_1), f(a_j)\}, S_2 = \{f(a_1)\}, S_3 = \{f(a_{n-1})\}, S_4 = \{f(b_i) : i \in [n-1]\}, S_5 = \{f(v_2), f(b_n), f(a_i)\}$. Likewise, if $f(a_{n+1}) = f(a_{n-1})$ then we obtain the K_5-minor $S_1 = \{f(v_1)\}, S_2 = \{f(a_1)\}, S_3 = \{f(b_{n-1}), f(b_{n-2}), f(a_{n-2})\}, S_4 = \{f(a_i) : i \in [2, n-3]\}, S_5 = \{f(v_2), f(b_n), f(a_{n-1})\}$. Consequently, $f(a_{n+1}) = f(b_i)$ for some $i \in [1, n-1]$. This produces an edge $(f(v_1), f(b_i))$ and thus gives rise to the K_5-minor $S_1 = \{f(v_1)\}, S_2 = \{f(a_j) : j \in [1, i-1]\}, S_3 = \{f(a_i)\}, S_4 = \{f(a_j) : j \in [i+1, n-1]\}, S_5 = \{f(v_2), f(b_1), f(b_i), f(b_{n-1})\}$. It follows that f' cannot satisfy (3); via very similar reasoning, we exclude cases (4) and (5). ◀

Having established the above, our characterisation of the K_5-minor-free homomorphic images of D_n follows easily.

▶ **Proposition 15.** *Fix $n \geq 4$. Then any K_4-free and K_5-minor-free homomorphic image of D_n contains an induced copy of D_m for some $m \geq 4$ such that $m|n$.*

Proof. Consider a homomorphism $f : D_n \to H$ where H is K_4-free and K_5-minor-free. Then f descends to a homomorphism $f' : G_n \to H$. It follows that one of the five cases of Lemma 14 holds. If f' is injective, then in particular f is injective; since the addition of any edge in D_n creates a K_5-minor, it follows that f is in fact an embedding as required. Suppose that case (2) is true, and let $m \in [4, n-1]$ and $\hat{f} : D_m \to H$ be such that $f' = \hat{f} \circ \delta_{n,m}$. In particular, D_m is an induced subgraph of H and $m|n$. Finally, case (3) leads to a contradiction as the edge (a_1, a_n) in D_n implies that $f(a_1) \neq f(a_n)$, case (4) leads to a contradiction as the edge (a_1, b_n) implies that $f(a_1) \neq f(b_n)$, and likewise, case (5) leads to a contradiction as the edge (b_1, b_n) implies that $f(b_1) \neq f(b_n)$. ◀

We also define G_∞ as the countably infinite analogue of G_n, i.e. the graph on the vertex set $V(G_\infty) = \{v_1, v_2\} \cup \{a_i : i \in \mathbb{N}_{>0}\} \cup \{b_i : i \in \mathbb{N}_{>0}\}$ and edge set

$$E(G_n) = \{(v_1, a_i) : i \in \mathbb{N}_{>0}\} \cup \{(v_2, b_i) : i \in \mathbb{N}_{>0}\} \cup \{(a_i, b_i) : i \in \mathbb{N}_{>0}\}$$

$$\cup \{(a_i, a_{i+1}) : i \in \mathbb{N}_{>0}\} \cup \{(b_i, b_{i+1}) : i \in \mathbb{N}_{>0}\} \cup \{(a_{i+1}, b_i) : i \in \mathbb{N}_{>0}\}.$$

Likewise, we define the homomorphism $\delta_{\infty,m} : G_\infty \to D_m$ in analogy to the homomorphisms $\delta_{n,m} : G_n \to D_m$. Lemma 14 allows us to also characterise the finite K_5-minor-free homomorphic images of G_∞.

▶ **Lemma 16.** *Let $f : G_\infty \to H$ be a homomorphism where H is finite, K_4-free, and K_5-minor-free. Then there is some $m \geq 4$ and an embedding $\hat{f} : D_m \to H$ such that $f = \hat{f} \circ \delta_{\infty,m}$.*

Proof. Fix f as above and let $n := |H|$. It follows that f restricts to a homomorphism $f' : G_n \to H$; this satisfies one of the five cases of Lemma 14. Clearly, cases (1),(3),(4), and (5) are ruled out due to size restrictions. Consequently, case (2) holds and the claim follows. ◀

With the above, we show that the existence of the graphs D_n as induced subgraphs is definable among K_4-free K_5-minor-free graphs by a simple first-order formula. Indeed, let

$$\chi(x_1, x_2, y_1, z_1, y_2, z_2) = E(x_1, y_2) \wedge E(y_1, y_2) \wedge E(z_1, y_2) \wedge E(z_1, z_2) \wedge E(y_2, z_2) \wedge E(z_2, x_2), \text{ and}$$

$$\phi = \exists x_1, x_2, y, z[E(x_1, y) \wedge E(y, z) \wedge E(z, x_2) \wedge \forall a, b(E(x_1, a) \wedge E(a, b) \wedge E(b, x_2))$$
$$\to \exists c, d\ \chi(x_1, x_2, a, b, c, d))]$$

▶ **Lemma 17.** *Let H be a finite K_4-free and K_5-minor free graph. If $H \models \phi$ then there is some $n \geq 4$ such that H contains D_n as an induced subgraph.*

Proof. Fix a graph H as above, and suppose that $G \models \phi$. We inductively define a chain of partial homomorphisms $f_1 \subseteq f_2 \subseteq f_3 \subseteq \ldots$ from $G_\infty \to H$ such that $\text{dom}(f_n) = G_n$. Then the map $f = \cup_{n=1}^{\infty} f_n$ is a homomorphism $G_\infty \to H$, and hence Lemma 16 implies that H contains some D_n as an induced subgraph.

Since $H \models \phi$ it follows that there are $x_1, x_2, y, z \in V(H)$ such that

$$H \models E(x_1, y) \wedge E(y, z) \wedge E(z, x_2).$$

Consequently, the map $f_1 : G_1 \to H$ given by $f(v_1) = x_1, f(v_2) = x_2, f(a_1) = y, f(b_1) = z$ is a homomorphism as required. So, suppose that f_n has been defined. Since

$$H \models E(x_1, f(a_n)) \wedge E(f(a_n), f(b_n)) \wedge E(f(b_n), x_2)$$

it follows that

$$H \models \exists c, d\ \chi(x_1, x_2, f(a_n), f(b_n), c, d).$$

We consequently extend $f_n : G_n \to H$ to $f_{n+1} : G_{n+1} \to H$ by letting $f_{n+1}(a_{n+1}) = c$ and $f_{n+1}(b_{n+1}) = d$; this is easily seen to be a valid homomorphism by the choice of χ. ◀

▶ **Lemma 18.** *Let H be a K_5-minor-free graph. If H contains some D_n for $n \geq 3$ as an induced subgraph then $H \models \phi$.*

Proof. Let $D_n \leq H$ be as above. Clearly, $H \models E(v_1, a_1) \wedge E(a_1, b_1) \wedge E(b_1, v_2)$ So, let $a, b \in V(H)$ be arbitrary vertices such that $H \models E(v_1, a) \wedge E(a, b) \wedge E(b, v_2)$; we first argue that $a \in \{a_i : i \in [n]\}$ and $b \in \{b_i : i \in [n]\}$. Towards this, observe first that $b \notin \{a_i : i \in [n]\} \cup \{v_1\}$ as otherwise the edge (v_2, a_i) or (v_2, v_1) would contradict that D_n is induced in H. So, assume for a contradiction that $a \notin \{a_i : i \in [n]\}$. Since there is an edge (v_1, a) it follows that $a \neq v_1$, and so in particular the sets $S_1 = \{v_1\}, S_2 = \{a_1\}, S_3 = \{a_n\}, S_4 = \{a_j : j \in [2, n-1]\}, S_5 = \{v_2, b_1, b_n, a, b\}$ produce a K_5-minor in H. With a symmetric argument we obtain that $b \in \{b_i : i \in [n]\}$. Now, since there is an edge (a, b) it follows that there is some $i \in [n]$ such that $a = a_i$ and $b = b_i$ or $b = b_{i-1 \bmod n}$. In these two respective cases we have that

$$H \models \chi(v_1, v_2, a, b, a_{i+1 \bmod n}, b_{i+1 \bmod n}), \text{ or } H \models \chi(v_1, v_2, a, b, a_{i-1 \bmod n}, b_{i-2 \bmod n}).$$

In either case, $H \models \exists c, d\ \chi(v_1, v_2, a, b, c, d)$, and since the choice of $a, b \in V(H)$ was arbitrary we obtain that $H \models \phi$ as required. ◀

Putting all the above together, we deduce the main theorem of this section.

▶ **Theorem 19.** *The class of planar graphs does not have the homomorphism preservation property.*

Proof. Let $\hat{\phi}$ be the disjunction of ϕ with the formula that induces a copy of K_4, i.e.

$$\hat{\phi} := \phi \vee \exists x_1, x_2, x_3, x_4 \bigwedge_{i \neq j} E(x_i, x_j).$$

We argue that $\hat{\phi}$ is preserved by homomorphisms over the class of planar graphs. Indeed, let $f : G \to H$ be a homomorphism with G, H planar such that $G \models \hat{\phi}$. Clearly, if H contains a copy of K_4 then $H \models \hat{\phi}$. So, without loss of generality we may assume that $G \models \phi$ and G, H are K_4-free. It follows by Lemma 17 that there exists some $n \geq 4$ such that G contains D_n as a subgraph. Consequently, Proposition 15 implies that there is some $m \geq 4$ such that H contains D_m as a subgraph. Lemma 18 then implies that $H \models \phi$, and thus $H \models \hat{\phi}$ as required. To conclude, observe that the minimal models of $\hat{\phi}$ over the class of planar graphs are K_4 and the graphs D_n for $n \geq 4$; since these are infinitely many Lemma 2 implies that $\hat{\phi}$ is not equivalent to an existential-positive formula over the class of planar graphs. ◀

Since we only use exclusion of K_5-minors in the above, we additionally obtain the following.

▶ **Theorem 20.** *The class of all K_5-minor-free graphs does not have the homomorphism preservation property.*

Finally, while we have not referred to topological minors to simplify our arguments, an easy check reveals that the above are still valid when considering graphs that forbid K_5 as a topological minor, implying that homomorphism preservation also fails on the class of K_5-topological-minor-free graphs.

6 Conclusion

Much work in finite model theory explores *tame* classes of finite structures. In [6], two related notions of tameness are identified: algorithmic tameness and model-theoretic tameness. The former is centred around the tractability of model-checking for first-order logic while the latter is illustrated by preservation theorems and it was argued that these occurred together in sparse classes of structures. More recently, algorithmic tameness has been explored extensively for dense classes as well (see [8] for example). On the other hand, the results here show that the status of preservation theorems on sparse classes is more subtle and relies on closure properties that are not always present in natural classes such as planar graphs. Nonetheless, it is an interesting question whether the recent understanding of tame dense classes, such as *monadically stable* and more generally *monadically dependent* classes, can also cast light on preservation theorems. The arguments for homomorphism and extension preservation rely on quasi-wideness and almost-wideness respectively. Similar wideness phenomena occur for dense classes, by replacing deletion of bottleneck points with performing *flips*, that is, edge-complementations within subsets of the domain (see Table 1 in [9]). A question inspired by this is whether, for every $k \in \mathbb{N}$, the class of all graphs of cliquewidth at most k has the extension preservation property. For $k \leq 2$ this follows from the fact that cographs are well-quasi-ordered by the induced subgraph relation [5].

References

1 Miklos Ajtai and Yuri Gurevich. Datalog vs first-order logic. *Journal of Computer and System Sciences*, 49(3):562–588, 1994. 30th IEEE Conference on Foundations of Computer Science. doi:10.1016/S0022-0000(05)80071-6.

2 Albert Atserias, Anuj Dawar, and Martin Grohe. Preservation under extensions on well-behaved finite structures. *SIAM Journal on Computing*, 38(4):1364–1381, 2008. doi:10.1137/060658709.

3 Albert Atserias, Anuj Dawar, and Phokion G Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *Journal of the ACM (JACM)*, 53(2):208–237, 2006.

4 Gary Chartrand and Frank Harary. Planar Permutation Graphs. *Annales de l'institut Henri Poincaré. Section B. Calcul des probabilités et statistiques*, 3(4):433–438, 1967. URL: http://www.numdam.org/item/AIHPB_1967__3_4_433_0/.

5 Peter Damaschke. Induced subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 14(4):427–435, 1990.

6 Anuj Dawar. Finite model theory on tame classes of structures. In *International Symposium on Mathematical Foundations of Computer Science*, pages 2–12. Springer, 2007.

7 Anuj Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76(5):324–332, 2010.

8 Jan Dreier, Ioannis Eleftheriadis, Nikolas Mählmann, Rose McCarty, Michał Pilipczuk, and Szymon Toruńczyk. First-order model checking on monadically stable graph classes. *arXiv preprint*, 2023. Accepted to FOCS 2024. arXiv:2311.18740.

9 Jan Dreier, Nikolas Mählmann, and Szymon Toruńczyk. Flip-breakability: A combinatorial dichotomy for monadically dependent graph classes. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1550–1560, 2024.

10 H-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 2nd edition, 1999.

11 Robert Ganian, Petr Hliněný, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona De Mendez. Shrub-depth: Capturing height of dense graphs. *Logical Methods in Computer Science*, 15, 2019.

12 Yuri Gurevich. Toward logic tailored for computational complexity. *Computation and Proof Theory*, pages 175–216, 1984.

13 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

14 Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM (JACM)*, 55(3):1–53, 2008.

15 W. W. Tait. A counterexample to a conjecture of Scott and Suppes. *Journal of Symbolic Logic*, 24(1):15–16, 1959. doi:10.2307/2964569.

16 Klaus Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114(1):570–590, 1937.

Local Certification of Geometric Graph Classes

Oscar Defrain
Aix Marseille Université, CNRS, LIS, Marseille, France

Louis Esperet
Laboratoire G-SCOP (CNRS, Univ. Grenoble Alpes), Grenoble, France

Aurélie Lagoutte
Laboratoire G-SCOP (CNRS, Univ. Grenoble Alpes), Grenoble, France

Pat Morin
School of Computer Science, Carleton University, Canada

Jean-Florent Raymond
CNRS, LIP, ENS de Lyon, France

Abstract

The goal of local certification is to locally convince the vertices of a graph G that G satisfies a given property. A prover assigns short certificates to the vertices of the graph, then the vertices are allowed to check their certificates and the certificates of their neighbors, and based only on this local view and their own unique identifier, they must decide whether G satisfies the given property. If the graph indeed satisfies the property, all vertices must accept the instance, and otherwise at least one vertex must reject the instance (for any possible assignment of certificates). The goal is to minimize the size of the certificates.

In this paper we study the local certification of geometric and topological graph classes. While it is known that in n-vertex graphs, planarity can be certified locally with certificates of size $O(\log n)$, we show that several closely related graph classes require certificates of size $\Omega(n)$. This includes penny graphs, unit-distance graphs, (induced) subgraphs of the square grid, 1-planar graphs, and unit-square graphs. These bounds are tight up to a constant factor and give the first known examples of hereditary (and even monotone) graph classes for which the certificates must have linear size. For unit-disk graphs we obtain a lower bound of $\Omega(n^{1-\delta})$ for any $\delta > 0$ on the size of the certificates, and an upper bound of $O(n \log n)$. The lower bounds are obtained by proving rigidity properties of the considered graphs, which might be of independent interest.

2012 ACM Subject Classification Mathematics of computing → Graph theory; Computing methodologies → Distributed computing methodologies; Theory of computation → Computational geometry

Keywords and phrases Local certification, proof labeling schemes, geometric intersection graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.48

Related Version *Full Version*: `https://arxiv.org/abs/2311.16953` [5]

Funding *Louis Esperet*: partially supported by the French ANR Project TWIN-WIDTH (ANR-21-CE48-0014-01), and by LabEx PERSYVAL-lab (ANR-11-LABX-0025).
Aurélie Lagoutte: partially supported by French ANR project GRALMECO (ANR-21-CE48-0004).
Pat Morin: partially supported by NSERC.
Jean-Florent Raymond: partially supported by French ANR project GRALMECO (ANR-21-CE48-0004).

1 Introduction

Local certification is an emerging subfield of distributed computing where the goal is to assign short certificates to each of the nodes of a network (some connected graph G) such that the nodes can collectively decide whether G satisfies a given property (i.e., whether it belongs

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 48; pp. 48:1–48:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to some given graph class $\mathcal{C}$) by only inspecting their unique identifier, their certificate and the certificates of their neighbors. This assignment of certificates is called a *proof labeling scheme*, and its *complexity* is the maximum number of bits of a certificate (as a function of the number of vertices of G, which is usually denoted by n in the paper). If a graph class $\mathcal{C}$ admits a proof labeling scheme of complexity $f(n)$, we say that $\mathcal{C}$ has *local complexity* $f(n)$. Proof labelling schemes are distributed analogues of traditional non-deterministic algorithms, and graph classes of logarithmic local complexity can be considered as distributed analogues of classes whose recognition is in NP [7]. The notion of proof labeling scheme was formally introduced by Korman, Kutten and Peleg in [17], but originates in earlier work on self-stabilizing algorithms (see again [7] for the history of local certification and a thorough introduction to the field). While every graph class has local complexity $O(n^2)$ [17],[1] the work of [13] identified three natural ranges of local complexity for graph classes:

- $\Theta(1)$: this includes k-colorability for fixed k, and in particular bipartiteness;
- $\Theta(\log n)$: this includes non-bipartiteness and acyclicity; and
- $\Theta(\text{poly}(n))$: this includes non-3-colorability and problems involving symmetry.

It was later proved in [19] that any graph class which can be recognized in linear time (by a centralized algorithm) has an "interactive" proof labeling scheme of complexity $O(\log n)$, where "interactive" means that there are several rounds of interaction between the prover (the entity which assigns certificates) and the nodes of the network (see also [16] for more on distributed interactive protocols). A natural question is whether the interactions are necessary or whether such graph classes have classical proof labeling schemes of complexity $O(\log n)$ as defined above, that is, without multiple rounds of interaction. This question triggered the work of [9] on planar graphs, which have a well-known linear time recognition algorithm. The authors of [9] proved that the class of planar graphs indeed has local complexity $O(\log n)$, and asked whether the same holds for any proper minor-closed class.[2] This was later proved for graphs embeddable on any fixed surface in [10] (see also [6]) and in [2] for classes excluding small minors, while it was proved in [12] that classes excluding a planar graph H as a minor have local complexity $O(\log^2 n)$. The authors of [12] also proved the related result that any graph class of bounded treewidth which is expressible in second order monadic logic has local complexity $O(\log^2 n)$ (this implies in particular that for any fixed k, the class of graphs of treewidth at most k has local complexity $O(\log^2 n)$). Similar meta-theorems involving graph classes expressible in some logic were proved for graphs of bounded treedepth in [8] and graphs of bounded cliquewidth in [11].

Closer to the topic of the present paper, the authors of [14] obtained proof labeling schemes of complexity $O(\log n)$ for a number of classes of geometric intersection graphs, including interval graphs, chordal graphs, circular-arc graphs, trapezoid graphs, and permutation graphs. It was noted earlier in [15] (which proved various results on interactive proof labeling schemes for geometric graph classes) that the "only" classes of graphs for which large lower bounds on the local complexity are known (for instance non-3-colorability, or some properties involving symmetry) are not *hereditary*, meaning that they are not closed under taking induced subgraphs. It turns out that an example of a hereditary class with polynomial local complexity had already been identified in [4] a couple of years earlier: triangle-free graphs (the lower bound on the local complexity given there was sublinear). It was speculated in [15]

[1] Give to every vertex the adjacency matrix of the graph.

[2] Note that it is easy to show that for any minor-closed class $\mathcal{C}$, the complement of $\mathcal{C}$ has local complexity $O(\log n)$, using Robertson and Seymour's Graph Minor Theorem [21].

that any class of geometric intersection graphs has small local complexity, as such classes are both hereditary and well-structured.

Results

In this paper we identify a key rigidity property in graph classes and use it to derive a number of *linear* lower bounds on the local complexity of graph classes defined using geometric or topological properties. These bounds are all best possible, up to $n^{o(1)}$ factors. So our main result is that for a number of classical hereditary graph classes studied in structural graph theory, topological graph theory, and graph drawing, the local complexity is $\Theta(n)$. These are the first non-trivial examples of hereditary classes (some of our examples are even monotone) with linear local complexity. Interestingly, all the classes we consider are very close to the class of planar graphs (which is known to have local complexity $\Theta(\log n)$ [9, 6]): most of these classes are either subclasses or superclasses of planar graphs. Given the earlier results on graphs of bounded treewidth [12] and planar graphs, it is natural to try to understand which sparse graph classes have (poly)logarithmic local complexity. It would have been tempting to conjecture that any (monotone or hereditary) graph class of *bounded expansion* (in the sense of Nešetřil and Ossona de Mendez [20]) has polylogarithmic local complexity, but our results show that this is false, even for very simple monotone classes of linear expansion.

We first show that every class of graphs that contains at most $2^{f(n)}$ unlabeled graphs of size n has local complexity $f(n) + O(\log n)$. This implies all the upper bounds we obtain in this paper, as the classes of graphs we consider usually contain $2^{O(n)}$ or $2^{O(n \log n)}$ unlabeled graphs of size n.

We then turn to lower bounds. Using rigidity properties in the classes we consider, we give a $\Omega(n)$ bound on the local complexity of *penny graphs* (contact graphs of unit-disks in the plane), *unit-distance graphs* (graphs that admit an embedding in $\mathbb{R}^2$ where adjacent vertices are exactly the vertices at Euclidean distance 1), and (induced) subgraphs of the square grid. We then consider *1-planar graphs*, which are graphs admitting a planar drawing in which each edge is crossed by at most one edge. This superclass of planar graphs shares many similarities with them, but we nevertheless prove that it has local complexity $\Theta(n)$ (while planar graphs have local complexity $\Theta(\log n)$).

Next, we consider *unit-square graphs* (intersection graphs of unit-squares in the plane). We obtain a linear lower bound on the local complexity of triangle-free unit-square graphs (which are planar) and of unit-square graphs in general. Finally, we consider *unit-disk graphs* (intersection of unit-disks in the plane), which are widely used in distributed computing as a model of wireless communication networks. For this class we reuse some key ideas introduced in the unit-square case, but as unit-disk graphs are much less rigid we need to introduce a number of new tools, which might be of independent interest in the study of rigidity in geometric graph classes. In particular we answer questions such as: what is asymptotically the minimum number of vertices in a unit-disk graph G such that in any unit-disk embedding of G, two given vertices u and v are at Euclidean distance at least n and at most $n+1$? Or at distance at least n and at most $n+\varepsilon$, for $\varepsilon \ll n$? Using our constructions we obtain a lower bound of $\Omega(n^{1-\delta})$ (for every $\delta > 0$) on the local complexity of unit-disk graphs. As there are at most $2^{O(n \log n)}$ unlabelled unit-disk graphs on n vertices [18], our first result implies that the local complexity of unit-disk graphs is $O(n \log n)$, so our results are nearly tight.

Techniques

All our lower bounds are inspired by the set-disjointness problem in non-deterministic communication complexity. This approach was already used in earlier work in local certification, in order to provide lower bounds on the local complexity of computing the diameter [3], or for certifying non-3-colorability [13]. Here the main challenge is to translate the technique into geometric constraints. The key point of the set-disjointness problem is informally the following: let $A, B \subseteq \{1, \dots, N\}$ be the input of some kind of "two-party system" that must decide whether A and B are disjoint, given that one party knows A and the other knows B; then at least N bits of shared (or exchanged) information are necessary for them to decide correctly. Otherwise, there are fewer bit combinations than the 2^N entries of the form $(A, \overline{A})$, hence the two parties can be fooled to accept a negative instance built from two particular positive instances sharing the same bit combination. In the setting of non-deterministic communication complexity, the two parties are Alice and Bob; in our setting, the two parties will be two subsets of vertices covering the graph and with small intersection (the intersection must be a small cutset of the whole graph): in the following, we refer to those two connected subsets of vertices as respectively the "left" part and the "right" part of the graph. The "shared" bits of information will be the certificates given to their intersection (and to its neighborhood). To express the sets A, B and their disjointness, the left (resp. right) part of the graph will be equipped with a path P_A (resp. P_B) of length $\Omega(N)$, such that P_A and P_B only intersect in their endpoints.[3] The crucial rigidity property which we will require is that in any embedding of G as a geometric graph from some class $\mathcal{C}$, the two paths P_A and P_B will be very close, in the sense that if $P_A = a_1, \dots, a_\ell$ and $P_B = b_1, \dots, b_\ell$, then a_i is close to b_i for any $1 \leq i \leq \ell$. Using this property, we will attach some gadgets to the vertices of the path P_A (resp. P_B) depending on A (resp. B), in such a way that the resulting graph lies in the class $\mathcal{C}$ if and only if A and B are disjoint. As there is little connectivity between the left and the right part, the endpoints of the paths will have to contain very long certificates in order to decide whether A and B are disjoint, hence whether $G \in \mathcal{C}$ or not.

We present the results in increasing order of difficulty. Subgraphs or induced subgraphs of infinite graphs such as grids are perfectly rigid in some sense, with some graphs having unique embeddings up to symmetry. Unit-square graphs are much less rigid but we can use nice properties of the ℓ_∞-distance and the uniqueness of embeddings of 3-connected planar graphs. We conclude with unit-disk graphs, which is the least rigid class we consider. The Euclidean distance misses most of the properties enjoyed by the ℓ_∞-distance and we must work much harder to obtain the desired rigidity property.

Outline

We start with some preliminaries on graph classes and local certification in Section 2. We prove our general upper bound result in Section 3. Section 4 introduces the notion of *disjointness-expressing* class of graphs, highlighting the key properties needed to derive our local certification lower bounds. We deduce in Section 5 linear lower bounds on the local complexity of subgraphs of the grid, penny graphs, and 1-planar graphs. Section 6 is devoted for the linear lower bound on the local complexity of unit-square graphs, while Section 7 contains our main result, a quasi-linear lower bound on the local complexity of unit-disk graphs. We conclude in Section 8 with a number of questions and open problems.

Due to the limit on the number of pages of the submission, most of the proofs have been omitted in this version. They are available in the full version of the paper [5].

[3] We note here that the proof for 1-planar graphs diverges from this approach, but it is the only one.

2 Preliminaries

In this paper logarithms are binary, and graphs are assumed to be simple, loopless, undirected, and connected. The *length* of a path P, denoted by $|P|$, is the number of edges of P. The *distance* between two vertices u and v in a graph G, denoted by $d_G(u, v)$ is the minimum length of a path between u and v. The *neighborhood* of a vertex v in a graph G, denoted by $N_G(v)$ (or $N(v)$ if G is clear from the context), is the set of vertices at distance exactly 1 from v. The *closed neighborhood* of v, denoted by $N_G[v] := \{v\} \cup N_G(v)$, is the set of vertices at distance at most 1 from v. For a set S of vertices of G, we define $N_G[S] := \bigcup_{v\in S} N_G[v]$.

2.1 Local certification

The vertices of any n-vertex graph G are assumed to be assigned distinct (but otherwise arbitrary) identifiers $(\mathrm{id}(v))_{v\in V(G)}$ from $\{1, \ldots, \mathrm{poly}(n)\}$. When we refer to a subgraph H of a graph G, we implicitly refer to the corresponding labeled subgraph of G. Note that the identifiers of each of the vertices of G can be stored using $O(\log n)$ bits, where log denotes the binary logarithm. We follow the terminology introduced by Göös and Suomela [13].

Proofs and provers

A *proof* for a graph G is a function $P : V(G) \to \{0, 1\}^*$ (as G is a labeled graph, the proof P is allowed to depend on the identifiers of the vertices of G). The binary words $P(v)$ are called *certificates*. The *size* of P is the maximum size of a certificate $P(v)$, for $v \in V(G)$. A *prover* for a graph class $\mathcal{G}$ is a function that maps every $G \in \mathcal{G}$ to a proof for G.

Local verifiers

A *verifier* $\mathcal{A}$ is a function that takes a graph G, a proof P for G, and a vertex $v \in V(G)$ as inputs, and outputs an element of $\{0, 1\}$. We say that v *accepts* the instance if $\mathcal{A}(G, P, v) = 1$ and that v *rejects* the instance if $\mathcal{A}(G, P, v) = 0$.

Consider a graph G, a proof P for G, and a vertex $v \in V(G)$. We denote by $G[v]$ the subgraph of G induced by $N[v]$, the closed neighborhood of v, and similarly we denote by $P[v]$ the restriction of P to $N[v]$.

A verifier $\mathcal{A}$ is *local* if for any $v \in G$, the output of v only depends on its identifier and $P[v]$.

Note that our lower bounds hold in the stronger model of *locally checkable proofs* of Göös and Suomela [13], where in addition the output of v is allowed to depend on $G[v]$, that is $\mathcal{A}(G, P, v) = \mathcal{A}(G[v], P[v], v)$ for any vertex v of G.

Proof labeling schemes

A *proof labeling scheme* for a graph class $\mathcal{G}$ is a prover-verifier pair $(\mathcal{P}, \mathcal{A})$ where $\mathcal{A}$ is local, with the following properties.

Completeness: If $G \in \mathcal{G}$, then $P := \mathcal{P}(G)$ is a proof for G such that for any vertex $v \in V(G)$, $\mathcal{A}(G, P, v) = 1$.

Soundness: If $G \notin \mathcal{G}$, then for every proof P' for G, there exists a vertex $v \in V(G)$ such that $\mathcal{A}(G, P', v) = 0$.

In other words, upon looking at its closed neighborhood (labeled by the identifiers and certificates), the local verifier of each vertex of a graph $G \in \mathcal{G}$ accepts the instance, while if $G \notin \mathcal{G}$, for every possible choice of certificates, the local verifier of at least one vertex rejects the instance.

The *complexity* of the proof labeling scheme is the maximum size of a proof $P = \mathcal{P}(G)$ for an n-vertex graph $G \in \mathcal{G}$, and the *local complexity* of $\mathcal{G}$ is the minimum complexity of a proof labeling scheme for $\mathcal{G}$. If we say that the complexity is $O(f(n))$, for some function f, the $O(\cdot)$ notation refers to $n \to \infty$. See [7, 13] for more details on proof labeling schemes and local certification in general.

2.2 Geometric graph classes

In this section we collect some useful properties that are shared by most of the graph classes we will investigate in the paper.

A *unit-disk graph* (respectively *unit-square graph*) is the intersection graph of unit-disks (respectively unit-squares) in the plane. That is, G is a unit-disk graph if every vertex of G can be mapped to a unit-disk in the plane so that two vertices are adjacent if and only if the corresponding disks intersect, and similarly for squares. A *penny graph* is the contact graph of unit-disks in the plane, i.e., in the definition of unit-disk graphs above we additionally require the disks to be pairwise interior-disjoint. A *unit-distance* graph is a graph whose vertices are points in the plane, where two points are adjacent if and only if their Euclidean distance is equal to 1. Unit-distance graphs clearly form a superclass of penny graphs.

A *drawing* of a graph G in the plane is a mapping from the vertices of G to distinct points in the plane and from the edges of G to Jordan curves, such that for each edge uv in G, the curve associated to uv joins the images of u and v and does not contain the image of any other vertex of G. A graph is *planar* if it has a drawing in the plane with no edge crossings (such a drawing will also be called a *planar graph drawing* in the remainder). Every planar graph drawing of a graph G gives a clockwise cyclic ordering of the neighbors around each vertex of G. We say that that two planar graph drawings of G are *equivalent* if the corresponding cyclic orderings are the same. A *planar graph embedding* of a graph G is an equivalence class of planar graph drawings of G. Given a planar graph embedding of a graph G, all the corresponding (equivalent) planar drawings of G have the same set of faces (but different choices of outerface yield different planar drawings).

A graph is *1-planar* if it has a drawing in the plane such that for each edge e of G, there is at most one edge e' of G distinct from e such that the interior of the curve associated to e intersects the interior of the curve associated to e'.

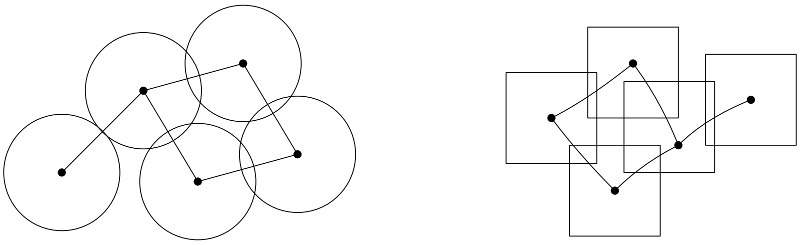


Figure 1 Triangle-free intersection graphs of unit-disks and unit-squares in the plane, and the associated planar graph embeddings.

The following well-known proposition will be useful (see Figure 1 for an illustration).

▶ **Proposition 2.1.** *Consider a family of unit-disks or a family of unit-squares in the plane, and assume that the intersection graph G of the family is connected and triangle-free. Then G is planar, and moreover each representation of G as such an intersection graph of unit-disks or unit-squares in the plane gives rise to a planar graph embedding of G in a natural way (see for instance Figure 1). Furthermore, the representation of G as an intersection graph (of unit-disks or unit-squares) and the resulting planar graph embedding are equivalent, in the sense that the clockwise cyclic ordering of the neighbors around each vertex is the same.*

We will often need to argue that some planar graphs have unique planar embeddings. The following classical result of Whitney will be crucial.

▶ **Theorem 2.2** ([22]). *If a planar graph G is 3-connected (or can be obtained from a 3-connected simple graph by subdividing some edges), then it has a unique planar graph embedding, up to the reversal of all cyclic orderings of neighbors around the vertices.*

We note that the reversal of all cyclic orderings in the statement of Theorem 2.2 corresponds to a reflection of the corresponding planar drawings.

3 Linear upper bounds for tiny classes

Given a class of graphs $\mathcal{C}$ and a positive integer n, let $\mathcal{C}_n$ be the set of all unlabeled graphs of $\mathcal{C}$ having exactly n vertices (i.e., we consider graphs up to isomorphism).

If there is a constant $c > 0$ such that for every positive integer n, $|\mathcal{C}_n| \leq c^n$, then the class $\mathcal{C}$ is said to be *tiny*. This is the case for all proper minor-closed classes (for instance planar graphs). On the other hand, unit-interval graphs and unit-disk graphs do not form tiny classes as proved in [18]. The local complexity and the number of unlabeled graphs in a class are related by the following simple result.

▶ **Theorem 3.1** ([5]). *Any class $\mathcal{C}$ of connected graphs has local complexity at most $\log(|\mathcal{C}_n|) + O(\log n)$. In particular if $\mathcal{C}$ is a tiny class, then the local complexity is $O(n)$.*

Proof (sketch). The prover gives each vertex v the same description of G (as an unlabelled graph, so using $\log(|\mathcal{C}_n|)$ bits), together with the name of the image of v in this description, and the number n of vertices of G. Each vertex checks that its neighborhood is consistent with its image in the description of G, and if so the graph under consideration must have a locally bijective homomorphism to G. The vertices then check that the number of vertices of the graph is indeed $n = |V(G)|$, which implies that this locally bijective homomorphism is an isomorphism to G, as desired. ◀

As a consequence, we immediately obtain the following.

▶ **Corollary 3.2.** *The following classes have local complexity $O(n)$: the class of all (induced) subgraphs of the square grid, penny graphs, 1-planar graphs, triangle-free unit-square graphs, and triangle-free unit-disk graphs.*

The next result directly follows from a bound of order $2^{O(n \log n)}$ on the number of unit-square graphs and unit-disk graphs [18], and on the number of unit-distance graphs [1].

▶ **Corollary 3.3.** *The classes of unit-distance graphs, unit-square graphs, and unit-disk graphs have local complexity $O(n \log n)$.*

The remainder of the paper consists in proving lower bounds of order $\Omega(n)$ (or $\Omega(n^{1-\delta})$, for any $\delta > 0$), for all the classes mentioned in Corollaries 3.2 and 3.3, except triangle-free unit-disk graphs (our quasi-linear lower bound only applies to unit-disk graphs).

4 Disjointness-expressing graph classes

In this section we describe the framework relating the disjointness problem to proof labeling schemes. Our main source of inspiration is [13], where a lower bound on the local complexity of non-3-colorability is proved using a similar approach, and [3] where an explicit reduction to the non-deterministic communication complexity of the disjointness problem is used.

Here we adapt the disjointness problem to fit in our local certification setting. A class $\mathcal{C}$ of graphs is said to be (s, κ)-*disjointness-expressing* if for some constant $\alpha > 0$, for every positive integer N and every $X \subseteq \{1, \dots, N\}$, one can define graphs $L(X)$ (referred to as the "left part") and $R(X)$ ("right part"), each containing a labeled set S of special vertices such that for every $A, B \subseteq \{1, \dots, N\}$ the following holds:

1. the graph $g(L(A), R(B))$ obtained by identifying vertices of S in $L(A)$ to the corresponding vertices of S in $R(B)$ is connected and has at most $\alpha N^{1/\kappa}$ vertices;
2. the subgraph of $g(L(A), R(B))$ induced by the closed neighborhood $N_{g(L(A),R(B))}[S]$ of S is independent[4] of the choice of A and B and has at most s vertices; and
3. $g(L(A), R(B))$ belongs to $\mathcal{C}$ if and only if $A \cap B = \emptyset$.

The idea is that S is a small cutset between vertices of $L(A)$, having information on A, and vertices of $R(B)$, having information on B. Deciding whether the graph $g(L(A), R(B))$ belongs to $\mathcal{C}$ amounts to deciding whether A and B are disjoint, which requires N bits of information even in a non-deterministic setting, thus the small cutset at the frontier between $L(A)$ and $R(B)$ must receive long certificates. Otherwise, there are fewer bit combinations at the frontier than the 2^N entries of the form $(A, \overline{A})$, hence the vertices can be fooled to accept a negative instance built from two particular positive instances sharing the same bit combination.

The role of s and κ is explained by the result below.

▶ **Theorem 4.1** ([5]). *Let $\mathcal{C}$ be a (s, κ)-disjointness-expressing class of graphs. Then any proof labeling scheme for the class $\mathcal{C}$ has complexity $\Omega\left(\frac{n^\kappa}{s}\right)$. In particular if s is a constant and $\kappa = 1$, the complexity is $\Omega(n)$.*

Proof (sketch). Let $(\mathcal{P}, \mathcal{A})$ be a proof labeling scheme for the class $\mathcal{C}$ and let $p \colon \mathbb{N} \to \mathbb{N}$ be its complexity. For every $A \subseteq \{1, \dots, N\}$, let $G_A = g(L(A), R(\overline{A}))$. Clearly $G_A \in \mathcal{C}$ so the verifier $\mathcal{A}$ accepts the proof $P_A = P(G_A)$ on every vertex of G_A. Let n denote the maximum order of G_A for $A \subseteq \{1, \dots, N\}$.

There are 2^N choices for the set A. On the other hand, in G_A there are at most $2^{s \cdot p(n)}$ different ways to assign certificates to the vertices of $N[S]$. By the Pigeonhole Principle, if $2^N > 2^{sp(n)}$ there are two sets $A, A' \subseteq \{1, \dots, N\}$ such that the proofs P_A and $P_{A'}$ coincide on the subgraph of G_A and $G_{A'}$ induced by $N[S]$. Since $A \neq A'$, we may assume without loss of generality that $A \cap \overline{A'} \neq \emptyset$. So the graph $G = g(L(A), R(\overline{A'}))$ does not belong to $\mathcal{C}$. We now consider a proof P for G defined as follows: if $v \in V(L(A))$ then $P(v) := P_A(v)$ and if $v \in V(R(\overline{A'}))$ then $P(v) := P_{A'}(v)$. The verifier $\mathcal{A}$ will accept P on every vertex of G, contradiction. Therefore $2^N \leq 2^{s \cdot p(n)}$. Recall that $n \leq \alpha N^{1/\kappa}$, by the definition of disjointness-expressibility. Hence $p(n) = \Omega(n^\kappa / s)$, as claimed. ◀

[4] i.e., for every $A, A', B, B' \subseteq \{1, \dots, N\}$ there is an isomorphism from $g(L(A), R(B))[N_{g(L(A),R(B))}[S]]$ to $g(L(A'), R(B'))[N_{g(L(A'),R(B'))}[S]]$ that is the identity on S.

5 Linear lower bounds in rigid classes

In this section we obtain linear lower bounds on the local complexity of several graph classes using the framework described in Section 4. We only sketch the argument in the case of penny graphs.

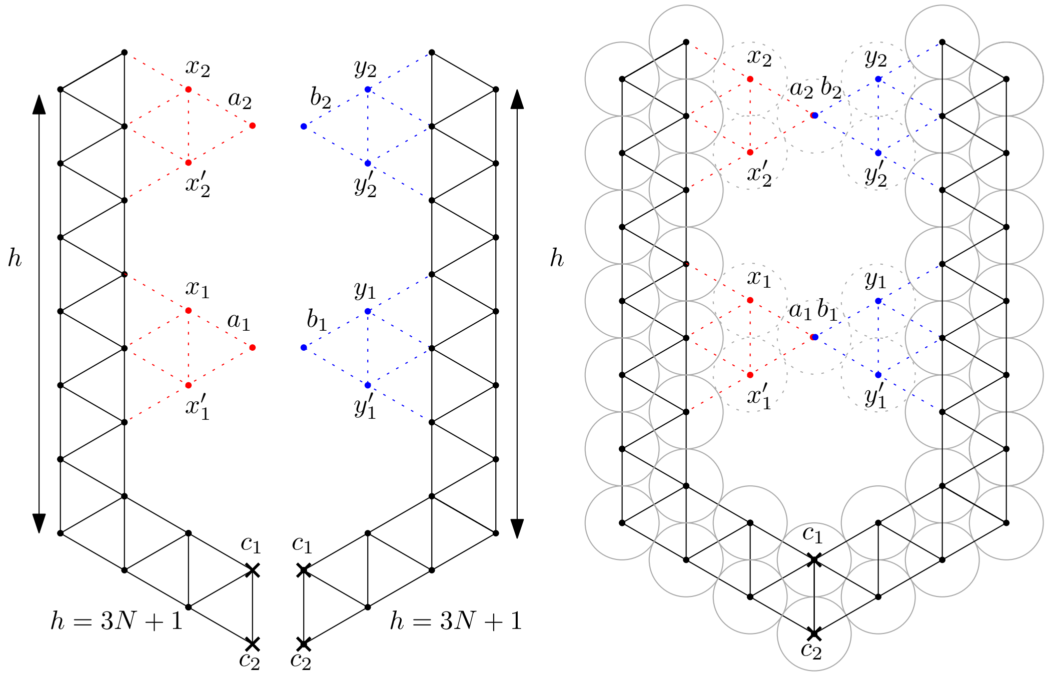


Figure 2 Construction of L, R and g for penny graphs in the case where $N = 2$, with $A, B \subseteq \{1, \ldots, N\}$. Color red highlights vertices and edges that depend on the choice of A, and color blue highlights vertices and edges that depend on the choice of B.

▶ **Theorem 5.1** ([5]). *The class of penny graphs is $(6, 1)$-disjointness-expressing.*

Proof (sketch). The proof is illustrated in Figure 2. The graph $g(L(A), R(B))$ (on the right) is obtained by identifying each c_i $(i = 1, 2)$ in $L(A)$ with the corresponding vertex in $R(B)$. The graphs $L(A)$ and $R(B)$ are depicted on the left. Vertices a_j, x_j, x'_j are added to the graph $L(A)$ if and only if $j \in A$, and similarly vertices b_j, y_j, y'_j are added to the graph $R(B)$ if and only if $j \in B$. The crucial properties of the construction are that (1) the graph without the added gadgets has a unique embedding as a penny graph, and (2) there is a small cut separating the left and right parts ($\{c_1, c_2\}$, which is far from the gadgets on both sides), and (3) if two gadgets a_i, x_i, x'_i and b_j, y_j, y'_j are added with $i = j$, then the graph is not a penny graph. The last item follows from the fact that a_i and b_j would be mapped to the same point in the plane, and thus a_i would also need to be adjacent to y_j and y'_j in the graph (which they are not). ◀

From Theorems 5.1 and 4.1, together with Corollary 3.2, we immediately deduce the following.

▶ **Theorem 5.2.** *The local complexity of the class of penny graphs is $\Theta(n)$.*

Using our framework we obtain the following results (whose proofs are available in appendix).

▶ **Theorem 5.3.** *The class of unit-distance graphs is* $(6, 1)$*-disjointness-expressing.*

▶ **Theorem 5.4** ([5])**.** *The class of subgraphs of the square grid is* $(6, 1)$*-disjointness-expressing.*

▶ **Theorem 5.5** ([5])**.** *The class of 1-planar graphs is* $(20, 1)$*-disjointness-expressing.*

We immediately deduce the following.

▶ **Theorem 5.6.** *The classes of subgraphs of the square grid, unit-distance graphs and 1-planar graphs all have local complexity* $\Theta(n)$*.*

6 Unit-square graphs

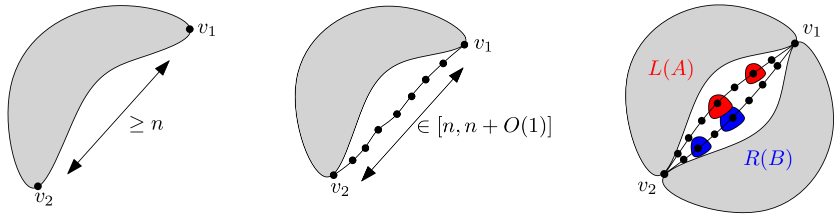


Figure 3 A sketch of the general approach to prove Theorems 6.1 and 7.1.

The graph we constructed in the previous section had some perfect rigidity properties: if the images of a constant number of vertices in the plane were fixed, then the whole graph had at most one embedding in the plane. This does not hold in unit-square graphs, but for our framework it is enough to make sure that once a constant number of vertices are fixed, each vertex of the graph can only be mapped to a small fixed region in any embedding. More precisely, we construct for any n a unit-square graph with $O(n)$ vertices with two specific vertices v_1, v_2 that are at ℓ_∞-distance at least n in any embedding. We add a shortest path connecting v_1 and v_2, so that the resulting graph is still a unit-square graph with $O(n)$ vertices, and the ℓ_∞-distance between v_1 and v_2 is at least n and at most $n + O(1)$ in any embedding. This is illustrated in Figure 3 above, where the path between v_1 and v_2 is close from being mapped to the line segment between the image of v_1 and the image of v_2. This path is then used as an interface to add gadgets expressing any set A for part $L(A)$ and any set B for part $R(B)$, very much as in the proof of Theorem 5.1. The difficulty lies in proving this approximate rigidity property, which follows from the rigidity of the ℓ_∞ norm.

▶ **Theorem 6.1** ([5])**.** *The classes of triangle-free unit-square graphs and unit-square graphs are both* $(6, 1)$*-disjointness-expressing.*

Using Theorem 4.1, together with Corollaries 3.2 and 3.3, we immediately deduce the following.

▶ **Theorem 6.2.** *The local complexity of the class of triangle-free unit-square graphs is* $\Theta(n)$*, and the local complexity of the class of unit-square graphs is* $\Omega(n)$ *and* $O(n \log n)$*.*

We note that the proof approach of Theorem 6.1 naturally extends to higher dimension.

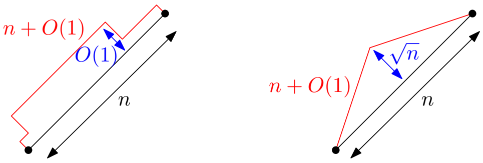


Figure 4 The difference between the ℓ_∞-distance (left) and the ℓ_2-distance (right).

7 Unit-disk graphs

We would like to prove a variant of Theorem 6.1 for unit-disk graphs, but there are two major obstacles. The first is that there does not seem to be a simple unit-disk graph with $O(n)$ vertices with two specified vertices that are at Euclidean distance at least n in any unit-disk embedding. Our construction of such a graph will be significantly more involved (and thus the number of vertices will be only upper bounded by $O(n^{1+\varepsilon})$ for any $\varepsilon > 0$, rather than $O(n)$). The second obstacle comes from Pythagoras' theorem: In the unit-square case, if we consider a path P of length $n + O(1)$ between two vertices u, v embedded in the plane such that their x- and y-coordinates both differ by exactly n, then in any unit-square embedding of P, the vertices of P deviate by at most a constant from the line segment $[u, v]$ between u and v. This is what we used in the proof of Theorem 6.1 to make sure that $L(A)$ and $R(B)$ are so close that the i-th gadget cannot exist both on $L(A)$ and $R(B)$ simultaneously when $i \in A \cap B$. However, as illustrated in Figure 4, Pythagoras' theorem implies that in the Euclidean case, when the Euclidean distance between u and v is equal to n, the vertices of P can deviate by $\Theta(\sqrt{n})$ from the line segment $[u, v]$, which is too much for our purpose (we need a constant deviation). So we need different ideas to make sure the gadgets are embedded sufficiently close to each other.

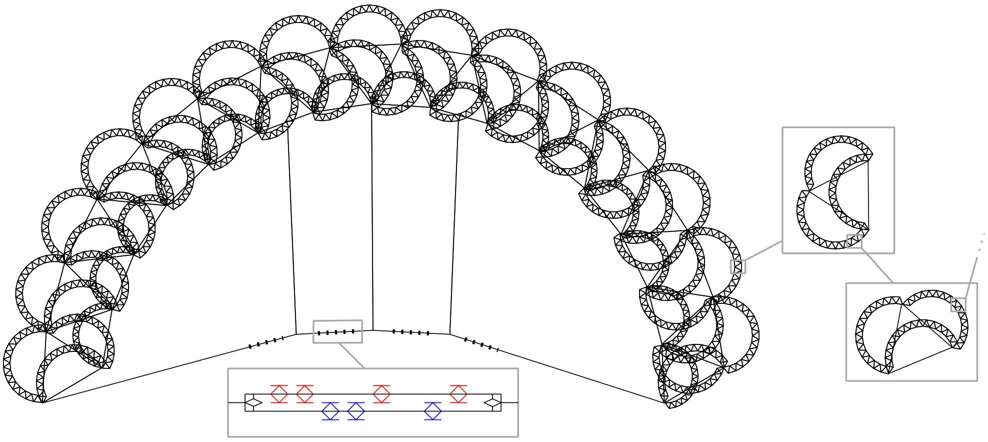


Figure 5 A summary of the construction used in the proof of Theorem 7.1.

The main result of this paper is the following.

▶ **Theorem 7.1** ([5]). *For any $\delta > 0$, the class of unit-disk graphs is $(O(\log n), 1 - \delta)$-disjointness-expressing.*

Proof (sketch). A summary of our construction is depicted in Figure 5. The first component of the construction is the arch-shaped part, which is defined recursively and has the property that its endpoints lie at Euclidean distance $\Omega(n)$ in any unit-disk embedding, while the graph only contains $O(n^{1+\varepsilon})$ vertices (for any $\varepsilon > 0$). We then add a shortest possible path P between the endpoints of the arch (with the condition that the resulting graph is still a unit-disk graph) and we would like to argue that the path P is *tight*, in the sense that in any embedding, all the unit-disks corresponding to the vertices of P lie at distance $O(1)$ from the line segment between the endpoints of P. For this we need to add a number of paths connecting P to the arch to make P even tighter. These paths delimit subpaths of P and the construction forces *at least one of* those subpaths to be tight. Since we do not know which subpath will be tight, we add gadgets, called *decorated corridors*, along all these subpaths. When a subpath is tight, the corresponding corridor is sufficiently narrow so that gadgets of $L(A)$ and $R(B)$ along the corridor can emulate the disjointness problem between A and B, as in the proof of Theorem 5.1. There is a gadget of $L(A)$ at position j of every corridor if and only if $j \in A$ and similarly for $R(B)$, and the gadgets of $L(A)$ and $R(B)$ intersect at position j of some corridor if and only if $j \in A \cap B$. Since these gadgets are not adjacent in the graph, this shows that the graph is a unit-disk graph if and only if $A \cap B = \emptyset$, as desired. ◀

Using Theorem 4.1, together with Corollary 3.3, we immediately deduce the following.

▶ **Theorem 7.2.** *The local complexity of the class of unit-disk graphs is $O(n \log n)$ and $\Omega(n^{1-\delta})$ for any $\delta > 0$.*

8 Open problems

In this paper we have obtained a number of optimal (or close to optimal) results on the local complexity of geometric graph classes. Our proofs are based on a new notion of rigidity. It is natural to ask which other graph classes enjoy similar properties. A natural candidate is the class of segment graphs (intersection graphs of line segments in the plane), which have several properties in common with unit-disk graphs. In particular the recognition problems for these classes are complete for the existential theory of the reals, and the minimum bit size for representing an embedding of some of these graphs in the plane is at least exponential in their number of vertices. We believe that the local complexity of segment graphs (and that of the more general class of string graphs) is at least polynomial in their number of vertices. More generally, is it true that all classes of graphs for which the recognition problem is hard for the existential theory of the reals have polynomial local complexity?

It might also be interesting to investigate the smaller class of *circle graphs* (intersection graphs of chords of a circle). The authors of [14] proved that the closely related class of permutation graphs has logarithmic local complexity. It is quite possible that the same holds for circle graphs. See [15] for results on interactive proof labeling schemes for this class and related classes.

We proved that 1-planar graphs have local complexity $\Theta(n)$. What can we say about the local complexity of other graph classes defined with constrained on their drawings in the plane? For instance is it true that for every $k \geq 2$, the local complexity of the class of graphs with queue number at most k is polynomial? What about graphs with stack number at most k?

We have given the first example of non-trivial hereditary (and even monotone) classes of local complexity $\Omega(n)$. Can this be improved? Are there hereditary (or even monotone) classes of local complexity $\Omega(n^c)$ for $c > 1$?

References

1 Noga Alon and Andrey Kupavskii. Two notions of unit distance graphs. *Journal of Combinatorial Theory, Series A*, 125:1–17, 2014.

2 Nicolas Bousquet, Laurent Feuilloley, and Théo Pierron. Local certification of graph decompositions and applications to minor-free classes. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France*, volume 217 of *LIPIcs*, pages 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.OPODIS.2021.22.

3 Keren Censor-Hillel, Ami Paz, and Mor Perry. Approximate proof-labeling schemes. *Theoretical Computer Science*, 811:112–124, 2020. doi:10.1016/j.tcs.2018.08.020.

4 Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. Trade-offs in distributed interactive proofs. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 13:1–13:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.DISC.2019.13.

5 Oscar Defrain, Louis Esperet, Aurélie Lagoutte, Pat Morin, and Jean-Florent Raymond. Local certification of geometric graph classes. *CoRR*, abs/2311.16953, 2023. arXiv:2311.16953, doi:10.48550/arXiv.2311.16953.

6 Louis Esperet and Benjamin Lévêque. Local certification of graphs on surfaces. *Theoretical Computer Science*, 909:68–75, 2022. doi:10.1016/j.tcs.2022.01.023.

7 Laurent Feuilloley. Introduction to local certification. *Discrete Mathematics & Theoretical Computer Science*, 23(3), 2021. arXiv:1910.12747.

8 Laurent Feuilloley, Nicolas Bousquet, and Théo Pierron. What can be certified compactly? Compact local certification of MSO properties in tree-like graphs. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 131–140. ACM, 2022. doi:10.1145/3519270.3538416.

9 Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Compact distributed certification of planar graphs. *Algorithmica*, 83(7):2215–2244, 2021. doi:10.1007/s00453-021-00823-w.

10 Laurent Feuilloley, Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, Éric Rémila, and Ioan Todinca. Local certification of graphs with bounded genus. *Discrete Applied Mathematics*, 325:9–36, 2023. doi:10.1016/j.dam.2022.10.004.

11 Pierre Fraigniaud, Frédéric Mazoit, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. Distributed certification for classes of dense graphs. *arXiv preprint*, 2023. arXiv:2307.14292.

12 Pierre Fraigniaud, Pedro Montealegre, Ivan Rapaport, and Ioan Todinca. A meta-theorem for distributed certification. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 116–134. Springer, 2022. doi:10.1007/978-3-031-09993-9_7.

13 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016. doi:10.4086/toc.2016.v012a019.

14 Benjamin Jauregui, Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Local certification of some geometric intersection graph classes. *CoRR*, abs/2309.04789, 2023. doi:10.48550/arXiv.2309.04789.

15 Benjamin Jauregui, Pedro Montealegre, and Ivan Rapaport. Distributed interactive proofs for the recognition of some geometric intersection graph classes. In Merav Parter, editor, *Structural Information and Communication Complexity - 29th International Colloquium, SIROCCO 2022, Paderborn, Germany, June 27-29, 2022, Proceedings*, volume 13298 of *Lecture Notes in Computer Science*, pages 212–233. Springer, 2022. doi:10.1007/978-3-031-09993-9_12.

16 Gillat Kol, Rotem Oshman, and Raghuvansh R. Saxena. Interactive distributed proofs. 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 255–264. ACM, 2018. URL: https://dl.acm.org/citation.cfm?id=3212771.

17 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010. doi:10.1007/s00446-010-0095-3.

18 Colin McDiarmid and Tobias Müller. The number of disk graphs. *European Journal of Combinatorics*, 35:413–431, 2014. Selected Papers of EuroComb'11. doi:10.1016/j.ejc.2013.06.037.

19 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. 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 1096–115. SIAM, 2020. doi:10.1137/1.9781611975994.67.

20 Jaroslav Nešetřil and Patrice Ossona De Mendez. *Sparsity: graphs, structures, and algorithms*, volume 28. Springer Science & Business Media, 2012.

21 Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

22 Hassler Whitney. 2-isomorphic graphs. *American Journal of Mathematics*, 55(1):245–254, 1933.

Efficient Computation in Congested Anonymous Dynamic Networks*

Giuseppe A. Di Luna[1]
DIAG, Sapienza University of Rome, Italy

Giovanni Viglietta[1]
Department of Computer Science and Engineering, University of Aizu, Japan

Abstract

An *anonymous dynamic network* is a network of indistinguishable processes whose communication links may appear or disappear unpredictably over time. Previous research has shown that deterministically computing an arbitrary function of a multiset of input values given to these processes takes only a linear number of communication rounds (Di Luna–Viglietta, FOCS 2022).

However, fast algorithms for anonymous dynamic networks rely on the construction and transmission of large data structures called *history trees*, whose size is polynomial in the number of processes. This approach is unfeasible if the network is *congested*, and only messages of logarithmic size can be sent through its links. Observe that sending a large message piece by piece over several rounds is not in itself a solution, due to the anonymity of the processes combined with the dynamic nature of the network. Moreover, it is known that certain basic tasks such as all-to-all token dissemination (by means of single-token forwarding) require $\Omega(n^2/\log n)$ rounds in congested networks (Dutta et al., SODA 2013).

In this work, we develop a series of practical and efficient techniques that make it possible to use history trees in congested anonymous dynamic networks. Among other applications, we show how to compute arbitrary functions in such networks in $O(n^3)$ communication rounds, greatly improving upon previous state-of-the-art algorithms for congested networks.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Computing methodologies → Distributed algorithms

Keywords and phrases anonymous dynamic network, congested network, history tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.49

Related Version *Full Version*: https://arxiv.org/abs/2301.07849

Funding Giuseppe A. Di Luna was Giovanni Viglietta was
Giuseppe A. Di Luna: partially supported by Sapienza Progetto Ateneo RM1221816C1760BF.
Giovanni Viglietta: partially supported by the JSPS KAKENHI grant 23K10985.

1 Introduction

Dynamic networks. In recent years, distributed computing has seen a remarkable increase in research on the algorithmic aspects of networks that constantly change their topology [7, 30, 32]. The study of these *dynamic networks* is motivated by technologies such as wireless sensors networks, software-defined networks, and networks of smart devices. Typically, the distributed system consists of n processes that communicate with each other in synchronous rounds. At each round, the network topology is rearranged arbitrarily, and communication links appear or disappear unpredictably.

[*] A preliminary version of this paper appeared as a Brief Announcement in the Proceedings of PODC 2023 [16].
[1] Both authors contributed equally to this research.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 49; pp. 49:1–49:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Anonymity and leadership. There are efficient algorithms for various tasks that work under the assumption that processes have *unique IDs* [6, 27, 28, 29, 32, 34]. However, unique IDs may not be available due to operational limitations [34] or to protect user privacy; for instance, assigning temporary random IDs to users of COVID-19 tracking apps was not sufficient to eliminate privacy concerns [40]. Systems where processes are indistinguishable are called *anonymous*.

It is known that many fundamental problems for anonymous networks cannot be solved without additional "symmetry-breaking" assumptions: A notable example is the *Counting problem*, i.e., determining the total number of processes n. The most typical symmetry-breaking choice is assuming the presence of a single distinguished process in the system, called *leader* [1, 2, 3, 4, 14, 19, 21, 23, 31, 39, 42]. A leader process may represent a base station in a sensor network, a super-node in a P2P network, etc.

Disconnected networks. Another common assumption is that the network is connected at every round [28, 34]. However, this assumption appears somewhat far-fetched when one considers the highly dynamic nature of some real-world networks, such as P2P networks of smart devices moving unpredictably. A weaker and more reasonable assumption is that the union of all the network's links across any T consecutive rounds induces a connected (multi)graph on the processes [25, 36]. Such a network is said to be *T-union-connected*, and $T \geq 1$ is its *dynamic disconnectivity* [17].

Congested networks. Almost all previous research on anonymous dynamic networks pertains to models that impose no limit on the size of messages exchanged by processes [11, 15, 21, 22, 23, 24, 26, 34, 36]. Unfortunately, in most mobile networks, sending small-size messages is not only desirable but also a necessity; for example, in sensor networks, short communication times significantly increase battery life. A more realistic model assumes the network to be "congested" and limits the size of every message to $O(\log n)$ bits, where n is the number of processes [38].[2]

General computation. A recent innovation in the study of anonymous dynamic networks with leaders was the introduction of *history trees* in [15], which led to an optimal deterministic solution to the *Generalized Counting problem*[3] in the non-congested network model. This problem is "complete" for a large class of functions called *multiset-based functions*, which in turn are the only computable functions in this model. The theory of history trees was extended in [17] to *leaderless* networks, providing optimal algorithms for the *Frequency problem*:[4] This problem is complete for the class of *frequency-based* functions, which are the only computable functions in leaderless systems. Thus, the computational landscape for the non-congested network model is fully understood, and optimal linear-time algorithms are known for anonymous dynamic systems with and without leaders. No previous research exists on the congested network model, except for a recent preprint that gives a Counting algorithm in $\widetilde{O}(n^{2T(1+\epsilon)+3})$ rounds for networks with leaders [25]. Note that its running time is exponential in the dynamic disconnectivity T and becomes $\widetilde{O}(n^{5+\epsilon})$ for connected networks.

[2] This $O(\log n)$ limit on message sizes does not imply that the processes have a-priori information about n. The size limit is not explicitly given to the processes, and it is up to the algorithm to automatically prevent larger messages from being sent.

[3] In the Generalized Counting problem, each process starts with a certain input, and the goal is to determine how many processes have each input. That is, each process has to compute the *multiset* of all inputs.

[4] In the Frequency problem, the goal is to determine the percentage of processes that have each input.

1.1 Contributions and Techniques

Contributions. In this paper, we provide a state-of-the-art general algorithmic technique for T-union-connected anonymous dynamic *congested* networks, with and without leaders. The resulting algorithms run in $O(Tn^3)$ rounds, where n is the (initially unknown) total number of processes.

In Section 4 we give a basic and slightly inefficient Counting algorithm that applies to a limited setting but already contains all of the key ideas of our technique. In Section 5 we sketch its correctness (the technical details of the proof are found in the arXiv version linked in the title page), and in Section 6 we discuss optimizations and extensions of the basic algorithm to several other settings. This includes the computation of all multiset-based functions. Section 7 concludes the paper with some directions for future research.

Technical background. Informally, a *history tree* is a way of representing the history of a network in the form of an infinite tree. Each node in a history tree represents a set of anonymous processes that are "indistinguishable" at a certain round, where two processes become "distinguishable" as soon as they receive different sets of messages (see Section 3).

The theory of history trees developed in [15, 17] yields optimal general algorithms for anonymous dynamic networks with and without leaders, assuming the network is not congested. The idea is that processes can work together to incrementally construct the history tree by repeatedly exchanging and merging together their respective "views" of it. Once they have a sufficiently large portion of the history tree (i.e., a number of "levels" proportional to n), each process can locally analyze its structure and perform arbitrary computations on the multiset of input values originally assigned to the processes.

Challenges. Unfortunately, implementing the above idea requires sending messages containing entire "views" of the history tree. The size of a view is $\Theta(n^3 \log n)$ bits in the worst case, and is therefore unsuitable for the congested network model [15]. There is a major difficulty in dealing with this problem deterministically, which stems from the lack of unique IDs combined with the dynamic nature of the network.

It is worth noting that the "naive" approach of breaking down large messages into smaller pieces to be sent in different rounds does not work. Indeed, it is not clear how the original message can then be reconstructed, because the pieces carry no IDs and a process' neighbors may change at every round. This may result in messages from different processes being mixed up and non-existent messages being reconstructed.

Methodology. Our main contribution is a general method that allows history trees to be transmitted reliably and deterministically between anonymous processes in a dynamic *congested* network with a leader. To overcome the fundamental issues outlined above, we devised a basic protocol combining different techniques, as well as a number of extensions, including leaderless ones. Although the techniques introduced in this paper are self-contained and do not rely on the results of [15], they effectively allow us to reduce the congested network model to the non-congested one, making it possible to apply the Counting algorithm in [15] as a "black box".

Firstly, we developed a method for dynamically assigning temporary (non-unique) IDs to processes; this method is an essential part of the history tree transmission algorithm. In fact, the nodes of our history trees are now augmented with IDs, meaning that each node represents the set of processes with a certain ID. When processes with equal IDs get disambiguated, they get new IDs.

The transmission of history trees occurs level by level, one edge at a time. Since the total ordering between IDs induces a total ordering on the history tree's edges, the processes can collectively transmit sets of edges with a method reminiscent of *Token Dissemination* [28].

Essentially, all processes participate in a series of broadcasts; the goal of each broadcast is to transmit the next "highest-value" edge to the whole network. The problem is that no upper bound on the *dynamic diameter* of the network is known, and so there is no way of knowing how many rounds it may take for all processes to receive the edge being broadcast.

We adopt a self-stabilizing approach to ensure that all messages are successfully broadcast. We give a communication protocol based on acknowledgments by the leader, where failure to broadcast a message alerts at least one process. Alerted processes start broadcasting error messages, which eventually cause a reset of the broadcast that caused the error. A mechanism that dynamically estimates the diameter of the network guarantees that no more than $O(\log n)$ resets are performed.

Finally, in order to achieve a cubic running time, we do not construct the history tree of the actual network, but a more compact history tree corresponding to a *virtual network*. The virtual network is carefully derived from the real one in such a way as to amortize the number of edges in the resulting history tree and further reduce the final worst-case running time by a factor of n.

2 Previous Work

Non-congested networks. The study of computation in anonymous dynamic networks has been mainly devoted to two fundamental problems: The *Counting problem* in networks with a leader [11, 12, 13, 21, 22, 23, 24, 26] and the *Average Consensus problem* in leaderless networks [5, 8, 9, 10, 24, 33, 35, 37, 41, 43]. This research thread produced a series of algorithms for these problems; the underlying technique used is a local averaging or mass-distribution method coupled with refined termination strategies.

A radically different technique based on history trees was recently used to optimally compute arbitrary multiset-based functions in $3n$ rounds in networks with a leader [15]. This approach was successfully extended to multi-leader, leaderless, and disconnected networks [17].

Congested networks. As for the congested model, the only paper that has ever studied deterministic algorithms for anonymous dynamic networks, to the best of our knowledge, is the recent preprint [25], which solves the Counting problem in $\widetilde{O}(n^{2T(1+\epsilon)+3})$ rounds. As usual, T is the dynamic disconnectivity of the network and ϵ is an arbitrarily small positive constant. By comparison, our main algorithm has a running time of $O(Tn^3)$ rounds; hence, its dependence on T is linear (as opposed to exponential) and, for connected networks (i.e., when $T = 1$), the improvement is a factor of $\Theta(n^{2+\epsilon}\log^k n)$.

Most previous research efforts on congested networks in the dynamic setting have focused on randomized algorithms or processes with unique IDs [18, 20, 28]. In this context, a problem similar to Counting is *Token Dissemination*, where each process starts with a token and the goal is for every process to collect all tokens. In connected networks, this problem is solved in $O(n^2)$ rounds by a simple *token-forwarding* algorithm (i.e., no manipulation is done on tokens other than storing, copying, and individually transmitting them) [28]. Interestingly, solving the Token Dissemination problem by token-forwarding algorithms requires at least $\Omega(n^2/\log n)$ rounds [18].

It is worth remarking that the randomized algorithm in [28] only solves the Counting problem approximately and assumes a-priori knowledge of an upper bound on n. Moreover, this algorithm only works with high probability. These are three key differences that make our contribution preferable. Furthermore, assuming processes to have unique IDs (or randomly generating unique IDs) as in [28] defeats the purpose of safeguarding user privacy, which is a motivation of our work.

3 Definitions and Fundamentals

Computation model. A *dynamic network* is modeled by an infinite sequence $\mathcal{G} = (G_t)_{t\geq 1}$, where $G_t = (V, E_t)$ is an undirected multigraph whose vertex set $V = \{p_1, p_2, \dots, p_n\}$ is a system of n *anonymous processes* and E_t is a multiset of edges representing *links* between processes. Hence, there may be multiple links between two processes, or even from a process to itself.[5]

If there is a constant $T \geq 1$ such that, for every $t \geq 1$, the multigraph $G_t^\star = \left(V, \bigcup_{i=t}^{t+T-1} E_i\right)$ is connected, the network is said to be T*-union-connected*, and the smallest such T is its *dynamic disconnectivity*.[6]

Each process p_i starts with an *input*, which is assigned to it at *round* 0. It also has an internal state, which is initially determined by its input. At each *round* $t \geq 1$, every process composes a message (as a function of its internal state) and sends it to its neighbors in G_t through all its incident links.[7] In the congested network model, only messages of $O(\log n)$ bits can be sent.[8] By the end of round t, each process reads all messages coming from its neighbors and updates its internal state according to a local algorithm $\mathcal{A}$. Note that $\mathcal{A}$ is the same for all processes, and is a deterministic function of the internal state and the multiset of messages received in the current round.

The input of each process also includes a *leader flag*. In Section 4, we will assume that the leader flag of exactly one process is set (this process is the *unique leader*); in Section 6, we will discuss the *leaderless* case, where none of the processes has the leader flag set.

A process may return an *output* at the end of a round, which must be a function of its current internal state. A process may also *terminate* execution after returning an output. An algorithm $\mathcal{A}$ solves the *Counting problem* if executing $\mathcal{A}$ at every round eventually causes all processes to simultaneously output n and terminate. The (worst-case) *running time* of $\mathcal{A}$, as a function of n, is the maximum number of rounds it takes for $\mathcal{A}$ to solve the problem, across all possible dynamic networks of size n and all possible input assignments.

History trees. *History trees* were introduced in [15] as a tool of investigation for anonymous dynamic networks; an example is found in Figure 1. A history tree is a representation of a dynamic network given some inputs to the processes. It is an infinite graph whose nodes are partitioned into *levels* L_t, with $t \geq -1$; each node in L_t represents a class of processes

[5] Each self-loop in G_t represents a single link, hence a single message being sent and received by the same process.

[6] A similar parameter for dynamic networks is the *dynamic diameter* D, defined as the maximum number of rounds it may take for a message to be broadcast from a process to all other processes. The parameters T and D are related by the inequalities $T \leq D \leq T(n-1)$, which are best possible [17].

[7] Contrary to static networks, where processes may be allowed to send different messages to different neighbors, dynamic networks usually require processes to "broadcast" the same message through all incident links due to the lack of unique port numbers. This is the case, for instance, in wireless radio communications, where messages are sent to all processes within communication range, and the anonymity of the network prevents destinations from being specified.

[8] We may want to assume that the total number of links in a network multigraph G_t (counted with their multiplicities) is bounded by a polynomial in n. In fact, when dealing with congested networks, we explicitly make this assumption, as it ensures that the multiplicity of a link can always fit in a single $O(\log n)$-sized message.

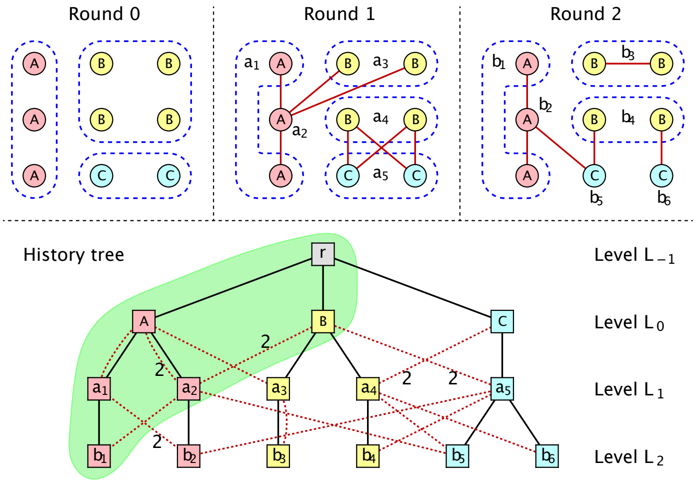


Figure 1 The first rounds of a dynamic network with $n = 9$ processes and the corresponding levels of the history tree. Level L_t consists of the nodes at distance $t + 1$ from the root r, which represent indistinguishable processes after the tth communication round. There are no leaders in the network, but each process has an input from the set $\{A, B, C\}$. Only the nodes in L_0 have explicit labels; all labels of the form a_i and b_i were added for the reader's convenience, and indicate classes of indistinguishable processes (in contrast, the nodes of the *virtual history tree* introduced in Section 4 do have IDs). Note that the two processes in b_4 are still indistinguishable at the end of round 2, although they are linked to the distinguishable processes b_5 and b_6. This is because such processes were in the same class a_5 at round 1. The subgraph induced by the nodes in the green blob is the *view* of the two processes in b_1. We remark that a history tree does not contain any explicit information about how many processes each node represents.

that are *indistinguishable* at the end of round t (with the exception of L_{-1}, which contains a single node r representing all processes). The definition of distinguishability is inductive: At the end of round 0, two processes are distinguishable if and only if they have different inputs. At the end of round $t \geq 1$, two processes are distinguishable if and only if they were already distinguishable at round $t - 1$ or if they have received different multisets of messages at round t. (We refer to "multisets" of messages, as opposed to sets, because multiple copies of identical messages may be received; each message has a *multiplicity*.)

Each node in level L_0 has a label indicating the input of the processes it represents. There are also two types of edges connecting nodes in adjacent levels. The *black edges* induce an infinite tree spanning all nodes, rooted at node $r \in L_{-1}$. The presence of a black edge $\{v, v'\}$, with $v \in L_t$ and $v' \in L_{t+1}$, indicates that the *child node* v' represents a subset of the processes represented by the *parent node* v. The *red multi-edges* represent communications between processes. The presence of a red edge $\{v, v'\}$ with multiplicity m, with $v \in L_t$ and $v' \in L_{t+1}$, indicates that, at round $t+1$, each process represented by v' receives m (identical) messages from processes represented by v.

The *view* of a process p at round $t \geq 0$ as the subgraph of the history tree which is spanned by all the shortest paths (using black and red edges indifferently) from the root r to the node in L_t representing p (Figure 1 shows an example of a view).

Applications of history trees. As proved in [15], the view of any process at round $3n$ contains enough information to determine the multiset of the initial inputs (i.e., how many processes have each input value). Thus, once a process is able to locally construct a view spanning $3n$ levels of a history tree, it can immediately do any computation on the inputs, and in particular determine n and solve the Generalized Counting problem.

History trees were adopted in [15, 17], where it is shown how processes can construct their views of the history tree in real time. For the algorithm to work, each process is required to repeatedly send its current view to all its neighbors at every round, merging it with all the views it receives from them.

This approach is not feasible in the congested network model, because a view at round t has size $\Theta(tn^2 \log n)$ in the worst case, since there may be $\Theta(n^2)$ red edges in each level. In the following, we will develop a strategy whereby processes can construct a history tree one red edge at a time. The core idea is that the nodes of the history tree are assigned unique IDs, and therefore a single red edge can be encoded in only $O(\log n)$ bits as a pair of IDs and a multiplicity.

4 Basic Counting Algorithm

In this section we describe our deterministic Counting algorithm for congested anonymous dynamic networks in its most basic version. The algorithm assumes the network to be connected ($T = 1$), to have a unique leader, and execution terminates in $O(n^3 \log n)$ rounds with the leader reporting the total number of processes in the system, n. The complete pseudocode is found in the arXiv version of this paper.

In Section 6, we will optimize this basic algorithm, making it terminate in $O(n^3)$ rounds. We will also extend the algorithm in several directions, for instance by making all processes (as opposed to the leader only) simultaneously output n and terminate. Furthermore, we will show how to not only count the number of processes, but also compute arbitrary (multiset-based, cf. [15]) functions, assuming that input values are assigned to the processes. Finally, we will extend the algorithm to leaderless networks and T-union-connected networks.

4.1 Algorithm Outline

The only input given to each process is whether or not it is the leader. Each process also has some private memory which is used to permanently store information in the form of internal variables.

Virtual history tree (VHT). The overall goal of the algorithm is for the processes to implicitly agree on the first $O(n)$ levels of a particular history tree, called *virtual history tree (VHT)*, which corresponds to a dynamic network $\mathcal{N}$ of n processes. Once the construction of each new level of the VHT is complete (refer to Section 4.5), the leader locally runs the Counting algorithm from [15] on the VHT. If this algorithm successfully returns a number (as opposed to "Unknown"), the leader outputs it; otherwise, the construction of a new level of the VHT is initiated.

Virtual network ($\mathcal{N}$). The dynamic network $\mathcal{N} = (N_1, N_2, \dots)$ represented by the VHT is in fact a *virtual network*, in the sense that none of the multigraphs N_t necessarily coincides with any multigraph of links actually occurring in the real communication network $\mathcal{G} = (G_1, G_2, \dots)$. However, each N_t is obtained by carefully adding and removing links from some G_{i_t} (see Figure 2). This manipulation has the purpose of reducing the size of the resulting VHT by a factor of n (see Section 4.4).

Temporary IDs. To cope with the fact that processes are anonymous and information can only be sent in small chunks of size $O(\log n)$, each process has a *temporary ID* stored in a local variable called `MyID`. Each node v in the VHT also has an ID, indicating that v

represents all processes having that ID. Thus, a *red-edge triplet* of the form $(\mathtt{ID1}, \mathtt{ID2}, \mathtt{Mult})$ can be used to unambiguously represent a red edge of multiplicity `Mult` between the nodes of the VHT whose IDs are `ID1` and `ID2`. Since a red-edge triplet has size $O(\log n)$, it can be included in a single message. Note that the variable `MyID` of each process may be modified over time as the VHT acquires more nodes.

Broadcast phases. The construction of the VHT is carried out level by level, and is done through several *broadcast phases*, which are indirectly coordinated by the leader (see Section 4.3). At first, each process knows the red edges incident to its corresponding node of the VHT. Then, ideally every two broadcast phases, the whole network learns a new red edge of the VHT. The broadcast phases continue until all processes know all red edges in the level (see Section 4.6).

Estimating the diameter. In order to guarantee the success of a broadcast phase, all processes must keep sending each other information for a certain number of rounds, which depends on the *dynamic diameter* of the network, and is $n - 1$ in the worst case [28]. Since the processes do not initially possess any information at all, they can only make estimations on the dynamic diameter. The current estimate is stored by each process in the variable `DiamEstimate`. Its value is initially 1, and it is doubled every time the processes detect an error in a broadcast.

Error phases. Detecting broadcasting errors and consistently reacting to them is by no means a trivial task, and is discussed in Section 4.7. Our broadcasting technique ensures that, if some red-edge triplet fails to be broadcast to the entire network and does not become part of the local VHT of all processes, at least one process becomes aware of this fact. Such a process enters an *error phase*, sending a high-priority message at every round containing the level number at which the error occurred. Error messages supersede the regular ones and eventually reach the leader.

Reset phases. When the leader finally receives an error message, it initiates a *reset phase*, whose goal is to force the whole network to restore a previous state of the VHT and continue from there. This is achieved by broadcasting a high-priority reset message. Since the error must have occurred because `DiamEstimate` was too small, its value is doubled at the end of the reset phase.

Note that there is no obvious way for the leader to tell if any level of the VHT is actually missing some parts: At any time, there may be processes in an error phase unbeknownst to the leader. One of the challenges of our method is to ensure that the leader will not terminate with an incorrect guess on n due to the VHT being incomplete.

4.2 Communication and Priority

Counting rounds. The processes have to implicitly synchronize with one another to start and finish each broadcast phase at the same time. Part of the synchronization is achieved by the function `SendAndReceive`, which is called by each process at every communication round. This function simply sends a given message to all neighbors, collects all messages coming from the neighbors, and increments the internal variable `CurrentRound`. Since communications in the network are synchronous, all processes always agree on the value of `CurrentRound`.

Message types. The processes use messages of various types to share information with one another. Each message has a *label* describing its type, as well as at most three additional integer parameters. As it will turn out in the analysis of the algorithm, each parameter has size $O(\log n)$ bits.[9] The message types and their parameters are as follows.

- **Null message.** Label: "Null". No parameters.
- **Level-begin message.** Label: "Begin". Parameters: `ID`.
- **Level-end message.** Label: "End". No parameters.
- **Done message.** Label: "Done". Parameters: `ID`.
- **Red-edge message.** Label: "Edge". Parameters: `ID1`, `ID2`, `Mult`.
- **Error message.** Label: "Error". Parameters: `ErrorLevel`.
- **Reset message.** Label: "Reset". Parameters: `ResetLevel`, `StartingRound`, `NewDiam`.

Priority. Messages have *priorities* that determine how they are handled during a broadcast. The priority of a message is defined as follows:

$$\text{Null} < \text{Begin} < \text{End} < \text{Done} < \text{Edge} < \ldots < \text{Reset } k+1 < \text{Error } k < \text{Reset } k < \ldots < \text{Error } 1 < \text{Reset } 1$$

That is, the message with lowest priority is the Null message, followed by all possible Level-begin messages, then the Level-end message, etc. The priority of a Level-begin message is independent of its parameter. For all other message types, however, the priority is also a function of the parameters.

As for Done messages, different `ID` parameters yield different priorities. Thus, priority induces a total ordering on the set of all possible Done messages; the precise ordering is irrelevant, as long as all processes implicitly agree on it. All Done messages have greater priority than Null, Level-begin and Level-end messages. The same goes for Red-edge messages: Different parameters yield different priorities, and all processes agree on the priority function. All Red-edge messages have greater priority than all Done messages, and lower priority than all Error and Reset messages.

An Error message (respectively, a Reset message) with a smaller `ErrorLevel` (respectively, `ResetLevel`) has a greater priority. Moreover, the priorities of Error and Reset messages are interleaved: The priority of an Error message with `ErrorLevel` $= k$ is strictly between the priority of a Reset message with `ResetLevel` $= k+1$ and the priority of a Reset message with `ResetLevel` $= k$.

4.3 Broadcasting Data

During the execution of the algorithm, a non-leader process may have a particular piece of information that it wishes to send to the leader. Similarly, the leader may have some information that it wishes to share with all processes in the network. Both operations are performed via a *broadcast* spanning several rounds.

The broadcast technique used to construct the VHT is implemented as follows. It is assumed that all processes participating in the broadcast are *synchronized*, i.e., they start at the same round and continue broadcasting for the same number of rounds (which is equal to `DiamEstimate`). Each process is also assumed to have the information it wishes to share packed in a message of the appropriate type (see Section 4.6), which is passed to the function `BroadcastPhase` as the argument `Message`.

[9] As already remarked, the algorithm spontaneously creates $O(\log n)$-sized messages without any a-priori knowledge on n.

At each broadcast round, each process sends its message to all its neighbors. Then it examines the messages received from the neighbors, as well as its own message, and keeps only the message with highest priority, discarding all others (function `BroadcastStep`). This is the message the process will send in the next round, and so on.[10]

Ideally, if the broadcast is continued for a sufficiently large number of rounds, all processes participating in the broadcast will eventually obtain the message having the highest priority among the ones initially owned by the processes (note that this may be an Error or Reset message, as well).

4.4 Defining the Virtual Network

Definition. Recall that the VHT is the history tree of the virtual network $\mathcal{N} = (N_1, N_2, N_3, \dots)$. We will now define N_t by induction on t. That is, assuming that the multigraphs N_1, N_2, …, N_{t-1} are already known, we will construct N_t based on the multigraph G_{i_t}, which represents the real communication network $\mathcal{G}$ at a selected round i_t (Section 4.6 explains how i_t is selected).

Since the first $t-1$ rounds of the virtual network are known, we can construct the first levels of the VHT up to level L_{t-1}. By definition of history tree, each node $v \in L_{t-1}$ represents a class P_v of processes that are indistinguishable after the first $t-1$ "virtual rounds" modeled by the communication networks N_1, N_2, …, N_{t-1}.

Consider the simple undirected graph $H = (V, E)$, where $V = L_{t-1}$ and $\{u, v\} \in E$ if and only if $u \neq v$ and there is at least one link in G_{i_t} between a process in P_u and one in P_v. Recall that we are assuming that G_{i_t} is connected, and therefore H is connected, as well (disconnected networks will be discussed in Section 6). Let $S = (V, E')$ be an arbitrary spanning tree of H.

Now, N_t is the network having the following links (refer to Figure 2):

- For every edge $\{u, v\} \in E'$ of S, the multigraph N_t contains all the links in G_{i_t} having an endpoint in P_u and an endpoint in P_v (with the respective multiplicities).
- For every $v \in V$, the multigraph N_t contains a cycle C_v spanning all the processes in P_v. In the special case where P_v contains exactly two processes p_1 and p_2, C_v is a double link between p_1 and p_2. If P_v contains a single process p, C_v is a double self-loop on p.

Note that, in every case, C_v induces a 2-regular multigraph on P_v. The purpose of these (possibly degenerate) cycles C_v is to ensure that N_t is connected. On the other hand, the purpose of using a spanning tree S, as opposed to the full graph H, is to reduce the size of the VHT.

Implementation. In the actual algorithm, N_t is defined only implicitly in a distributed manner (because the ultimate goal is merely the construction of the VHT). The implicit construction of N_t starts at round i_t, where each process (not in an error phase) invokes the function `SetUpNewLevel` and sends a Begin message containing its own ID to each neighbor. As a result, each process learns the IDs of all its neighbors in G_{i_t}, as well as their multiplicities, and stores this information as a list of pairs of the form $(\texttt{ID}, \texttt{Mult})$ in the internal variable `ObsList`. It should be noted that a process discards all Begin messages from processes with the same ID as its own. These are replaced by the single pair $(\texttt{MyID}, 2)$, which accounts for the two edges of the cycle C_v incident to the process in N_t.

[10] Note that our broadcasting strategy implements a token-forwarding mechanism, and the sequence of all broadcast phases is akin to a Token Dissemination algorithm. It is known that any such algorithm must have a worst-case running time of at least $\Omega(n^2/\log n)$ rounds [18].

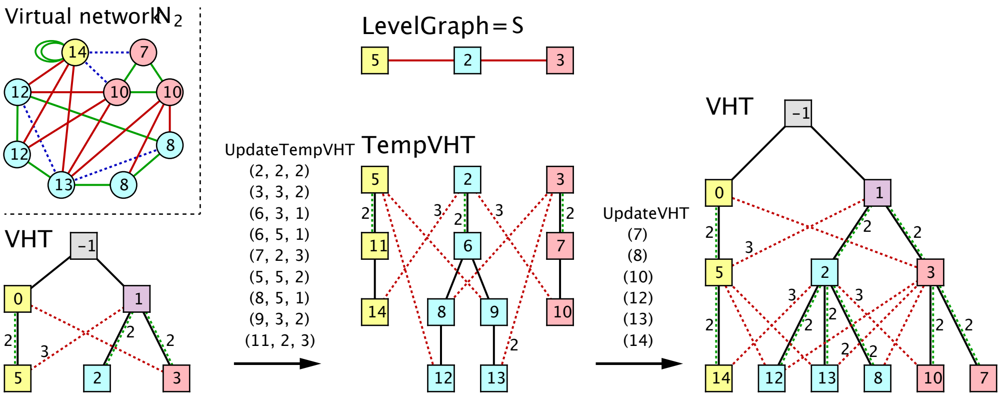


Figure 2 Construction of the virtual network N_2 and level L_2 of the virtual history tree (VHT). The real network G_{i_2} consists of the $n = 9$ processes in the upper-left picture, connected by the red and blue links (not the green ones). Before the construction starts, the IDs in the network are 5, 2, and 3. Same-colored processes have equal IDs when the construction of L_2 starts; the labels indicate their IDs when the construction finishes. Accordingly, level L_1 of the VHT has three nodes with IDs 5, 2, 3. The graph H is a triangle on these three nodes, while its spanning tree S is the same as `LevelGraph`. Therefore, to construct N_2 we remove the blue edges from G_{i_2} and keep the red ones (see Section 4.4). For example, after the triplets $(6, 3, 1)$ and $(6, 5, 1)$ are accepted, any elements of `ObsList` corresponding to the two blue edges incident to the yellow process are deleted and are never broadcast. We also have to add the green edges, which represent the cycles C_v. Note that failing to add them would result in a disconnected network, because the process with ID 7 would be isolated. For clarity, the edges of the (temporary) VHT are colored red or green to match the edges of the virtual network that they represent (although technically they are all red edges).

The choice of links to be included in N_t (which directly reflects on the red edges included in the VHT) is guided by the construction and maintenance of an *auxiliary graph* stored in each process' internal variable `LevelGraph`. The auxiliary graph is a graph on $V = L_{t-1}$ and starts with no edges at all; it gradually acquires more edges until it becomes the spanning tree S (as defined above). This is carried out as part of the function `UpdateTempVHT`, which is invoked every time a red-edge triplet $(\texttt{ID1}, \texttt{ID2}, \texttt{Mult})$ is selected to become a new red edge of the VHT at the end of a broadcast phase. Among other operations (described in Section 4.5), this function adds an edge to the auxiliary graph which connects the two nodes corresponding to `ID1` and `ID2`. Then the function `PreventCyclesInLevelGraph` is called, which deletes all the pairs in `ObsList` whose selection would cause the creation of a cycle in the auxiliary graph. This guarantees that, eventually, `LevelGraph` will be a tree (representing S).

4.5 Constructing the Virtual History Tree (VHT)

Initialization. The VHT is initialized by the function `InitializeVariables`. At first, the VHT only contains level L_{-1} (a single root node whose ID is -1) and level L_0 (two nodes with IDs 0 and 1, representing the leader and the non-leader processes, respectively). Accordingly, the leader initializes its own `MyID` variable to 0 and the non-leaders to 1.

Temporary VHT. Then, for all $t \geq 1$, the level L_t of the VHT is constructed based on the virtual network N_t as in Figure 2. When the construction begins, all processes (that are not in an error phase) acquire new pairs of the form $(\texttt{ID}, \texttt{Mult})$ at the same round i_t and store them in `ObsList`, as explained in Section 4.4. The construction of a new level is not carried out directly on the VHT, but in a separate *temporary VHT* stored in the internal variable TempVHT. Initially, this is just a copy of level L_{t-1} of the VHT, i.e., a set of nodes, each with a distinct ID, and no edges.

Adding red edges. In order to determine how TempVHT should be updated, several broadcast phases are performed, allowing the processes to share their red-edge triplets with one another. Note that a process can transform each element of ObsList of the form (ID, Mult) into a red-edge triplet by simply adding its own ID as the first element. Every time a red-edge triplet (ID1, ID2, Mult) has been broadcast by a process to the entire network and has been "accepted" (see Section 4.6 for details), the function UpdateTempVHT is called. The result is that the node v of TempVHT whose ID is ID1 gets a new child v' with a new unique ID. Pictorially, a black edge is created connecting v' to v. Also, a red edge with multiplicity Mult is added to TempVHT, connecting the new node with the node whose ID is ID2.

Updating IDs. By definition, the red-edge triplet (ID1, ID2, Mult) indicates that *some* processes whose ID is ID1 have received exactly Mult messages from processes whose ID is ID2. These processes, which were previously represented by the node v, are now represented by v'. Therefore, every process whose ID is ID1 that has the pair (ID2, Mult) in its local ObsList removes it from the list and modifies its own ID from ID1 to the ID of v'.[11]

Updating the VHT. When a process has no more red-edge triplets to share (that is, its ObsList is empty), it broadcasts a Done message containing its current ID. Recall that Done messages have lower priority than Red-edge messages (see Section 4.2). When the final result of a broadcast is a Done message containing a certain ID, all processes assume that *some* processes with that ID have sent all their red-edge triplets. Therefore, the node of the temporary VHT with that ID is ready to be added to the VHT.

To this end, the function UpdateVHT is called with the ID contained in the Done message. This function creates a new node v in level L_t of VHT corresponding to the node v' of TempVHT whose ID is the one passed to the function (refer to Figure 2). The node v gets the same ID as v' and becomes a child of the node $u \in L_{t-1}$ having the same ID as the root v'' of the tree containing v'. Then, v takes all the red edges found along the path from v' to v''.

Finalizing the level. When the ObsList of a process is empty and the VHT already contains a node with its ID, the process broadcasts an Level-end message, which has lower priority than Done and Red-edge messages. When the result of a broadcast is an Level-end message, the construction of the level is finished.

4.6 Main Loop

The entry point of the algorithm is the function Main. After initializing the internal variables as already explained, the function goes through a loop that constructs the VHT level by level. At several points in this loop there may be errors that cause some of the levels to be undone and execution to resume from the beginning of the loop; we will discuss errors in Section 4.7. Next, we will describe an ideal error-free execution.

[11] To clarify, the nodes of the VHT have unique IDs. Each node represents a class of "indistinguishable" processes, all of which have the same ID as the node itself. When processes disambiguate in the virtual network, they obtain different IDs. However, it is not necessarily true that all processes will have distinct IDs eventually. For example, if the network is the (static) complete graph, all non-leader processes will always have the same ID (which is incremented at every virtual round).

Level initialization. With each iteration of the main loop, a new level of the VHT is constructed. At the beginning, the function `SetUpNewLevel` is executed by all processes: This marks a selected round G_{i_t} as defined in Section 4.4 (the non-trivial fact that all processes always call this function at the same round will become apparent from our analysis of the algorithm, in Section 5).

VHT broadcast. Then an inner loop is entered; the purpose of each iteration is for all processes to learn new information, which causes an update of the temporary VHT or the VHT itself. At first, each process calls the function `MakeVHTMessage`, which picks an element from `ObsList`, converts it into a red-edge triplet by adding its own ID to it, and wraps it in a Red-edge message. If `ObsList` is empty, a Done message containing the process' ID is generated instead. The resulting message is used in a first broadcast phase which, after a number of rounds equal to `DiamEstimate`, returns the highest-priority message circulating in the network. This message is stored in the variable `VHTMessage`.

Acknowledgment broadcast. Note that, in the presence of a faulty broadcast, different processes may end up having different versions of `VHTMessage`. To ensure that all processes (that are not in an error phase) update their local copies of the (temporary) VHT in a consistent way, an "acknowledgment" broadcast phase is performed. Its purpose is for the leader to inform all other processes of the *accepted message*; by definition, the accepted message is the leader's version of `VHTMessage`.

In the acknowledgment phase, all non-leader processes broadcast a low-priority Null message, while the leader broadcasts the accepted message. The message resulting from the broadcast is then stored in the variable `AckMessage`.

Updating the level. Now, each process compares the contents of the two messages `VHTMessage` and `AckMessage`. If they are the same (hence not Null), then this is indeed the accepted message coming from the leader, and the data therein is used to update the temporary VHT or the VHT. Specifically, if `AckMessage` is a Red-edge message, the red-edge triplet therein is used to update the temporary VHT. If it is a Done message, the VHT is updated instead.

Finalizing the level. When a process is done broadcasting red-edge triplets and the VHT already contains a node representing it, the process broadcasts a low-priority Level-end message. As soon as the accepted message is the Level-end message, the level is considered complete and the inner loop is exited.

Counting processes. Now the leader extracts its own view from the VHT and locally runs the Counting algorithm from [15] on it (function `CountFromView`). By "extracts its own view" we mean that it makes a copy of the VHT and deletes all nodes that are not on a shortest path from the root to the deepest leader's node.

If the value returned by `CountFromView` is a number, this is taken as the correct number n of processes in the virtual network $\mathcal{N}$ (and hence in the real network $\mathcal{G}$) and becomes the leader's output. Otherwise, a new iteration of the main loop starts, a new level of the VHT is constructed, and so on.

4.7 Handling Errors and Resets

Detecting errors. As mentioned, a broadcast phase lasts a number of rounds indicated by the variable `DiamEstimate`. If this number happens to be smaller than the dynamic diameter of the network, the broadcast may be unsuccessful, in the sense that not all processes may end up agreeing on the same highest-priority message. Fortunately, the protocol described in Section 4.6 makes error detection very simple: At the end of every acknowledgment broadcast, if the contents of a process' variables `VHTMessage` and `AckMessage` are different, the broadcast was unsuccessful. Any process that detects this event enters an error phase.

Error phase. When a process enters an error phase, it runs the function `BroadcastError`; as it turns out, only non-leader processes ever execute this function. During an error phase, a process continually broadcasts an Error message containing the index of the level of the VHT where the error was detected. This message is replaced by any other message of higher priority received by the process during the error phase (see Section 4.2). As soon as the process receives a Reset message for a level of smaller or equal depth than the one indicated by the current Error message (i.e., a Reset message of higher priority), it ends the error phase and enters a reset phase.

Another situation where a non-leader process enters an error phase is when it receives an Error message from some other process. In this case, it calls the function `HandleError` and starts broadcasting a new Error message containing the smaller between the index of the current level and the one contained in the received Error message. This causes higher-priority error messages to propagate through the network and eventually reach the leader.

Reset phase. This phase is initiated by the leader at the end of a broadcast phase (or after sending a Begin message) in case it received an Error message. The leader first waits for $2 \cdot \texttt{DiamEstimate} + 1$ rounds, sending only Null messages. This is to ensure that all non-leader processes finish any broadcast phase and enter an error phase. In turn, this prevents any possible conflicts between different broadcast phases before and after a reset.

Then the leader creates a Reset message containing the index of the VHT level where the error occurred, the current round number, and the new estimate on the dynamic diameter, i.e., twice the current one, since the error occurred because the estimate was too small.

Now the leader calls the function `BroadcastReset`, where it broadcasts the Reset message; any process that receives this message starts broadcasting it as well (provided that it has higher priority than the Error message it is currently broadcasting, as usual). The broadcast continues for a total number of rounds equal to the new dynamic diameter estimate. All processes that receive the Reset message are able to synchronize with one another and finish the reset phase at the same round, thanks to the information contained in the message itself.

At the end of the reset phase, the reset is actually performed: Every process that got the Reset message deletes the most recent levels of the VHT up to the level where the error occurred (this information is contained in the Reset message) and reverts its ID to the one it had at the beginning of the construction of that level. The variable `DiamEstimate` is also updated with the new estimate. At this point, the network is ready to resume construction of the VHT.

5 Proof of Correctness

We will now sketch a proof of correctness for the Counting algorithm in Section 4. The interested reader may refer to the arXiv version for a detailed and technically rigorous proof.

We say that two processes *agree* on a variable at a certain round if their local instances of that variable are equal. As it turns out, all processes that are not in an error phase must always agree on the variable `DiamEstimate`: this can be proved by induction on the number of reset phases that the leader has completed.

In turn, this implies that all processes (that are not in an error phase) are able to implicitly synchronize their broadcast phases with the leader, since these phases have a duration of `DiamEstimate` rounds. As a consequence, all processes (not in an error phase) also agree on the variables `CurrentLevel`, `VHT`, `TempVHT`, `LevelGraph`, and `NextFreshID`, because these variables are updated either at the beginning or at the end of a broadcast or reset phase.

Thus, we can unambiguously refer to phases and variables with no explicit mention of any individual process, because these phases and variables are the same throughout the whole network (excluding the processes in an error phase).

The goal of the algorithm is to construct the first levels of the VHT, i.e., the history tree of the virtual network $\mathcal{N}$. However, information on the virtual network is shared by all processes through several broadcast phases, and there is no guarantee that every broadcast phase will have the correct outcome (some data may fail to be communicated if `DiamEstimate` is smaller than the network's actual dynamic diameter). Thus, we distinguish between the *ideal VHT*, i.e., the "correct" history tree corresponding to $\mathcal{N}$, and the *effective VHT*, i.e., the one that is actually being constructed by the processes, which may be missing some information.

It is a byproduct of our protocol that, whenever a node is added to the effective VHT, then all red edges incident to it (and to nodes in the previous level of the ideal VHT) are also included. In particular, extracting the leader's view from the effective VHT yields a graph isomorphic to the view of the leader in the ideal VHT. This implies that the function `CountFromView` is always given as input a view of the history tree of a connected network of n processes. As proved in [15], this function must return either "Unknown" or n; in particular, it returns n when the input view spans at least $3n$ levels.

Hence, our algorithm cannot return an incorrect number. It remains to prove that it does terminate within the desired number of rounds. Every time a faulty broadcast is detected and the leader is notified of it, a reset phase is initiated and `DiamEstimate` is doubled. It follows that no more than $O(\log n)$ reset phases ever occur, and `DiamEstimate` is always $O(n)$; this is also the duration of each broadcast phase.

Recall that links are not included in the virtual network if they cause cycles to appear in `LevelGraph` (see Section 4.4). Although this strategy may not guarantee that every level of the VHT has $O(n)$ red edges, it does amortize the total number of red edges over several levels. In fact, the first $O(n)$ levels of the ideal VHT only contain $O(n^2)$ red edges (as opposed to $O(n^3)$, which may be the case for the history tree of a generic network).

Summarizing, it takes $O(n^2)$ broadcast phases to construct the first $3n$ levels of the VHT, resulting in a total of $O(n^3)$ rounds. The construction process is partially undone and repeated at most $O(\log n)$ times due to resets, and so the final running time is $O(n^3 \log n)$.

It is straightforward to prove that the algorithm works in the congested network model, because all the data contained in a message are binary representations of values that are polynomial in n, e.g., round numbers, estimates of the dynamic diameter, or temporary IDs of processes. Since each message contains at most three of these values, its size is $O(\log n)$.

6 Extensions and Improvements

Simultaneous termination. In Section 4 we showed how the leader can determine n. If we want all processes, not just the leader, to simultaneously output n and terminate, we can use the following protocol. As soon as the leader knows n, it broadcasts a message of maximum priority containing n and the current round number c. Any process that receives this message keeps forwarding it until round $c+n$. By this round they all have it, and thus can output n and terminate simultaneously.

General computation. If each process is assigned an $O(\log n)$-sized input (other than the leader flag), we can also perform general computations on the multiset of inputs. We only have to adapt the algorithm of Section 4 to work with history trees whose level L_0 contains one node per input occurring in the system, and then run the algorithm in [15] on this extended history tree. This can easily be done with the techniques developed in Section 4: We can construct L_0 like any other level, except that we use broadcast phases to transmit inputs instead of red edges.

Optimized running time. We can sightly speed up our algorithm at the cost of some additional bookkeeping. The idea is to refine the error and reset mechanism by resuming the construction of the VHT not from the *level* that caused an error, but from the *broadcast phase* that caused it.

The processes have to remember the order in which the leader accepted Red-edge and Done messages, as well as the Begin messages received at every begin round. When a process detects an error, it broadcasts an Error message containing not the current level number, but the number of messages that the leader has accepted up to that time. The advantage is that the reset phase can rewind the (temporary) VHT exactly to the desired point without erasing entire levels. Furthermore, by looking up the Begin messages received in the appropriate begin round, each process is also able to reconstruct its local `ObsList` at the desired time.

If a broadcast phase causes an error, the leader receives a notification after less than n rounds, and so it only has to undo the work done in $O(n)$ rounds. Since there can be at most $O(\log n)$ resets, the total time spent in reset phases or doing work that later gets undone is $O(n \log n)$ rounds. Thus, the new algorithm runs in $O(n^3) + O(n \log n) = O(n^3)$ rounds.

Leaderless computation. It is known that doing any non-trivial computation in a leaderless network requires some extra assumptions, for example the knowledge of an upper bound D on its dynamic diameter [17]. However, knowing D allows processes to reliably broadcast messages in phases of D rounds. This immediately yields an extension of our algorithm to leaderless networks in $O(Dn^2)$ rounds, where no acknowledgment phases or error and reset phases are needed.

Disconnected networks. We can extend our algorithm to T-union-connected networks, assuming the parameter T is known. As discussed in [17], the idea is to divide the sequence of rounds into blocks of size T. Within every block, each process keeps sending the same message and stores all incoming messages. At the end of a block, each process runs the algorithm from Section 4 (or its optimization) pretending that all the stored messages arrived in a single round. This is equivalent to running the algorithm on the dynamic network $\mathcal{G}^\star = (G_1^\star, G_{T+1}^\star, G_{2T+1}^\star, \dots)$, which is always connected ($G_t^\star$ is defined in Section 3). The running time is simply $O(Tn^3)$ rounds.

7 Concluding Remarks

In this paper we have extended the theory of history trees by introducing the tools necessary for the distributed construction and transmission of history trees in the congested network model. This resulted in a new state of the art for general computation in disconnected anonymous dynamic congested networks, with or without leaders.

Our history tree construction technique leads to general algorithms whose running time is cubic in the size of the network. An immediate open problem is whether this running time can be reduced. Since our algorithm broadcasts information using a token-forwarding approach, and by virtue of the $\Omega(n^2/\log n)$ lower bound of [18], we believe that it would be unlikely to achieve a better running time without a radical change in the technique used.

Understanding whether the Counting problem has as a super-linear lower bound in congested networks is of special importance, because it would mark a computational difference between anonymous dynamic networks in the congested and non-congested models.

It would be interesting to do a thorough fine-grained tradeoff analysis of our algorithm. For instance, it is not difficult to show that, if messages have size $O(n \log n)$, the running time of our algorithm can be reduced to $O(n^2)$.

References

1 D. Angluin, J. Aspnes, and D. Eisenstat. Fast Computation by Population Protocols with a Leader. *Distributed Computing*, 21(3):61–75, 2008.

2 J. Aspnes, J. Beauquier, J. Burman, and D. Sohier. Time and Space Optimal Counting in Population Protocols. In *Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS '16)*, pages 13:1–13:17, 2016.

3 J. Beauquier, J. Burman, S. Clavière, and D. Sohier. Space-Optimal Counting in Population Protocols. In *Proceedings of the 29th International Symposium on Distributed Computing (DISC '15)*, pages 631–646, 2015.

4 J. Beauquier, J. Burman, and S. Kutten. A Self-stabilizing Transformer for Population Protocols with Covering. *Theoretical Computer Science*, 412(33):4247–4259, 2011.

5 D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Inc., USA, 1989.

6 A. Casteigts, F. Flocchini, B. Mans, and N. Santoro. Shortest, Fastest, and Foremost Broadcast in Dynamic Networks. *International Journal of Foundations of Computer Science*, 26(4):499–522, 2015.

7 A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-Varying Graphs and Dynamic Networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

8 B. Charron-Bost and P. Lambein-Monette. Randomization and Quantization for Average Consensus. In *Proceedings of the 57th IEEE Conference on Decision and Control (CDC '18)*, pages 3716–3721, 2018.

9 B. Charron-Bost and P. Lambein-Monette. Computing Outside the Box: Average Consensus over Dynamic Networks. In *Proceedings of the 1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND '22)*, pages 10:1–10:16, 2022.

10 B. Chazelle. The Total s-Energy of a Multiagent System. *SIAM Journal on Control and Optimization*, 49(4):1680–1706, 2011.

11 G. A. Di Luna and G. Baldoni. Brief Announcement: Investigating the Cost of Anonymity on Dynamic Networks. In *Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC '15)*, pages 339–341, 2015.

12 G. A. Di Luna, R. Baldoni, S. Bonomi, and I. Chatzigiannakis. Counting in Anonymous Dynamic Networks Under Worst-Case Adversary. In *Proceedings of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS '14)*, pages 338–347, 2014.

13 G. A. Di Luna, S. Bonomi, I. Chatzigiannakis, and R. Baldoni. Counting in Anonymous Dynamic Networks: An Experimental Perspective. In *Proceedings of the 9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS '13)*, pages 139–154, 2013.

14 G. A. Di Luna, P. Flocchini, T. Izumi, T. Izumi, N. Santoro, and G. Viglietta. Population Protocols with Faulty Interactions: The Impact of a Leader. *Theoretical Computer Science*, 754:35–49, 2019.

15 G. A. Di Luna and G. Viglietta. Computing in Anonymous Dynamic Networks Is Linear. In *Proceedings of the 63rd IEEE Symposium on Foundations of Computer Science (FOCS '22)*, pages 1122–1133, 2022.

16 G. A. Di Luna and G. Viglietta. Brief Announcement: Efficient Computation in Congested Anonymous Dynamic Networks. In *Proceedings of the 42nd ACM Symposium on Principles of Distributed Computing (PODC '23)*, pages 176–179, 2023.

17 G. A. Di Luna and G. Viglietta. Optimal computation in leaderless and multi-leader disconnected anonymous dynamic networks. In *Proceedings of the 37th International Symposium on Distributed Computing (DISC '23)*, pages 18:1–18:20, 2023.

18 C. Dutta, G. Pandurangan, R. Rajaraman, Z. Sun, and E. Viola. On the Complexity of Information Spreading in Dynamic Networks. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '13)*, pages 717–736, 2013.

19 P. Fraigniaud, A. Pelc, D. Peleg, and S. Pérennes. Assigning Labels in Unknown Anonymous Networks. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC '00)*, pages 101–111, 2000.

20 B. Haeupler and F. Kuhn. Lower Bounds on Information Dissemination in Dynamic Networks. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC '12)*, pages 166–180, 2012.

21 D. R. Kowalski and M. A. Mosteiro. Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP '18)*, pages 156:1–156:14, 2018.

22 D. R. Kowalski and M. A. Mosteiro. Polynomial Anonymous Dynamic Distributed Computing Without a Unique Leader. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 147:1–147:15, 2019.

23 D. R. Kowalski and M. A. Mosteiro. Polynomial Counting in Anonymous Dynamic Networks with Applications to Anonymous Dynamic Algebraic Computations. *Journal of the ACM*, 67(2):11:1–11:17, 2020.

24 D. R. Kowalski and M. A. Mosteiro. Supervised Average Consensus in Anonymous Dynamic Networks. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '21)*, pages 307–317, 2021.

25 D. R. Kowalski and M. A. Mosteiro. Efficient Distributed Computations in Anonymous Dynamic Congested Systems with Opportunistic Connectivity. *arXiv:2202.07167 [cs.DC]*, pages 1–28, 2022.

26 D. R. Kowalski and M. A. Mosteiro. Polynomial Anonymous Dynamic Distributed Computing Without a Unique Leader. *Journal of Computer and System Sciences*, 123:37–63, 2022.

27 F. Kuhn, T. Locher, and R. Oshman. Gradient Clock Synchronization in Dynamic Networks. *Theory of Computing Systems*, 49(4):781–816, 2011.

28 F. Kuhn, N. Lynch, and R. Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC '10)*, pages 513–522, 2010.

29 F. Kuhn, Y. Moses, and R. Oshman. Coordinated Consensus in Dynamic Networks. In *Proceedings of the 30th ACM Symposium on Principles of Distributed Computing (PODC '11)*, pages 1–10, 2011.

30 F. Kuhn and R. Oshman. Dynamic Networks: Models and Algorithms. *SIGACT News*, 42(1):82–96, 2011.

31 O. Michail, I. Chatzigiannakis, and P. G. Spirakis. Naming and Counting in Anonymous Unknown Dynamic Networks. In *Proceedings of the 15th International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS '13)*, pages 281–295, 2013.

32 O. Michail and P. G. Spirakis. Elements of the Theory of Dynamic Networks. *Communications of the ACM*, 61(2):72, 2018.

33 A. Nedić, A. Olshevsky, A. E. Ozdaglar, and J. N. Tsitsiklis. On Distributed Averaging Algorithms and Quantization Effects. *IEEE Transactions on Automatic Control*, 54(11):2506–2517, 2009.

34 R. O'Dell and R. Wattenhofer. Information Dissemination in Highly Dynamic Graphs. In *Proceedings of the 5th Joint Workshop on Foundations of Mobile Computing (DIALM-POMC '05)*, pages 104–110, 2005.

35 A. Olshevsky. Linear Time Average Consensus and Distributed Optimization on Fixed Graphs. *SIAM Journal on Control and Optimization*, 55(6):3990–4014, 2017.

36 A. Olshevsky and J. N. Tsitsiklis. Convergence Speed in Distributed Consensus and Averaging. *SIAM Journal on Control and Optimization*, 48(1):33–55, 2009.

37 A. Olshevsky and J. N. Tsitsiklis. A Lower Bound for Distributed Averaging Algorithms on the Line Graph. *IEEE Transactions on Automatic Control*, 56(11):2694–2698, 2011.

38 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

39 N. Sakamoto. Comparison of Initial Conditions for Distributed Algorithms on Anonymous Networks. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing (PODC '99)*, pages 173–179, 1999.

40 T. Sharma and M. Bashir. Use of Apps in the COVID-19 Response and the Loss of Privacy Protection. *Nature Medicine*, 26(8):1165–1167, 2020.

41 J. N. Tsitsiklis. *Problems in Decentralized Decision Making and Computation*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1984.

42 M. Yamashita and T. Kameda. Computing on Anonymous Networks. I. Characterizing the Solvable Cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.

43 Y. Yuan, G.-B. Stan, L. Shi, M. Barahona, and J. Goncalves. Decentralised Minimum-Time Consensus. *Automatica*, 49(5):1227–1235, 2013.

An Oracle with no UP-Complete Sets, but NP = PSPACE

David Dingel
Julius-Maximilians-Universität Würzburg, Germany

Fabian Egidy
Julius-Maximilians-Universität Würzburg, Germany

Christian Glaßer
Julius-Maximilians-Universität Würzburg, Germany

Abstract

We construct an oracle relative to which NP = PSPACE, but UP has no many-one complete sets. This combines the properties of an oracle by Hartmanis and Hemachandra [25] and one by Ogiwara and Hemachandra [38].

The oracle provides new separations of classical conjectures on optimal proof systems and complete sets in promise classes. This answers several questions by Pudlák [43], e.g., the implications $\mathsf{UP} \implies \mathsf{CON}^{\mathsf{N}}$ and $\mathsf{SAT} \implies \mathsf{TFNP}$ are false relative to our oracle.

Moreover, the oracle demonstrates that, in principle, it is possible that TFNP-complete problems exist, while at the same time SAT has no p-optimal proof systems.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Oracles and decision trees; Theory of computation → Complexity theory and logic

Keywords and phrases Computational Complexity, Promise Classes, Complete Sets, Oracle Construction

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.50

Funding *Fabian Egidy*: supported by the German Academic Scholarship Foundation.

Acknowledgements We thank the referees for their helpful feedback. Special thanks to the anonymous reviewer who pointed us to related literature and brought several open questions to our attention.

1 Introduction

We investigate relationships between the following classical conjectures on NP, optimal proof systems, and complete sets in promise classes. They are deeply rooted in formal logic, but also have far reaching practical implications. Each of these conjectures remains open.

$\mathsf{P} \neq \mathsf{NP}$:	P does not equal NP [12, 35]
$\mathsf{NP} \neq \mathsf{coNP}$:	NP does not equal coNP [16]
CON:	p-optimal proof systems for TAUT do not exist [34]
$\mathsf{CON}^{\mathsf{N}}$:	optimal proof systems for TAUT do not exist [34]
SAT:	p-optimal proof systems for SAT do not exist [34]
TFNP:	TFNP does not contain many-one complete problems [36]
$\mathsf{NP} \cap \mathsf{coNP}$:	NP ∩ coNP does not contain many-one complete problems [30]
UP:	UP does not contain many-one complete problems [25]
DisjNP:	DisjNP does not contain many-one complete pairs [45]
$\mathsf{DisjCoNP}$:	DisjCoNP does not contain many-one complete pairs [37, 42]

We refer to Pudlák [41] for a comprehensive elaboration of the conjectures and their context.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 50; pp. 50:1–50:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

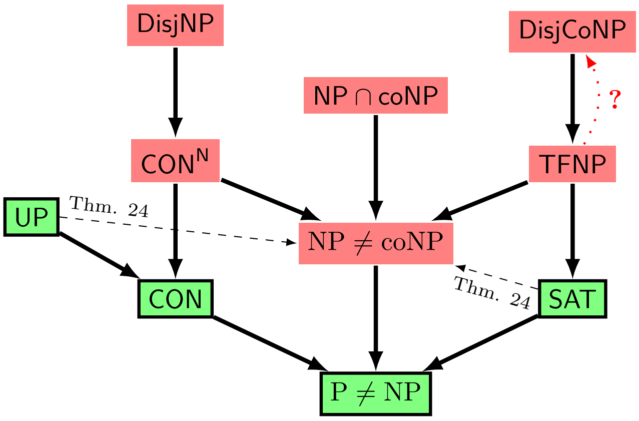


Figure 1 Hypotheses that are true relative to our oracle are filled green and have borders, whereas those that are false relative to the oracle are red without borders. Solid arrows mean relativizable implications. A dashed arrow from one conjecture A to another conjecture B means that there is an oracle X against the implication A ⇒ B, meaning that A ∧ ¬B holds relative to X. For clarity, we only depict the two strongest separations proved in this paper, and omit those that either follow from these, as well as those already known before. All possible separations have now been achieved, except for the one depicted as the red dotted arrow.

Pudlák's program. The known implications between these conjectures are shown in Figure 1. They raise the question of whether further implications can be recognized with the currently available methods. Pudlák [43] therefore initiated a research program to either prove further implications or to disprove them relative to an oracle. The latter shows that the implication is beyond the reach of current methods. Hence, from today's perspective, the corresponding conjectures are not mere reformulations of one another, but instead deserve individual attention.

So far several implications have been disproved relative to an oracle. The following selection of oracles covers all previously known disproofs of implications, in the sense that each such implication fails relative to at least one of these oracles:

$\mathsf{CON}^{\mathsf{N}} \wedge \neg\mathsf{DisjNP}$	(Glaßer, Selman, Sengupta, Zhang [20])
$\mathsf{DisjCoNP} \wedge \neg\mathsf{CON}$	(Khaniki [31])
$\mathsf{DisjNP} \wedge \mathsf{NP} \cap \mathsf{coNP} \wedge \neg\mathsf{UP}$	(Dose, Glaßer [15])
$\mathsf{NP} \cap \mathsf{coNP} \wedge \neg\mathsf{CON}$	(Dose [13])
$\mathrm{P} \neq \mathrm{NP} \wedge \neg\mathsf{CON} \wedge \neg\mathsf{SAT}$	(Dose [13])
$\mathsf{DisjNP} \wedge \mathsf{UP} \wedge \mathsf{NP} \cap \mathsf{coNP} \wedge \neg\mathsf{SAT}$	(Dose [14])
$\mathsf{DisjNP} \wedge \mathsf{UP} \wedge \mathsf{DisjCoNP} \wedge \neg\mathsf{NP} \cap \mathsf{coNP}$	(Egidy, Ehrmanntraut, Glaßer [17])

Optimal proof systems. The hypotheses CON, CON$^{\mathsf{N}}$, and SAT are concerned with the existence of (p-)optimal proof systems. The study of proof systems was initiated by Cook and Reckhow [11] and motivated by the NP $\overset{?}{=}$ coNP question, because the set of tautologies TAUT has a proof system with polynomially bounded proofs if and only if NP = coNP. Krajíček and Pudlák [34, 33, 44] link questions about proof systems to questions in bounded arithmetic. Especially the notions of optimal and p-optimal proof systems have gained much attention in research. They are closely connected to the separation of fundamental complexity classes, e.g., Köbler, Messner, and Torán [32] show that the absence of optimal proof systems for TAUT implies NEE ≠ coNEE. Moreover, (p-)optimal proof systems have tight connections to promise classes, e.g., if TAUT has p-optimal proof systems, then UP has many-one complete sets [32]. Many more relationships are known between proof systems and promise classes [3, 4, 5, 6, 7, 8, 21, 22, 32, 43, 45].

TFNP-complete problems. The class TFNP contains a multitude of problems that are of central importance to various practical applications, and as such is currently being researched with great intensity. The search for a complete problem in TFNP is of primary concern to certain fields such as cryptography, but it remains open whether such a problem exists.

For instance, the security of RSA [46] is primarily based on the hardness of integer factorization, which is a TFNP problem. However, no solid argument is known that factorization actually is a particularly hard problem within TFNP, i.e., being able to solve factorization might not require being able to solve arbitrary TFNP problems. RSA is now widely considered to no longer represent the state of the art in modern cryptography. Hence researchers develop cryptoschemes whose security is based on the hardness of other problems. However, it is still not clear whether these are strictly harder than factorization. A TFNP-complete problem would allow cryptoschemes that are at least as hard to break as any TFNP problem, i.e., schemes with optimal security guarantees.

Since the search for TFNP-complete problems has been unsuccessful, various subclasses were defined and studied in order to construct at least a complete problem for those subclasses. Their definitions are based on the proof of totality used to show the membership to TFNP. Prominent examples are PLS [29], based on the argument that *every dag has a sink*, PPA [39], based on the fact that *every graph has an even number of nodes with odd degree*, PPAD [39] and PPADS, based on slight variations of PPA for directed graphs, and PPP [40], based on the polynomial pigeonhole principle.[1] All of them have complete problems [29, 40] and can be defined in a syntactical way too [40]. Beame et al. [2] constructed oracles relative to which these classes are not a subset of P, and no inclusions exist beyond the few that are already known. The classes are significant to various practical applications like polynomially hard one way functions [9, 19], collision resistant hash functions [23], computing square roots modulo n [28], and the security of prominent homomorphic encryption schemes [27].

Our Contribution

1. Oracle with NP = PSPACE, but UP has no many-one complete sets.

Hartmanis and Hemachandra [25] construct an oracle relative to which UP does not have many-one complete sets. Ogiwara and Hemachandra [38] construct an oracle relative to which UP $\neq$ NP = PSPACE. Our oracle provides both properties at the same time, but this is achieved using completely different methods. Due to the far-reaching collapse NP = PSPACE and the close connection between UP-completeness and optimality of proof systems, we obtain a number of useful properties summarized in Corollary 25.

2. Consequences for Pudlák's program.

Regarding the conjectures shown in Figure 1, all known implications are presented in the figure, while for some of the remaining implications there exist oracles relative to which the implications do not hold. From the implications that were left open, our oracle refutes all but one. The key to this observation was to take the conjecture NP $\neq$ coNP into consideration. As we make sure that our oracle satisfies NP = coNP, the conjectures $\mathsf{CON}^{\mathsf{N}}$ and TFNP are

[1] For precise definitions we refer to recent papers [23], [24, notably Figure 1].

false, while the conjectures CON and SAT are equivalent. In addition, the oracle satisfies the conjecture UP, which implies CON, and thereby in summary refutes all of the following previously open implications:

$$\begin{array}{rlrlrl}
\mathsf{UP} \Longrightarrow \mathsf{CON}^{\mathsf{N}} & & \mathsf{UP} & \Longrightarrow \mathrm{NP} \neq \mathrm{coNP} & \mathsf{UP} & \Longrightarrow \mathsf{DisjNP} \\
\mathsf{CON} \Longrightarrow \mathsf{CON}^{\mathsf{N}} & & \mathsf{CON} & \Longrightarrow \mathrm{NP} \neq \mathrm{coNP} & \mathsf{SAT} & \Longrightarrow \mathsf{DisjCoNP} \\
\mathsf{SAT} \Longrightarrow \mathsf{TFNP} & & \mathsf{SAT} & \Longrightarrow \mathrm{NP} \neq \mathrm{coNP} & &
\end{array}$$

3. Consequences for TFNP-complete problems.
Regarding the search for TFNP-complete problems, our oracle demonstrates that, in principle, it is possible that NP = coNP and hence TFNP-complete problems exist, while at the same time SAT has no p-optimal proof systems.

Open questions

Hemaspaandra, Jain, and Vereshchagin [26] improve the oracle of Hartmanis and Hemachandra [25]. They show the existence of an oracle relative to which FewP [1] does not have Turing-hard sets for UP [26, Thm. 3.1]. They note that the theorem still holds when replacing the class FewP by Few [10]. Since UP $\subseteq$ FewP $\subseteq$ Few and many-one reducibility implies Turing reducibility, this raises the following questions (in ascending order of difficulty):
Does there exist an oracle relative to which

1. NP = PSPACE and UP has no set that is Turing-hard for UP?
2. NP = PSPACE and FewP has no set that is Turing-hard for UP?
3. NP = PSPACE and Few has no set that is Turing-hard for UP?

Regarding Pudlák's research program, all implications that are not presented in Figure 1 fail relative to some oracle, except for one single implication, which we leave as open question:

$$\mathsf{TFNP} \overset{?}{\Longrightarrow} \mathsf{DisjCoNP}$$

We suspect that such construction of an oracle with TFNP and ¬DisjCoNP is a particular challenge.

2 Preliminaries

2.1 Basic Definitions and Notations

This section covers the notation of this paper and introduces well-known complexity theoretic notions.

Words and sets. All sets and machines in this paper are defined over the alphabet $\Sigma = \{0, 1\}$. Σ^* denotes the set of all strings of finite length, and for a string $w \in \Sigma^*$ let $|w|$ denote the length of w. We write $\mathbb{N}$ for the set of non-negative integers, $\mathbb{N}^+$ for the set of positive integers, and $\emptyset$ for the empty set. For a set A and $k \in \mathbb{N}$ we write $A^{=k} := \{x \in A \mid |x| = k\}$ as the subset of A containing only words of length k. We define this analogously for $\leq$. For a clearer notation, we use the abbreviations $\Sigma^k := \Sigma^{*=k}$ and $\Sigma^{\leq k} := \Sigma^{*\leq k}$. Let $<_{\mathrm{lex}}$ denote the quasi-lexicographic (i.e., "shortlex") ordering of words over Σ^*.

Functions. Let enc be a polynomial-time computable, polynomial-time invertible order-isomorphism between $(\Sigma^*, <_{\mathrm{lex}})$ and $(\mathbb{N}, <)$, i.e., a variant of the dyadic representation. The domain and range of a function f are denoted by $\mathrm{dom}(f)$ and $\mathrm{ran}(f)$. For a function f and $A \subseteq \mathrm{dom}(f)$, we define $f(A) \coloneqq \{f(x) \mid x \in A\}$. We define certain polynomial functions $p_i \colon \mathbb{N} \to \mathbb{N}$ for $i \in \mathbb{N}$ by $p_i(n) \coloneqq n^i + i$ for $i \in \mathbb{N}^+$, and $p_i(n) \coloneqq 1$ for $i = 0$.

Let $\langle\cdot\rangle \colon \bigcup_{i \geq 0}(\Sigma^*)^i \to \Sigma^*$ be an injective, polynomial-time computable and polynomial-time invertible sequence encoding function[2] such that $|\langle u_1, \ldots, u_n\rangle| = 2(|u_1|+\cdots+|u_n|+n)+1$. The sequence encoding function has the following property.

▷ Claim 1. Let $w, x \in \Sigma^*$. If $|w| \geq |x| + 3$, then $|w| \geq |\langle x, w\rangle|/4$.

Proof. The following calculation proves the claimed statement:

$$|\langle x, w\rangle| = 2(|x| + |w| + 2) + 1 = 2|x| + 5 + 2|w| \leq 2|w| + 2|w| = 4|w| \qquad \lhd$$

Machines. We use the default model of a Turing machine in the deterministic as well as in the non-deterministic variant, abbreviated by DTM, resp., NTM. The language decided by a Turing machine M is denoted by $L(M)$. We use Turing transducers to compute functions. For a Turing transducer F we write $F(x) = y$ when on input x the transducer outputs y. We sometimes refer to the function computed by F as 'the function F'.

Complexity Classes and Reductions. The classes P, NP, coNP, FP and PSPACE denote the standard complexity classes. Furthermore, we are interested in several promise classes. Originally defined by Valiant [50], UP denotes the set of languages that can be decided by so called UP-machines, i.e., NTMs that have at most one accepting path on every input. The common polynomial-time many-one reducibility for sets $A, B \subseteq \Sigma^*$ is denoted by $\leq_{\mathrm{m}}^{\mathrm{p}}$, i.e., $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ if there exists an $f \in \mathrm{FP}$ such that $x \in A \Leftrightarrow f(x) \in B$. If $\leq$ is some notion of reducibility for some class $\mathcal{C}$, then we call a set A $\leq$-complete for $\mathcal{C}$, if $A \in \mathcal{C}$ and for all $B \in \mathcal{C}$ the reduction $B \leq A$ holds.

Oracle-specific definitions and notations. An oracle B is a subset of Σ^*. We relativize the concept of Turing machines and Turing transducers by giving them access to a write-only oracle tape as proposed by Simon [48][3]. Furthermore, we relativize complexity classes, proof systems, reducibilities and (p-)simulation by defining them over machines with oracle access, i.e., whenever a Turing machine or Turing transducer is part of a definition, we replace them by an oracle Turing machine or an oracle Turing transducer. We indicate the access to some oracle B in the superscript of the mentioned concepts, i.e., P^B, NP^B, FP^B, ... for complexity classes, M^B for a Turing machine or Turing transducer M, and $\leq_{\mathrm{m}}^{\mathrm{p},B}$ for reducibility. We sometimes omit the oracles in the superscripts, e.g., when sketching ideas in order to convey intuition, but never in the actual proof.

Let $\{F_i\}_{i \in \mathbb{N}}$ be a standard enumeration of polynomial-time oracle Turing transducers, such that relative to any oracle B, F_i^B has running time exactly p_i and for any function $f \in \mathrm{FP}^B$, there is some i such that F_i^B computes f. Since the functions computed by $\{F_i^B\}_{i \in \mathbb{N}}$ form exactly FP^B, we call such machines FP^B-machines. Similarly, let $\{N_i\}_{i \in \mathbb{N}}$ be

[2] Notice that the sequence encoding function explicitly is not surjective, so invertibility is meant in the sense that $\mathrm{ran}(\langle\cdot\rangle) \in \mathrm{P}$, and there is a function $f \in \mathrm{FP}$ such that $\langle f(y)_1, \ldots, f(y)_n\rangle = y$ for all $y \in \mathrm{ran}(\langle\cdot\rangle)$.

[3] When considering time-bounded computation, this machine model reflects the usual relativization of Turing machines. For space-bounded computations, the oracle tape is also subject to the space-bound.

a standard enumeration of polynomial-time non-deterministic oracle Turing machines, such that relative to any oracle B, N_i^B has running time exactly p_i on each path and for any set $L \in \mathrm{NP}^B$, there is some i such that $L(N_i^B) = L$. Since the sets decided by $\{N_i^B\}_{i\in\mathbb{N}}$ form exactly NP^B, we call such machines NP^B-machines. If p is an encoding of a computation path, then $Q(p)$ denotes the set of oracle queries on p. For a Turing transducer F, an NTM N and some word x, let $Q(N(F(x))$ denote the set of oracle queries of the computations $F(x)$ and $N(F(x))$. Hence, we interpret $N(F(x))$ as one computation which computes $F(x)$ first, followed by the computation N on input $F(x)$.

Relativized QBF. For any oracle B, we define the relativized problem QBF^B as the typical QBF problem as defined by Shamir [47], but the boolean formulas are extended by expressions $B(w)$ for $w \in \Sigma^*$, which evaluate to true if and only if $w \in B$. It holds that for any oracle B, QBF^B is $\leq_{\mathrm{m}}^{\mathrm{p},B}$-complete for PSPACE^B.

Throughout the remaining sections, M will be a polynomial-space oracle Turing machine which, given access to oracle B, accepts QBF^B. Choose $k \geq 3$ such that M on input $x \in \Sigma^*$ completes using at most $t(x) := |x|^k + k$ space (including queries to the oracle).

2.2 Notions for the Oracle Construction

In this section we define all necessary objects and properties for the oracle construction and convey their intuition.

Tools for UP. In order to achieve that UP^Φ has no complete set, we will ensure that for any set $L(N_i^\Phi) \in \mathrm{UP}^\Phi$ a set $W_i^\Phi \in \mathrm{UP}^\Phi$ exists such that W_i^Φ does not reduce to $L(N_i^\Phi)$, i.e., $W_i^\Phi \not\leq_{\mathrm{m}}^{\mathrm{p},\Phi} L(N_i^\Phi)$. Here, W_i^Φ is a witness that $L(N_i^\Phi)$ is not complete for UP^Φ. For this, we injectively assign countably infinitely many levels of the oracle to each set W_i, where a level consists of certain words of the same length as follows.

► **Definition 2** (H_i).
Let $e(0) := 2,\ e(i+1) := 2^{e(i)}$ and $H_i := \{e(2^i \cdot 3^j) \mid j \in \mathbb{N}\}$ for $i \in \mathbb{N}$.

► **Corollary 3** (Properties of H_i).

(i) *For every $i \in \mathbb{N}$, the set H_i is countably infinite and a subset of the even numbers.*
(ii) *$H_i \in \mathrm{P}$ for all $i \in \mathbb{N}$.*
(iii) *For $i, j \in \mathbb{N}$ with $i \neq j$, it holds that $H_i \cap H_j = \emptyset$.*

► **Definition 4** (Witness language W_i^B).
For $i \in \mathbb{N}$ and an oracle B, we define the set

$$W_i^B := \{0^n \mid n \in H_i \text{ and there exists } x \in \Sigma^n \text{ with } x \in B\}$$

► **Corollary 5** (Sufficient condition for $W_i^B \in \mathrm{UP}^B$).
For any oracle B and any $i \in \mathbb{N}$, the following implication holds:

$$|B^{=n}| \leq 1 \text{ for all } n \in H_i \Longrightarrow W_i^B \in \mathrm{UP}^B$$

Proof. Let B and i be arbitrary. Since $H_i \in \mathrm{P}$, an NP-machine can reject on all inputs except 0^n with $n \in H_i$. On these inputs 0^n the machine can non-deterministically query all words $x \in \Sigma^n$ and accept if $x \in B$. Now, this machine accepts W_i^B, and if $|B^{=n}| \leq 1$, then it has at most one accepting path on all inputs. Hence, $W_i^B \in \mathrm{UP}^B$. ◄

We can control the membership of W_i to UP in the oracle construction. We will never add more than one word on the levels H_i, except for when we can rule out N_i as a UP-machine.

Tools for NP = PSPACE. In order to achieve that $\mathrm{NP}^\Phi = \mathrm{PSPACE}^\Phi$, we will encode QBF^Φ into the oracle Φ, such that $\mathrm{QBF}^\Phi \in \mathrm{NP}^\Phi$, from which $\mathrm{NP}^\Phi = \mathrm{PSPACE}^\Phi$ follows.

▶ **Definition 6** (Coding set Z^B).
For any oracle B, we define the set

$$Z^B := \{x \in \Sigma^* \mid \exists w \in \Sigma^{t(x)} \colon \langle x, w\rangle \in B\}$$

▶ **Corollary 7.** *For any oracle B, it holds that $Z^B \in \mathrm{NP}^B$.*

We will assemble the oracle Φ in such a way that $Z^\Phi = \mathrm{QBF}^\Phi$, i.e., Z^Φ will be PSPACE^Φ-hard, and thus $\mathrm{NP}^\Phi = \mathrm{PSPACE}^\Phi$.

Goals of the construction.

▶ **Definition 8** (Desired properties of Φ).
The later constructed oracle Φ should satisfy the following properties:

P1 $\forall x \in \Sigma^* \colon (x \in \mathrm{QBF}^\Phi \iff x \in Z^\Phi)$.

(Meaning: $\mathrm{QBF}^\Phi \in \mathrm{NP}^\Phi$*.)*

P2 $\forall i \in \mathbb{N}$, *at least one of the following statements holds:*

- **(I)** $\exists x \in \Sigma^* \colon N_i^\Phi(x)$ *accepts on more than one path.*

  (Meaning: N_i^Φ is not a UP^Φ*-machine.)*
- **(II)** *Both of the following statements hold:*
  - **(a)** $\forall n \in H_i \colon \ |\Phi^{=n}| \leq 1$

    (Meaning: $W_i^\Phi \in \mathrm{UP}^\Phi$.)
  - **(b)** $\forall j \in \mathbb{N}\ \exists x \in \Sigma^* \colon N_i^\Phi(F_j^\Phi(x))$ *accepts if and only if $x \notin W_i^\Phi$.*

    (Meaning: W_i^Φ does not reduce to $L(N_i^\Phi)$ via F_j^Φ.)

The following lemma shows that these properties imply our desired structural properties of complexity classes.

▶ **Lemma 9.** *Let $\Phi \subseteq \Sigma^*$.*

(i) *If statement* P1 *is satisfied relative to Φ, then* $\mathrm{NP}^\Phi = \mathrm{PSPACE}^\Phi$*.*

(ii) *If statement* P2 *is satisfied relative to Φ, then there is no $\leq_\mathrm{m}^{\mathrm{p},\Phi}$-complete set for* UP^Φ*.*

Proof. To (i): Recall that QBF^Φ is $\leq_\mathrm{m}^{\mathrm{p},\Phi}$-complete for PSPACE^Φ. Since property P1 holds relative to Φ, $Z^\Phi = \mathrm{QBF}^\Phi$. From Corollary 7 we get $Z^\Phi \in \mathrm{NP}^\Phi$ and thus $\mathrm{NP}^\Phi = \mathrm{PSPACE}^\Phi$.

To (ii): Assume there is a $\leq_\mathrm{m}^{\mathrm{p},\Phi}$-complete set C for UP^Φ. Let N_i^Φ be a UP^Φ-machine such that $L(N_i^\Phi) = C$. Then property P2.I cannot hold for i, because N_i would not be a UP-machine. Hence, property P2.II holds for i. By P2.II.a and Corollary 5, we get $W_i^\Phi \in \mathrm{UP}^\Phi$. Since $L(N_i^\Phi)$ is complete, there is some $j \in \mathbb{N}$ such that for all $x \in \Sigma^*$ we have

$$x \in W_i^\Phi \iff F_j^\Phi(x) \in L(N_i^\Phi).$$

This is a contradiction to the satisfaction of property P2.II.b, because there has to be some x where this equivalence does not hold. Hence, the assumption has to be false and there is no $\leq_\mathrm{m}^{\mathrm{p},\Phi}$-complete set for UP^Φ. ◀

Stages. We choose certain stages on which we will deal with property P2.II.b. They are chosen as the range of a certain function m. We will call the values of m 'UP-stages', since only on these we will contribute to enforcing property P2.II.b and consequently UP^{Φ}. For now, it suffices to know that the stages will be chosen 'sufficiently large'.

We choose a total injective function $m : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, $(i, j) \mapsto m(i, j)$ which meets the following requirements:

M1 $m(i, j) \in H_i$.

(Meaning: UP-stages $m(i, \cdot)$ are important to the witness-language W_i.)

M2 $2^{m(i,j)/4} > 4(p_i(p_j(m(i, j))))^2 + 2p_i(p_j(m(i, j))) + 1$.

(Meaning: The number of words of length $m(i, j)$ is far bigger than the number of words the computation $N_i(F_j(0^{m(i,j)}))$ can query.)

M3 $m(i, j) > m(i', j') \implies m(i, j) > p_{i'}(p_{j'}(m(i', j')))$.

(Meaning: The stages are sufficiently distant from another such that for no $i', j' \in \mathbb{N}$, the computation $N_{i'}(F_{j'}(0^{m(i',j')}))$ can query any word of the UP-stages following $m(i', j')$.)

▶ **Observation 10.** *For all $k \in \operatorname{ran}(m)$: $\operatorname{ran}(\langle\cdot\rangle) \cap \Sigma^k = \emptyset$, because $\langle\cdot\rangle$ only maps to odd lengths. In particular, if $|x| = m(i, j)$ for some $i, j \in \mathbb{N}$, then $x \notin \operatorname{ran}(\langle\cdot\rangle)$.*

3 Oracle Construction

In this section we will construct an oracle Φ such that UP^{Φ} has no complete set and $\mathrm{NP}^{\Phi} = \mathrm{PSPACE}^{\Phi}$.

Construction of Φ. We construct the oracle Φ sequentially. For each $x \in \Sigma^*$, we decide whether to add some words to the oracle. We give a brief sketch of the construction and its ideas:

To P1: Whenever the input x has the form $\langle x', 0^{t(x')}\rangle$, we add $\langle x', w\rangle$ for some $w \in \Sigma^{t(x')}$ to the oracle if and only if $x' \in \mathrm{QBF}$. Since $M(x')$ cannot query words of length $|\langle x', w\rangle|$, the membership of x' to QBF does not change after adding $\langle x', w\rangle$. From then on, we never add shorter words to Φ, thus the encoding persists correct for the finished oracle.

To P2: On every UP-stage $n := m(i, j)$, we try to achieve property P2.I or P2.II. For this, we differentiate between three cases.

1. If $N_i(F_j(0^n))$ accepts relative to the oracle constructed so far, we leave the stage n empty. In subsequent iterations, we ensure that added words will not interfere with this accepting path. This achieves property P2.II, because $0^n \notin W_i$ and $F_j(0^n) \in L(N_i)$.
2. If $N_i(F_j(0^n))$ rejects relative to the oracle constructed so far, we add a word of length n to the oracle such that this computation keeps rejecting. This achieves property P2.II, because $0^n \in W_i$ and $F_j(0^n) \notin L(N_i)$. Notice that finding appropriate words may not be possible.
3. If $N_i(F_j(0^n))$ rejects relative to the oracle constructed so far, and there is no choice of a word to add such that $N_i(F_j(0^n))$ keeps rejecting (i.e., case 2 is not possible), then we force $N_i(F_j(0^n))$ to accept on two different paths (which is possible, as we will show). In subsequent iterations, we ensure that added words will not interfere with these accepting paths. This achieves property P2.I.

In iterations after a UP-stage, if we have to add words to maintain P1, we choose them in such a way that property P2 still remains upheld for the previous stage. This is captured in Procedure 3, `handle_UP_stage_aftercare`. Once the next stage is reached, our word length will have increased enough such that the stage before cannot be affected anymore.

▶ **Definition 11** (Oracle construction).
Define $\Phi_0 := \emptyset$*. For* $k \in \mathbb{N}^+$*, define* $\Phi_{k+1} := \Phi_k \cup \mathtt{construct}(\mathrm{enc}(k), \Phi_k)$ *according to Procedure 1. Finally, define* $\Phi := \bigcup_{k \in \mathbb{N}} \Phi_k$*.*

Procedure 1 `construct(x,B)`.

```
1  If x = $\langle$x',$0^{t(x')}\rangle$ and $M^B$(x') accepts:
2      If the largest UP-stage m(i,j) < |x| exists:
3          Return handle_UP_stage_aftercare(x',i,j,B)
4      Return {x}
5  Elif x = $0^{m(i,j)}$ for some i,j $\in \mathbb{N}$:
6      Return handle_UP_stage(i,j,B)
7  Else:
8    Return $\emptyset$.
```

Procedure 2 `handle_UP_stage(i,j,B)`.

```
1  n := m(i,j)
2  If $N_i^B(F_j^B(0^n))$ accepts:
3    Return $\emptyset$
   // Here, $N_i^B(F_j^B(0^n))$ rejects
4  If $\exists y \in \Sigma^n$ s.th. $N_i^{B\cup\{y\}}(F_j^{B\cup\{y\}}(0^n))$ rejects:
5    Return {y}
6  Else:
7    Let y,z $\in \Sigma^n$ s.th. $N_i^{B\cup\{y,z\}}(F_j^{B\cup\{y,z\}}(0^n))$ accepts on $\geq 2$ paths
8    Return {y,z}
```

Procedure 3 `handle_UP_stage_aftercare(x',i,j,B)`.

```
1  n := m(i,j)
2  If $N_i^B(F_j^B(0^n))$ accepts on at least one path $p_1$:
3    $Q := \begin{cases} Q(p_1) \cup Q(p_2) & \text{if } N_i^B(F_j^B(0^n)) \text{ accepts on another path } p_2 \neq p_1 \\ Q(p_1) & \text{else} \end{cases}$
4    Choose w $\in \Sigma^{t(x')}$ s.th. $\langle$x',w$\rangle \notin$ Q
5    Return {$\langle$x',w$\rangle$}
   // Here, $N_i^B(F_j^B(0^n))$ rejects
6  If $\exists y \in \{\langle x',w\rangle \mid w \in \Sigma^{t(x')}\}$ s.th. $N_i^{B\cup\{y\}}(F_j^{B\cup\{y\}}(0^n))$ rejects:
7    Return {y}
8  Let $y,z \in \{\langle x',w\rangle \mid w \in \Sigma^{t(x')}\}$ s.th. $N_i^{B\cup\{y,z\}}(F_j^{B\cup\{y,z\}}(0^n))$ accepts on $\geq 2$ paths
9  Return {y,z}
```

Notice that in `handle_UP_stage` reaching line 3 corresponds to case 1 of the construction sketch for P2, reaching line 5 corresponds to case 2, and reaching lines 7–8 corresponds to case 3. In lines 2–4 of `construct`, because M accepts, we need to add at least one word $\langle x', w\rangle$ to the oracle in order to achieve P1. Procedure 3, `handle_UP_stage_aftercare`, ensures that a previous UP-stage either remains unaffected by this (lines 2–7), or at least i now satisfies property P2.I instead (lines 8–9).

It is not clear that the oracle construction can be performed as stated, because lines 4 and 8 in `handle_UP_stage_aftercare` and line 7 in `handle_UP_stage` claim the existence of words with complex properties. The following claims attend to this problem.

▷ Claim 12 (Line 7 in `handle_UP_stage` can be performed).
If the line 7 is reached in `handle_UP_stage` in the step $\mathtt{construct}(\mathrm{enc}(k), \Phi_k)$, then y and z can be chosen as stated.

Proof. Let $x \coloneqq \mathrm{enc}(k)$. When reaching this line, then for some $i, j \in \mathbb{N}$ with $n \coloneqq m(i,j)$ we have that $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ rejects, and for all extensions $\Phi_k \cup \{x'\}$ with $x' \in \Sigma^n$, $N_i^{\Phi_k \cup \{x'\}}(F_j^{\Phi_k \cup \{x'\}}(0^n))$ accepts. Since the accepting paths appear after adding x' to Φ_k, these paths must query x'. There are 2^n words of length n and an accepting path can query at most $p_i(p_j(n)) + p_j(n) \leq 2p_i(p_j(n))$ of these words. Let X be a set consisting of $4(p_i(p_j(n)))^2 + 2p_i(p_j(n)) + 1$ pairwise different words x' of length n. By M2,

$$2^{n/4} > 4(p_i(p_j(n)))^2 + 2p_i(p_j(n)) + 1,$$

whereby such a set exists. We show that there must be two different words $y, z \in X$ such that $N_i^{\Phi_k \cup \{y\}}(F_j^{\Phi_k \cup \{y\}}(0^n))$ (resp., $N_i^{\Phi_k \cup \{z\}}(F_j^{\Phi_k \cup \{z\}}(0^n))$) accepts and does not query z (resp., y) on some accepting path.

To choose y, we look at $X' \subseteq X$ consisting of $2p_i(p_j(n)) + 1$ pairwise different words. When extending Φ_k by a word from X' for each word in X' separately, the resulting leftmost accepting paths query at most

$$2p_i(p_j(n)) \cdot |X'| = 2p_i(p_j(n)) \cdot (2p_i(p_j(n)) + 1) = 4(p_i(p_j(n)))^2 + 2p_i(p_j(n))$$

different words in total. Hence, there is at least one unqueried word $y \in X$.

To choose z, consider $N_i^{\Phi_k \cup \{y\}}(F_j^{\Phi_k \cup \{y\}}(0^n))$, which on the leftmost accepting path queries at most $p_i(p_j(n)) + p_j(n) \leq 2p_i(p_j(n))$ words. Hence, there is some unqueried word $z \in X' \subseteq X$.

So, $N_i^{\Phi_k \cup \{y\}}(F_j^{\Phi_k \cup \{y\}}(0^n))$ accepts on a path which queries y but not z, and $N_i^{\Phi_k \cup \{z\}}(F_j^{\Phi_k \cup \{z\}}(0^n))$ accepts on a path which queries z but not y (because $z \in X'$). Thus, $y \neq z$, both accepting paths are different and both are preserved when extending by both y and z, resulting in at least two different accepting paths for $N_i^{\Phi_k \cup \{y,z\}}(F_j^{\Phi_k \cup \{y,z\}}(0^n))$. ◁

▷ Claim 13 (Line 8 in `handle_UP_stage_aftercare` can be performed).
If the line 8 is reached in `handle_UP_stage_aftercare` in the step `construct`$(\mathrm{enc}(k), \Phi_k)$, then y and z can be chosen as stated.

Proof. Let $x \coloneqq \mathrm{enc}(k)$. When reaching this line, then for $x = \langle x', 0^{t(x')} \rangle$ and the biggest UP-stage $n \coloneqq m(i,j) \leq |x|$ we have that $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ rejects, but for all extensions $\Phi_k \cup \{y\}$ with $y \in \{\langle x', w \rangle \mid w \in \Sigma^{t(x')}\} =: Y$, the computation $N_i^{\Phi_k \cup \{y\}}(F_j^{\Phi_k \cup \{y\}}(0^n))$ accepts. Since the accepting paths appear after adding y to Φ_k, these paths must query y. By Claim 1, $|t(x')| \geq |x|/4 \geq n/4$. Hence, there are $\geq 2^{n/4}$ words $w \in \Sigma^{t(x')}$, and thus $|Y| \geq 2^{n/4}$. An accepting path can query at most $p_i(p_j(n)) + p_j(n) \leq 2p_i(p_j(n))$ of these words. Let X be a set consisting of $4(p_i(p_j(n)))^2 + 2p_i(p_j(n)) + 1$ pairwise different words $y \in Y$. By M2,

$$2^{n/4} > 4(p_i(p_j(n)))^2 + 2p_i(p_j(n)) + 1,$$

whereby such a set exists. From here on, we can proceed exactly as in the proof of Claim 12 to find two fitting words y and z. ◁

▷ Claim 14 (Line 4 in `handle_UP_stage_aftercare` can be performed).
If the line 4 is reached in `handle_UP_stage_aftercare` in the step `construct`$(\mathrm{enc}(k), \Phi_k)$, then w can be chosen accordingly.

Proof. Let $x \coloneqq \mathrm{enc}(k)$. When reaching this line, then for $x = \langle x', 0^{t(x')} \rangle$ and the biggest UP-stage $n \coloneqq m(i,j) \leq |x|$ it holds that $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ accepts. In the worst case, there are at least two accepting paths. Up to two accepting paths p_1 and p_2 query at most $2p_i(p_j(n)) + 2p_j(n) \leq 4p_i(p_j(n))$ words in total. Together with M2, we get

$$|Q(p_1) \cup Q(p_2)| \leq 4p_i(p_j(n)) \leq 4(p_i(p_j(n)))^2 + 2p_i(p_j(n)) < 2^{n/4}.$$

By Claim 1, $|t(x')| \geq |x|/4 \geq n/4$ and thus

$$|\Sigma^{t(x')}| \geq 2^{n/4} > 4(p_i(p_j(n)))^2 + 2p_i(p_j(n)) + 1 \geq |Q(p_1) \cup Q(p_2)|.$$

Consequently, there is some $w \in \Sigma^{t(x')}$ with $\langle x', w\rangle \notin (Q(p_1) \cup Q(p_2))$. ◁

The Claims 12, 13, and 14 show that $\Phi_{k+1} := \Phi_k \cup \texttt{construct}(\mathrm{enc}(k), \Phi_k)$ is well-defined. We make three further observations about `construct`.

▶ **Observation 15.** *For $k \in \mathbb{N}$, $\texttt{construct}(\mathrm{enc}(k), \Phi_k)$ only adds words of length $|\mathrm{enc}(k)|$.*

The next two observations follow from Observation 15 combined with Observation 10.

▶ **Observation 16.** *Let $n \in H_i$ for some $i \in \mathbb{N}$ and $k := \mathrm{enc}^{-1}(0^n)$.*
Then only the step $\texttt{construct}(0^n, \Phi_k)$ can add words of length n to Φ.

▶ **Observation 17.** *Let $x := \langle x', 0^{t(x')}\rangle$ for $x' \in \Sigma^*$ and $k := \mathrm{enc}^{-1}(x)$.*
Then only the step $\texttt{construct}(x, \Phi_k)$ can add words of the form $\langle x', w\rangle$ with $w \in \Sigma^{t(x')}$ to Φ.

Proving the Properties. Finally, we show that the properties P1 and P2 hold relative to Φ. We start with property P1.

▶ **Lemma 18.** *The oracle Φ satisfies property* P1*.*

Proof. First, recall that by definition of Z,

$$x \in Z^\Phi \iff \exists w \in \Sigma^{t(x)} \colon \langle x, w\rangle \in \Phi \tag{1}$$

Let $x \in \Sigma^*$ be arbitrary, and consider the step of the oracle construction where Φ_k is defined as $\texttt{construct}(\langle x, 0^{t(x)}\rangle, \Phi_{k-1})$. By Observation 17, only $\texttt{construct}(\langle x, 0^{t(x)}\rangle, \Phi_{k-1})$ may add words of the form $\langle x, w\rangle$ for $w \in \Sigma^{t(x)}$ to Φ. From this we can draw $x \notin Z^{\Phi_{k-1}}$, and further, together with $\Phi_k \subseteq \Phi$,

$$\exists w \in \Sigma^{t(x)} : \langle x, w\rangle \in \Phi \iff \exists w \in \Sigma^{t(x)} \colon \langle x, w\rangle \in \Phi_k \tag{2}$$

The lines 5 and 6 in $\texttt{construct}(\langle x, 0^{t(x)}\rangle, \Phi_{k-1})$ are skipped, since, by M1, $m(i,j)$ is always even, whereas $\langle x, 0^{t(x)}\rangle$ has odd length (see Observation 10). Hence, `handle_UP_stage` will not be entered. Since at the lines 2–4 of `construct` always at least one word is added to the oracle, $\texttt{construct}(\langle x, 0^{t(x)}\rangle, \Phi_{k-1})$ adds a word $\langle x, w\rangle$ with $w \in \Sigma^{t(x)}$ to Φ_{k-1} if and only if the condition in line 1 is met and these lines are entered, i.e., if $M^{\Phi_{k-1}}(x)$ accepts. This gives

$$x \in \mathrm{QBF}^{\Phi_{k-1}} \iff \exists w \in \Sigma^{t(x)} \colon \langle x, w\rangle \in \Phi_k \tag{3}$$

Since $M^{\Phi_{k-1}}(x)$ has a space bound of $t(x)$, it cannot ask words longer than $t(x)$. Thereby $M^{\Phi_{k-1}}(x)$ cannot notice a transition to the oracle Φ_k, because we only add words $\langle x, w\rangle$ with $w \in \Sigma^{t(x)}$, which have length $> t(x)$. So, $M^{\Phi_k}(x)$ accepts if and only if $M^{\Phi_{k-1}}(x)$ accepts. For the same reason, $M^{\Phi_k}(x)$ cannot notice a transition to the oracle Φ, because by Observation 15, all words in the set $\Phi \setminus \Phi_k$ have length $\geq |\langle x, 0^{t(x)}\rangle|$, and thus $M^\Phi(x)$ accepts if and only if $M^{\Phi_k}(x)$ accepts. Combining these two facts gives

$$x \in \mathrm{QBF}^\Phi \iff x \in \mathrm{QBF}^{\Phi_{k-1}} \tag{4}$$

In total, we conclude

$$\begin{aligned} x \in \mathrm{QBF}^{\Phi} &\overset{(4)}{\iff} x \in \mathrm{QBF}^{\Phi_{k-1}} \overset{(3)}{\iff} \exists w \in \Sigma^{t(x)} \colon \langle x, w\rangle \in \Phi_k \\ &\overset{(2)}{\iff} \exists w \in \Sigma^{t(x)} \colon \langle x, w\rangle \in \Phi \\ &\overset{(1)}{\iff} x \in Z^{\Phi} \end{aligned}$$

Since $x \in \Sigma^*$ was chosen arbitrarily, property P1 holds. ◀

Before proving Lemma 21, i.e., proving that property P2 holds, we state the helpful claim that `construct` does not alter certain accepting paths.

▷ Claim 19. Let $i, j, k \in \mathbb{N}$ such that $k > \mathrm{enc}^{-1}(0^{m(i,j)})$, and $n := m(i,j)$. If $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ accepts (resp., accepts on more than one path), then so does $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$.

Proof. If $\mathrm{enc}(k) \geq_{\mathrm{lex}} 0^{p_i(p_j(n))+1}$, then by Observation 15, only words of length $> p_i(p_j(n))$ are added to Φ_k. Hence, all paths of $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ and $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$ are the same, from which the claimed statement follows.

Otherwise, $\mathrm{enc}(k) \leq_{\mathrm{lex}} 1^{p_i(p_j(n))}$. By M3, $m(i,j)$ has to be the biggest UP-stage $\leq |\mathrm{enc}(k)|$. Consider the step `construct`$(\mathrm{enc}(k), \Phi_k)$. Either $\Phi_{k+1} = \Phi_k$ and the claim holds, or some words are added to Φ_k. The procedure `handle_UP_stage` is not entered, because $\mathrm{enc}(k) >_{\mathrm{lex}} 0^n$, so the condition in line 5 of `construct` is false, whereby any added words would need to be added by `handle_UP_stage_aftercare`. Further, since $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ accepts, this can only happen via the lines 3 to 5. Here, line 4 makes sure that $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$ remains accepting (resp., remains accepting on more than one path), since the respective paths do not query the chosen words. ◁

▶ **Corollary 20.** *Let $i, j, k \in \mathbb{N}$ such that $k > \mathrm{enc}^{-1}(0^{m(i,j)})$, and $n := m(i,j)$. If $N_i^{\Phi_k}(F_j^{\Phi_k}(0^n))$ accepts (resp., accepts on more than one path), then so does $N_i^{\Phi}(F_j^{\Phi}(0^n))$.*

▶ **Lemma 21.** *The oracle Φ satisfies property* P2*.*

Proof. Let $i \in \mathbb{N}$ be arbitrary. Assume that for i the property P2.I does not hold. Then we show that property P2.II holds.

▷ Claim 22. The property P2.II.a holds.

Proof. Let $n \in H_i$ be arbitrary and $k := \mathrm{enc}^{-1}(0^n)$. By Observation 16, only the step `construct`$(0^n, \Phi_k)$ adds words of length n to Φ. Since n has even and $\langle \cdot, \cdot \rangle$ has always odd length (see Observation 10), the condition in line 1 of `construct` evaluates to 'false'. So, only if line 8 in `handle_UP_stage` is executed, more than one word of length n is added to Φ_k and consequently to Φ. In this case, since the sets $H_0, H_1, \ldots$ are disjoint, there is some $j \in \mathbb{N}$ with $m(i,j) = n$ where $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$ accepts on more than one path. Invoking Corollary 20 with i, j and $k+1$, we get that also $N_i^{\Phi}(F_j^{\Phi}(0^n))$ accepts on more than one path. This is a contradiction to the assumption that the property P2.I does not hold for i. Hence, line 8 in `handle_UP_stage` cannot be executed during `construct`$(0^n, \Phi_k)$. This gives $|\Phi^{=n}| \leq 1$, as required. ◁

▷ Claim 23. The property P2.II.b holds.

Proof. Let $j \in \mathbb{N}$ be arbitrary, $n := m(i,j)$, and $k := \mathrm{enc}^{-1}(0^n)$. Consider the step `construct`$(0^n, \Phi_k)$. Since n has even length, the condition in line 1 of `construct` evaluates to 'false'. But the condition in line 5 is satisfied, so `handle_UP_stage` will define which words are added to the oracle. Also recall that only at this step words of length n are added to Φ (Observation 16), i.e., $\Phi_{k+1}^{=n} = \Phi^{=n}$. We distinguish between the three cases on how Φ_{k+1} can be defined in `handle_UP_stage`.

If Φ_{k+1} is defined via line 3, then $\Phi^{=n} = \emptyset$ and $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$ accepts. By Corollary 20, $N_i^{\Phi}(F_j^{\Phi}(0^n))$ accepts too. Hence, $N_i^{\Phi}(F_j^{\Phi}(0^n))$ accepts and $0^n \notin W_i^{\Phi}$, i.e., 0^n satisfies property P2.II.b.

If Φ_{k+1} is defined via line 8, then $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$ accepts on more than one path. By Corollary 20, $N_i^{\Phi}(F_j^{\Phi}(0^n))$ also accepts on more than one path. But then, property P2.I holds for i, a contradiction to the assumption at the start of Lemma 21. Hence, this case cannot occur.

If Φ_{k+1} is defined by line 5, then $|\Phi^{=n}| = 1$ and $N_i^{\Phi_{k+1}}(F_j^{\Phi_{k+1}}(0^n))$ rejects. Let $k' > k+1$ be the smallest number such that $N_i^{\Phi_{k'}}(F_j^{\Phi_{k'}}(0^n))$ accepts. Either k' does not exist and hence, $N_i^{\Phi}(F_j^{\Phi}(0^n))$ rejects, satisfying property P2.II.b.

Otherwise $\mathrm{enc}(k') \leq_{\mathrm{lex}} 1^{p_i(p_j(n))}$, because $N_i(F_j(0^n))$ can query only words of length $\leq p_i(p_j(n))$, and for bigger k', only words of length $> p_i(p_j(n))$ are added (Observation 15). Consider the step `construct`$(\mathrm{enc}(k'-1), \Phi_{k'-1})$. By M3, $m(i,j)$ is the biggest UP-stage $\leq |\mathrm{enc}(k'-1)|$. Since

$$0^n = \mathrm{enc}(k) <_{\mathrm{lex}} \mathrm{enc}(k+1) \leq_{\mathrm{lex}} \mathrm{enc}(k'-1),$$

the condition in line 5 is not met, thus `handle_UP_stage` is skipped. Since by the minimality of k' the computation $N_i^{\Phi_{k'-1}}(F_j^{\Phi_{k'-1}}(0^n))$ rejects, $\Phi_{k'}$ can only be defined via line 9 in `handle_UP_stage_aftercare`. But then, $N_i^{\Phi_{k'}}(F_j^{\Phi_{k'}}(0^n))$ accepts on more than one path. Invoking Corollary 20 for i, j, k', we get that also $N_i^{\Phi}(F_j^{\Phi}(0^n))$ accepts on more than one path. Consequently, property P2.I holds for i, a contradiction to the assumption at the start of Lemma 21. Hence, this case cannot occur. ◁

Since i is arbitrary, the Claims 22 and 23 show that if property P2.I does not hold, property P2.II does. Thus, property P2 holds. ◀

4 Conclusion

The oracle from the previous section gives the following result.

▶ **Theorem 24.** *There exists an oracle relative to which* UP *has no* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete sets and* NP = PSPACE.

Proof. This follows from the Lemmas 9, 18, and 21. ◀

We summarize the properties of the oracle from Theorem 24 relating to the hypotheses of Pudlák [43] and mention a few more.

▶ **Corollary 25.** *Relative to the oracle from Theorem 24, all of the following hold:*
1. NP = coNP, *i.e.,* ¬NP ≠ coNP.
2. UP *does not have* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete sets, i.e.,* UP.
3. TAUT *does not have p-optimal proof systems, i.e.,* CON.
4. SAT *does not have p-optimal proof systems, i.e.,* SAT.
5. TFNP *has a complete problem, i.e.,* ¬TFNP.

6. TAUT *has optimal proof systems, i.e.,* $\neg\mathsf{CON}^{\mathsf{N}}$.
7. DisjNP *has* $\leq_{\mathrm{m}}^{\mathrm{pp}}$*-complete pairs, i.e.,* $\neg\mathsf{DisjNP}$.
8. DisjCoNP *has* $\leq_{\mathrm{m}}^{\mathrm{pp}}$*-complete pairs, i.e.,* $\neg\mathsf{DisjCoNP}$.
9. NP ∩ coNP *has* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-complete sets, i.e.,* $\neg\mathsf{NP} \cap \mathsf{coNP}$.
10. $\mathrm{P} \subsetneq \mathrm{UP} \subsetneq \mathrm{NP} \cap \mathrm{coNP} = \mathrm{NP}$.
11. *The conjecture Q does not hold* [18, Def. 2].
12. *The conjecture Q′ does not hold* [18, Def. 3].
13. $\mathrm{TFNP} \not\subseteq_c \mathrm{PF}$.

Proof. To 1 and 2: Follows from Theorem 24.
To 3: Follows from 2 by Köbler, Messner, and Torán [32, Cor. 4.1].
To 4: Follows from 3 and 1.
To 5: Follows from 1 by Megiddo and Papadimitrou [36, Thm. 2.1].
To 6: Follows from 1 and the fact that NP has optimal proof systems [37, Thm. 3.1].
To 7: Follows from 6 by Köbler, Messner, Torán [32, Cor. 6.1].
To 8: Follows from 7 and 1.
To 9: Follows from 1 and that NP^{Φ} has $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete sets.
To 10: Follows from P ⊆ UP ⊆ NP, P has complete sets, UP^{Φ} does not have complete sets (2), NP has complete sets and $\mathrm{NP}^{\Phi} = \mathrm{coNP}^{\Phi}$ (1).
To 11 and 12: Follows from $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ (which is implied by 10), as shown by Fenner et al. [18, Thm. 2,Prop. 9].
To 13: Follows from 11, as shown by Fenner et al. [18, Prop. 7]. ◀

References

1 E. W. Allender and R. S. Rubinstein. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.

2 P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 303–314, New York, NY, USA, 1995. Association for Computing Machinery. doi:10.1145/225058.225147.

3 O. Beyersdorff. Representable disjoint NP-pairs. In *Proceedings 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*, pages 122–134. Springer, 2004. doi:10.1007/978-3-540-30538-5_11.

4 O. Beyersdorff. Disjoint NP-pairs from propositional proof systems. In *Proceedings of Third International Conference on Theory and Applications of Models of Computation*, volume 3959 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2006. doi:10.18452/15520.

5 O. Beyersdorff. Classes of representable disjoint NP-pairs. *Theoretical Computer Science*, 377(1-3):93–109, 2007. doi:10.1016/j.tcs.2007.02.005.

6 O. Beyersdorff. The deduction theorem for strong propositional proof systems. *Theory of Computing Systems*, 47(1):162–178, 2010. doi:10.1007/s00224-008-9146-6.

7 O. Beyersdorff, J. Köbler, and J. Messner. Nondeterministic functions and the existence of optimal proof systems. *Theoretical Computer Science*, 410(38-40):3839–3855, 2009. doi:10.1016/j.tcs.2009.05.021.

8 O. Beyersdorff and Z. Sadowski. Do there exist complete sets for promise classes? *Mathematical Logic Quarterly*, 57(6):535–550, 2011. doi:10.1002/malq.201010021.

9 N. Bitansky, O. Paneth, and A. Rosen. On the cryptographic hardness of finding a nash equilibrium. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1480–1498, 2015. doi:10.1109/FOCS.2015.94.

10 J. Cai and L. Hemachandra. On the power of parity polynomial time. *Mathematical systems theory*, 23:95–106, 1990. doi:10.1007/BF02090768.

11 S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979. doi:10.2307/2273702.

12 S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. Association for Computing Machinery. doi:10.1145/800157.805047.

13 T. Dose. Further oracles separating conjectures about incompleteness in the finite domain. *Theoretical Computer Science*, 847:76–94, 2020. doi:10.1016/j.tcs.2020.09.040.

14 T. Dose. An oracle separating conjectures about incompleteness in the finite domain. *Theoretical Computer Science*, 809:466–481, 2020. doi:10.1016/j.tcs.2020.01.003.

15 T. Dose and C. Glaßer. NP-completeness, proof systems, and disjoint NP-pairs. In C. Paul and M. Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.9.

16 J. Edmonds. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240, 1966.

17 A. Ehrmanntraut, F. Egidy, and C. Glaßer. Oracle with P = NP ∩ coNP, but No Many-One Completeness in UP, DisjNP, and DisjCoNP. In S. Szeider, R. Ganian, and A. Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.45.

18 S. A. Fenner, L. Fortnow, A. V. Naik, and J. D. Rogers. Inverting onto functions. *Information and Computation*, 186(1):90–103, 2003. doi:10.1016/S0890-5401(03)00119-6.

19 S. Garg, O. Pandey, and A. Srinivasan. Revisiting the cryptographic hardness of finding a nash equilibrium. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, pages 579–604, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

20 C. Glaßer, A. L. Selman, S. Sengupta, and L. Zhang. Disjoint NP-pairs. *SIAM Journal on Computing*, 33(6):1369–1416, 2004. doi:10.1137/S0097539703425848.

21 C. Glaßer, A. L. Selman, and L. Zhang. Canonical disjoint NP-pairs of propositional proof systems. *Theoretical Computer Science*, 370:60–73, 2007. doi:10.1007/11549345_35.

22 C. Glaßer, A. L. Selman, and L. Zhang. The informational content of canonical disjoint NP-pairs. *International Journal of Foundations of Computer Science*, 20(3):501–522, 2009. doi:10.1007/978-3-540-73545-8_31.

23 P. W. Goldberg and C. H. Papadimitriou. TFNP: An update. In D. Fotakis, A. Pagourtzis, and V. Th. Paschos, editors, *Algorithms and Complexity*, pages 3–9, Cham, 2017. Springer International Publishing.

24 M. Göös, A. Hollender, S. Jain, G. Maystre, W. Pires, R. Robere, and R. Tao. Further Collapses in TFNP. In S. Lovett, editor, *37th Computational Complexity Conference (CCC 2022)*, volume 234 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2022.33.

25 J. Hartmanis and L. A. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science*, 58:129–142, 1988.

26 L. A. Hemaspaandra, S. Jain, and N. K. Vereshchagin. Banishing robust turing completeness. *International Journal of Foundations of Computer Science*, 04(03):245–265, 1993. doi:10.1142/S012905419300016X.

27 R. Jawale, Y. T. Kalai, D. Khurana, and R. Zhang. Snargs for bounded depth computations and ppad hardness from sub-exponential lwe. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 708–721, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3406325.3451055.

28 E. Jeřábek. Integer factoring and modular square roots. *arXiv e-prints*, page arXiv:1207.5220, July 2012. doi:10.48550/arXiv.1207.5220.

29 D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

30 R. Kannan, 1979. Sipser [49] cites an unpublished work by Kannan for asking if there is a set complete for NP ∩ coNP.

31 E. Khaniki. New relations and separations of conjectures about incompleteness in the finite domain. *The Journal of Symbolic Logic*, 87(3):912–937, 2022. doi:10.1017/jsl.2021.99.

32 J. Köbler, J. Messner, and J. Torán. Optimal proof systems imply complete sets for promise classes. *Information and Computation*, 184(1):71–92, 2003. doi:10.1016/S0890-5401(03)00058-0.

33 J. Krajíček. *Bounded Arithmetic, Propositional Logic and Complexity Theory.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1995. doi:10.1017/CBO9780511529948.

34 J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *Journal of Symbolic Logic*, 54:1063–1079, 1989. doi:10.2307/2274765.

35 L. A. Levin. Universal search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973. doi:https://www.mathnet.ru/ppi914.

36 N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. doi:10.1016/0304-3975(91)90200-L.

37 J. Messner. *On the Simulation Order of Proof Systems.* PhD thesis, Universität Ulm, 2000.

38 M. Ogiwara and L.A. Hemachandra. A complexity theory for feasible closure properties. In *[1991] Proceedings of the Sixth Annual Structure in Complexity Theory Conference*, pages 16–29, 1991. doi:10.1109/SCT.1991.160240.

39 C. H. Papadimitriou. On inefficient proofs of existence and complexity classes. In J. Neŝetril and M. Fiedler, editors, *Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity*, volume 51 of *Annals of Discrete Mathematics*, pages 245–250. Elsevier, 1992. doi:10.1016/S0167-5060(08)70637-X.

40 C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

41 P. Pudlák. *Logical foundations of mathematics and computational complexity: A gentle introduction.* Springer, 2013.

42 P. Pudlák. On some problems in proof complexity. In O. Beyersdorff, E. A. Hirsch, J. Krajíček, and R. Santhanam, editors, *Optimal algorithms and proofs (Dagstuhl Seminar 14421)*, volume 4, pages 63–63, Dagstuhl, Germany, 2014. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/DagRep.4.10.51.

43 P. Pudlák. Incompleteness in the finite domain. *The Bulletin of Symbolic Logic*, 23(4):405–441, 2017.

44 P. Pudlák. The lengths of proofs. In S. R. Buss, editor, *Handbook of Proof Theory*, pages 547–637. Elsevier, Amsterdam, 1998.

45 A. Razborov. On provably disjoint NP-pairs. *BRICS Report Series*, 36, 1994. doi:10.7146/brics.v1i36.21607.

46 R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. doi:10.1145/359340.359342.

47 A. Shamir. IP=PSPACE (interactive proof=polynomial space). In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 11–15 vol.1, 1990. doi:10.1109/FSCS.1990.89519.

48 I. Simon. *On Some Subrecursive Reducibilities.* PhD thesis, Stanford University, 1977.

49 M. Sipser. On relativization and the existence of complete sets. In *Proceedings 9th ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 523–531. Springer Verlag, 1982. doi:10.1007/BFb0012797.

50 L. G. Valiant. Relative complexity of checking and evaluation. *Information Processing Letters*, 5:20–23, 1976. doi:10.1016/0020-0190(76)90097-1.

Half-Space Separation in Monophonic Convexity

Mohammed Elaroussi
Université de Bejaia, Faculté des Sciences Exactes, Unité de Recherche LaMOS,
06000 Bejaia, Algeria

Lhouari Nourine
Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS,
63000 Clermont-Ferrand, France

Simon Vilmin
Aix-Marseille Université, CNRS, LIS, Marseille, France

Abstract

We study half-space separation in the convexity of chordless paths of a graph, i.e., monophonic convexity. In this problem, one is given a graph and two (disjoint) subsets of vertices and asks whether these two sets can be separated by complementary convex sets, called half-spaces. While it is known this problem is NP-complete for geodesic convexity – the convexity of shortest paths – we show that it can be solved in polynomial time for monophonic convexity.

2012 ACM Subject Classification Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Graph algorithms

Keywords and phrases chordless paths, monophonic convexity, separation, half-space

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.51

Related Version *Full Version*: `https://arxiv.org/abs/2404.17564` [13]

Funding The first two authors have been funded by the CMEP Tassili project: "Argumentation et jeux: Structures et Algorithmes", Codes: 46085QH-21MDU320 PHC, 2021-2023. This research is also supported by the French government IDEXISITE initiative 16-IDEX-0001 (CAP 20-25).

Acknowledgements We thank the reviewers for their comments. We are also grateful to Victor Chepoi for suggesting the term "shadow" instead of "extension" as well as pointing us to further references, especially the recent works on monophonic convexity [4, 8].

1 Introduction

A (finite) convexity space is a pair $(V, \mathcal{C})$ where V is a (finite) groundset and $\mathcal{C}$ a collection of subsets of V, called *convex sets*, containing $\emptyset$, V and closed under taking intersections. Graphs provide a wide variety of different convexity notions, known as graph convexities. These are usually defined based on paths and include for instance the geodesic convexity [20], the monophonic convexity [10, 12, 15], the m^3-convexity [11], the triangle-path convexity [5], the toll convexity [1], or the weakly toll convexity [9].

In this paper, we are interested in the *half-space separation* problem: with an implicitly given convexity space $(V, \mathcal{C})$ and two (convex) subsets A, B of V, are there complementary convex sets $H, \overline{H}$ – the so-called *half-spaces* – such that $A \subseteq H$ and $B \subseteq \overline{H}$? This problem is a generalization to abstract convexity of the half-space separation problem in $\mathbb{R}^d$, being well-studied in machine learning [3, 16, 25]. Half-space separation has motivated the study of structural separation properties of convexity spaces. Among these properties, two have received particular attention, notably within graph convexities (see e.g. [2, 7, 14, 18, 23]): the S_3 property stating that any convex set C can be separated from any element of V not in C; and the S_4 or Kakutani property stating that any pair of disjoint convex sets can be separated. Besides, the study of the half-space separation problem on its own has recently been brought to the context of graph convexities [21, 22]. In particular, Seiffarth et al. [21] show that half-space separation is NP-complete for geodesic convexity, the convexity induced by the shortest paths of a graph. To our knowledge though, the complexity status of half-space separation for the other aforementioned graph convexities is still unknown.

In our contribution, we follow this latter line of research and study half-space separation for the monophonic convexity. Given a graph G with vertices $V(G)$, a set $C \subseteq V(G)$ is monophonically convex if for any two vertices u, v of C, all the vertices on a chordless path u and v lie in C. We prove that half-space separation can be decided in polynomial time for monophonic convexity. More formally, our main theorem reads as follows:

▶ **Theorem 1.** *Given a graph G and two subsets A, B of $V(G)$, whether A, B are separated by monophonic half-spaces can be decided in polynomial time.*

Theorem 1 contrasts with the NP-completeness of half-space separation for geodesic convexity [21] and suggests to study separation in further graph convexities. Besides, half-space separation also relates to the p-partition problem (in graph convexities). In the p-partition problem, one is given a graph G and has to decide whether $V(G)$ can be partitioned into p convex sets, where the meaning of convex depends on the convexity at hand. For monophonic convexity, Gonzalez et al. [17] show that p-partition is NP-complete for $p \geq 3$, but they leave open the case $p = 2$. Since 2-partition is possible if and only if there exists two separable vertices, Theorem 1 proves that 2-partition can be decided in polynomial time.

▶ Remark 2. In very recent contributions, Chepoi [8] and Bressan et al. [4] also showed that half-space separation can be decided in polynomial time for monophonic convexity. Their results have been obtained independently and using different approaches, even though they share some common points with the technique used in this paper.

Organization of the paper

In Section 2 we provide definitions, notations and we formally define the problem we investigate in the paper. In Section 3 we prove Theorem 1 by giving an algorithm which decides whether two sets can be separated by half-spaces. We conclude in Section 4.

▶ Remark 3. Due to space limitations, most proofs are omitted. All of them can found in the `arXiv` version of the present contribution [13].

2 Preliminaries

All the objects considered in this paper are finite. Let V be a set. The powerset of V is denoted $\mathbf{2}^V$. Given $X \subseteq V$, we write $\overline{X}$ the complement of X in V, i.e., $\overline{X} = V \setminus X$. Sometimes, we shall write a set X as the concatenation of its elements, e.g., uv instead of $\{u, v\}$. As a result, $X \cup uv$ and $X \cup v$ stands for $X \cup \{u, v\}$ and $X \cup \{v\}$ respectively.

Graphs

We assume the reader is familiar with standard graph terminology. We consider loopless undirected graphs. Let G be a graph with vertices $V(G)$ and edge set $E(G)$. A subgraph of G is any graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The *(open) neighborhood* of a vertex v in G is denoted $N(v)$ and is defined as $N(v) = \{u \in V(G) : uv \in E(G)\}$. The *closed neighborhood* of v in G is $N[v] = N(v) \cup v$. For $X \subseteq V(G)$, we put similarly $N(X) = \{u \in V(G) \setminus X : xu \in E(G) \text{ for some } x \in X\}$ and $N[X] = N(X) \cup X$. The subgraph of G *induced* by X is $G[X] = (X, E(G[X]))$, where $E(G[X]) = \{uv \in E(G) : u, v \in X\}$. If this is clear from the context, we identify X with $G[X]$, and we use $G - X$ to denote $G[V(G) - X]$. A *path* in G is a subgraph P of G with $V(P) = \{v_1, \ldots, v_k\}$ and such that $v_i v_{i+1} \in E(P)$ for each $1 \leq i < k$. Putting $u = v_1$ and $v = v_k$, P is an *uv-path* of G. An *induced uv-path* or *chordless uv-path* of G is an induced subgraph of G being an uv-path. A shortest path is an induced path with the least possible number of vertices. For simplicity we will identify a path P with the sequence $v_1, \ldots, v_k$ of its vertices. Let $A, B \subseteq V(G)$ be non-empty. The *(inner) frontier* of A with respect to B is $F(A, B) = A \cap N[B]$. We note that if A, B are disjoint, we obtain $F(A, B) = A \cap N(B)$. Remark that for every $X \subseteq V(G)$, $F(\overline{X}, X) = N(X)$.

Convexity spaces

We refer the reader to [24] for a thorough introduction to convexity theory. A *convexity space* is a pair $(V, \mathcal{C})$, with $\mathcal{C} \subseteq \mathbf{2}^V$, such that $\emptyset, V \in \mathcal{C}$ and for every $C_1, C_2 \in \mathcal{C}$, $C_1 \cap C_2 \in \mathcal{C}$. The sets in $\mathcal{C}$ are *convex* sets. A convexity space $(V, \mathcal{C})$ induces a *(convex) hull* operator $h \colon \mathbf{2}^V \to \mathbf{2}^V$ defined for all $X \subseteq V$ by:

$$h(X) = \bigcap \{C \in \mathcal{C} : X \subseteq C\}$$

The operator h satisfies, for all $X, Y \subseteq V$: $X \subseteq h(X)$; $h(X) \subseteq h(Y)$ if $X \subseteq Y$; and $h(h(X)) = h(X)$. The *Carathéodory number* of $(V, \mathcal{C})$ is the least integer d such that for every $X \subseteq V$ and $v \in V$, if $v \in h(X)$, there exists a subset Y of X such that $v \in h(Y)$ and $|Y| \leq d$. A *half-space* of $(V, \mathcal{C})$ is a convex set H whose set complement $\overline{H}$ is convex, that is, $H, \overline{H} \in \mathcal{C}$. Let A, B be two subsets of V. We say that A and B are *(half-space) separable* if there exists half-spaces $H, \overline{H}$ satisfying $A \subseteq H$ and $B \subseteq \overline{H}$. This is equivalent to $h(A) \subseteq H$ and $h(B) \subseteq \overline{H}$. The *shadow of A with respect to B* [6, 7] is the set $A/B = \{v \in V : h(B \cup v) \cap A \neq \emptyset\}$. Observe that $A \subseteq A/B$ and $B \subseteq B/A$.

▶ Remark 4. Usually, A/B is defined for disjoint sets. Here, it is more convenient to extend this definition to sets that may intersect. If $A \cap B \neq \emptyset$, then $A/B = V$ vacuously.

Monophonic convexity

We introduce monophonic convexity. We redirect the reader to [24, 20] for further details on graph and interval convexities. Let G be a graph, and let $u, v \in V(G)$. The *monophonic closed interval* of u, v is the set of all vertices that lie on a chordless uv-path, denoted by $J[u, v]$. For $X \subseteq V(G)$, we put $J[X] = \bigcup_{u,v \in X} J[u, v]$. A set C is *monophonically convex* if $J[C] = C$. Throughout the paper, if there is no ambiguity, we use the term convex sets as a shortening for monophonically convex sets. With $\mathcal{C} = \{C \subseteq V(G) : C \text{ is monophonically convex}\}$, the pair $(V(G), \mathcal{C})$ is a convexity space, the *monophonic convexity* of G. Its convex hull operator h is defined for all $X \subseteq V(G)$ by:

$$h(X) = \bigcup_{k=0}^{\infty} X_k$$

where $X_0 = X$ and $X_i = J[X_{i-1}]$ for $i \geq 1$. We now gather existing results regarding monophonic convexity that will be useful throughout the paper. These statements are rephrased to match our terminology.

▶ **Observation 5** (see also [12]). *In a connected graph G, every convex set is connected.*

▶ **Theorem 6** ([12], Theorem 5.1)**.** *The monophonic convexity of a connected graph has Carathéodory number is* 1 *if the graph is a clique and* 2 *otherwise.*

▶ **Theorem 7** ([10], Theorem 4.1)**.** *Let G be a graph and let $X \subseteq V(G)$. Then, $h(X)$ can be computed in polynomial time in the size of G.*

▶ **Theorem 8** ([10], Theorem 2.1)**.** *Let G be a connected graph and let $C \subseteq V$. The set C is convex if and only if for every connected component S of $G - C$, $F_G(C, S)$ is a clique.*

▶ **Lemma 9** ([17], Lemma 14)**.** *Let G be a connected graph, K a clique separator of G, and X the union of some of the connected components of $G - K$. Then $X \cup K$ is convex.*

We end these preliminaries by stating the problem we investigate in this paper. It is the problem of separating two sets of vertices by half-spaces. Its decision version is:

Half-space separation in monophonic convexity
Input: A graph G and two (non-empty and disjoint) subsets A, B of $V(G)$.
Question: Are A and B half-space separable?

Since h can be computed in polynomial time by Theorem 7 and A, B are separable if and only if $h(A)$, $h(B)$ are separable, we can assume w.l.o.g. that the sets A and B are convex.

3 Half-space separation

In this section, we prove Theorem 1, which we first restate.

▶ **Theorem 1.** *Given a graph G and two subsets A, B of $V(G)$, whether A, B are separated by monophonic half-spaces can be decided in polynomial time.*

Remark that if the input graph is not connected, one just has to solve the problem for each connected component. Thus, we can consider without loss of generality that the graphs we consider are connected. Hence, for the section, we fix a connected graph G. Let A, B be two (disjoint) convex sets of G. To prove Theorem 1, we give a polynomial time algorithm that decides whether A, B are separable. The algorithm first computes a shortest path $a = v_1, \ldots, v_k = b$ for some $a \in A$ and $b \in B$ in polynomial time. We show in Lemma 11 that A and B are separable if and only if there exists $1 \leq i < k$ such that $A_i := h(A \cup \{v_1, \ldots, v_i\})$ and $B_i := h(B \cup \{v_{i+1}, \ldots, v_k\})$ are separable. This step is the *linkage* of A and B. Then, for each i, the algorithm does the subsequent operations:

(1) It computes the *saturation* $A'_i := S(A_i, B_i)$, $B'_i := S(B_i, A_i)$ of A_i, B_i (resp.) with respect to h (see Subsection 3.2). Informally, the saturation step extends A_i and B_i with vertices that are forced on one of the two sides by the hull operator h. Lemma 13 shows that A_i, B_i are separable if and only if A'_i, B'_i are separable. Corollary 16 proves that computing saturation takes polynomial time.

(2) From A'_i and B'_i, it builds an equivalence relation $\equiv_{A'_iB'_i}$ on $\overline{A'_i \cup B'_i}$ and an associated graph $G_{A'_iB'_i}$. Theorem 31 states that A'_i and B'_i are separable if and only if $G_{A'_iB'_i}$ is bipartite and no equivalence class of $\equiv_{A'_iB'_i}$ contains a so-called *forbidden pair* of vertices. Finally, Theorem 32 proves that the conditions of Theorem 31 can be tested in polynomial time.

The algorithm outputs that A and B are separable if there is an integer i for which step (2) succeeds. Otherwise, A and B are not separable. The correctness of the algorithm follows from Lemma 11, Lemma 13 and Theorem 31. The fact that it runs in polynomial time is a consequence of Corollary 16 and Theorem 32. This proves Theorem 1.

The rest of the section is dedicated to the proof of the aforementioned statements.

3.1 Linkage along a shortest path

Let A, B be two non-empty disjoint convex subsets of $V(G)$. Assume that A and B are separable and let $H, \overline{H}$ be half-spaces separating A and B. Then for each $a \in A$, and each $b \in B$, all the vertices on the shortests ab-paths are distributed among H and $\overline{H}$. We show that, in fact, for each shortest ab-path, there is a vertex before which all vertices are assigned one half-space and all vertices after which are assigned the other half-space.

▶ **Proposition 10.** *Let $a \in A$, $b \in B$ and let $a = v_1, \dots, v_k = b$ be a shortest ab-path. For every half-space separation $H, \overline{H}$ of A and B with $A \subseteq H$ and $B \subseteq \overline{H}$, there exists $1 \leq i < k$ such that $\{v_1, \dots, v_i\} \subseteq H$ and $\{v_{i+1}, \dots, v_k\} \subseteq \overline{H}$.*

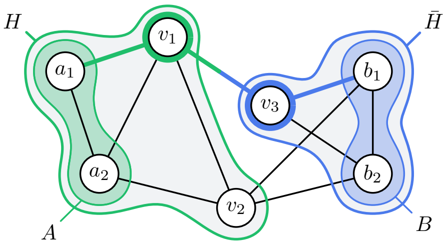


Figure 1 A graph G with two disjoint convex sets $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$ (circled in green and blue resp.). A and B are not linked, but they can be linked along the path a_1, v_1, v_3, b_1 (in bold green / bold blue). Namely, $A \cup v_1$ and $B \cup v_3$ are linked and convex. Two half-spaces $H, \overline{H}$ separating $A \cup v_1$ and $B \cup v_3$ (hence A and B) are drawn.

Following Proposition 10, we say that A and B are *linked* if there exists $a \in A$, $b \in B$ such that $ab \in E(G)$. Linked sets and Proposition 10 are illustrated in Figure 1. The next lemma is a direct consequence of Proposition 10.

▶ **Lemma 11.** *Let G be a connected graph and let A, B be two non-empty disjoint convex subsets of $V(G)$. Let $a \in A$, $b \in B$ and let $a = v_1, \dots, v_k = b$ be a shortest ab-path. Then, A and B are separable if and only if there exists $1 \leq i < k$ such that $h(A \cup \{v_1, \dots, v_i\})$ and $h(B \cup \{v_{i+1}, \dots, v_k\})$ are separable.*

Given $a \in A$ and $b \in B$, finding a shortest ab-path can be done in polynomial time. Hence, making A and B linked can be done efficiently. Moreover, if A and B are linked, then for any disjoint $A', B' \subseteq V$ such that $A \subseteq A'$ and $B \subseteq B'$, A' and B' must be linked too. In what follows, we will thus consider disjoint, convex and linked subsets of $V(G)$.

3.2 Saturation with the hull operator

Let A, B be two disjoint, linked and convex subsets of $V(G)$. In this part, we use the hull operator h to define two sets $S(A, B)$ and $S(B, A)$ – the saturation of A and B (see below) – with $A \subseteq S(A, B)$, $B \subseteq S(B, A)$ and such that A, B are separable if and only if their saturation is separable. Informally, we use h to identify vertices that will appear in the same half-space as A in any possible half-space separation of A (and similarly with B), if any. We use two properties built on h:

(1) **Shadow-closing.** Remind that A/B, the shadow of A with respect to B, is defined by $A/B = \{v \in V(G) : h(B \cup v) \cap A \neq \emptyset\}$. In particular, $A \subseteq A/B$.

(2) **Forbidden sets.** Let $X \subseteq \overline{A \cup B}$ and assume that $h(X) \cap A \neq \emptyset$ and $h(X) \cap B \neq \emptyset$. Since $h(v) = \{v\}$ for all $v \in V$, we have $|X| \geq 2$. Thus, separating A, B implies to split the vertices of X. We say that X is a *forbidden set* of A and B with respect to G. A set X is forbidden if and only if it includes an inclusion-wise minimal forbidden set as a subset. Henceforth, in order to use forbidden sets, we need only consider the family of inclusion-wise minimal forbidden sets, denoted $\mathrm{MFS}(A,B)$. Formally,

$$\mathrm{MFS}(A,B) = \min_{\subseteq}\{X \subseteq \overline{A \cup B} : h(X) \cap A \neq \emptyset \text{ and } h(X) \cap B \neq \emptyset\}.$$

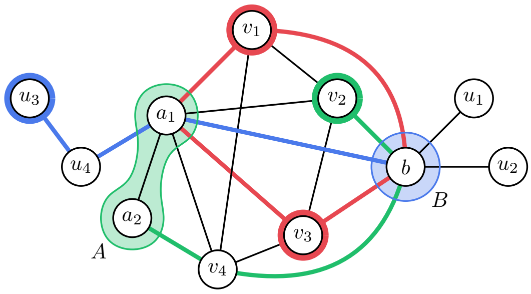


Figure 2 A graph G in which we seek to separate A and B (in circled green/blue resp.). The vertex a_1 is on a chordless u_3b-path (bold blue), so that $u_3 \in A/B$. Dually, b is on a chordless v_2a_2-path (bold green), i.e., $v_2 \in B/A$. Besides, $h(v_1v_3)$ intersects both A and B (bold red). Thus, v_1, v_3 must be separated to separate A and B, and $v_1v_3 \in \mathrm{MFS}(A,B)$ holds.

We illustrate shadows and forbidden sets in Figure 2. Now, we use A/B (resp. B/A) and $\mathrm{MFS}(A,B)$ in view of separating A and B. On the one hand, A/B cannot be separated from A by definition. On the other hand, for each $X \in \mathrm{MFS}(A,B)$ and every half-spaces $H, \bar{H}$ separating A and B with $A \subseteq H$, there exists at least one $x \in X$ such that $x \in H$, so that, $\bigcap_{x \in X} h(A \cup x) \subseteq H$ always hold. Based on the previous arguments, we define the *pre-saturation* of A with respect to B in G, denoted by $\sigma(A,B)$, by:

$$\sigma(A,B) = h\left(A/B \cup \bigcup \left\{ \bigcap_{x \in X} h(A \cup x) : X \in \mathrm{MFS}(A,B) \right\}\right)$$

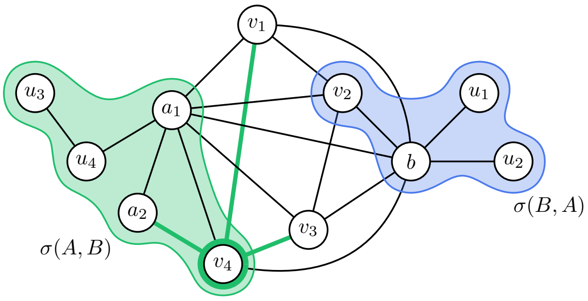
Observe that if $A \cap B \neq \emptyset$, then $\sigma(A,B) = \sigma(B,A) = V(G)$ as $A/B = B/A = V(G)$. In this case though, A and B cannot be separated. We prove in the next statement that $\sigma(A,B)$ preserves separation. Remark that it holds regardless of the disjointness of A and B.

▶ **Lemma 12.** *Let G be a connected graph, and let A, B be linked and convex subsets of $V(G)$. Then, A, B are separable if and only if $\sigma(A,B)$ and $\sigma(B,A)$ are separable.*

Proof. The if part follows from $A \subseteq A/B \subseteq \sigma(A,B)$ and $B \subseteq B/A \subseteq \sigma(B,A)$. We show the only if part. Suppose that A and B are separable and let $H, \overline{H}$ be half-spaces such that $A \subseteq H$, $B \subseteq \bar{H}$. Let $v \in A/B$. By definition, $h(B \cup v) \cap A \neq \emptyset$, hence $H \cap \overline{H} = \emptyset$ entails $v \in H$. Now let $X \in \mathrm{MFS}(A,B)$. By definition of forbidden sets, $X \cap H \neq \emptyset$ and $X \not\subseteq H$. Thus, there exists $x \in X$ such that $x \in H$, which entails $h(A \cup x) \subseteq H$ as H is convex. Since $\bigcap_{x' \in X} h(A \cup x') \subseteq h(A \cup x)$ for each $x \in X$, we deduce

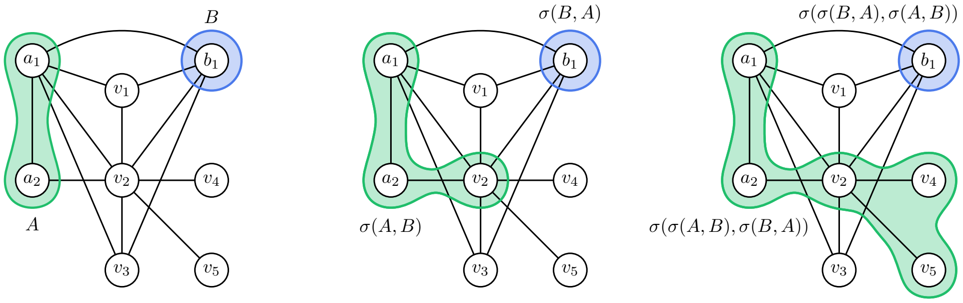
$$A/B \cup \bigcup \left\{ \bigcap_{x \in X} h(A \cup x) : X \in \mathrm{MFS}(A,B) \right\} \subseteq H.$$

As H is convex, we get $\sigma(A,B) \subseteq H$. Applying the symmetric reasoning on $\sigma(B,A)$ yields $\sigma(B,A) \subseteq \overline{H}$. This concludes the proof. ◀



■ **Figure 3** Pre-saturation applied to the sets A and B of Figure 2. For $\sigma(B,A)$, $u_1,u_2 \in B/A$ are added. For $\sigma(A,B)$, we have $u_3,u_4 \in A/B$ and $v_4 \in h(A\cup v_1)\cap h(A\cup v_3)$ (paths in bold green) with $v_1v_3 \in \mathrm{MFS}(A,B)$.

We illustrate pre-saturation in Figure 3, where the operation is applied to the set A and B of Figure 2. In this example, once pre-saturation has been applied, no further vertices can be assigned by applying pre-saturation once more. There are cases however where applying pre-saturation twice yields new vertices to assign. Figure 4 illustrates this situation.



■ **Figure 4** An example where pre-saturation can be applied twice. For $\sigma(A,B)$, v_2 is obtained from the forbidden pair v_1v_3. Once v_2 is added, v_4,v_5 become part of $\sigma(A,B)/\sigma(B,A)$. Observe that $B=\sigma(B,A)=\sigma(\sigma(B,A),\sigma(A,B))$. The remaining vertices v_1,v_3 can be separated in any way.

This suggests to iteratively apply the pre-saturation operator until no more vertices are added. For $A,B\subseteq V$, the *saturation* of A with respect to B, denoted by $S(A,B)$ is defined as follows:

$$S(A,B)=\bigcup_{i=0}^{\infty}\sigma(A_i,B_i)$$

where $A_0=A$, $B_0=B$ and for all $1\le i$, $A_i=\sigma(A_{i-1},B_{i-1})$ and $B_i=\sigma(B_{i-1},A_{i-1})$. Given $A,B\subseteq V(G)$, we say that A and B are *saturated* if $A=S(A,B)$ and $B=S(B,A)$. Since σ is increasing, the procedure for computing $S(A,B)$ terminates after $|V(G)|$ steps at most. Applying Lemma 12 inductively on $1\le i$, we get:

▶ **Lemma 13.** *Let G be a connected graph, and let A,B be two linked and convex subsets of $V(G)$. Then, A,B are separable if and only if $S(A,B)$, $S(B,A)$ are separable.*

▶ Remark 14. If $A_i\cap B_i\neq\emptyset$ for some i, then $S(A,B)=S(B,A)=V(G)$, and no separation can distinguish $S(A,B)$ and $S(B,A)$. In particular, A,B are thus not separable.

To conclude this paragraph, we argue that $S(A, B)$ can be computed in polynomial time in the size of G. Since S is at most $|V(G)|$ applications of σ on subsets of $V(G)$, it is sufficient to show that σ can be computed in polynomial time. The bottleneck of computing σ lies in finding $\mathrm{MFS}(A, B)$. However, the fact that the Carathéodory number of monophonic convexity is 2 by Theorem 6 makes the sets in $\mathrm{MFS}(A, B)$ of constant size.

▶ **Proposition 15.** *Given $A, B \subseteq V(G)$, $\sigma(A, B)$ can be computed in polynomial time in the size of G.*

▶ **Corollary 16.** *Given $A, B \subseteq V(G)$, $S(A, B)$ can be computed in polynomial time in the size of G.*

We note that saturation is not sufficient to decide separability, as suggested by Figure 5. This motivates the last step of the algorithm.

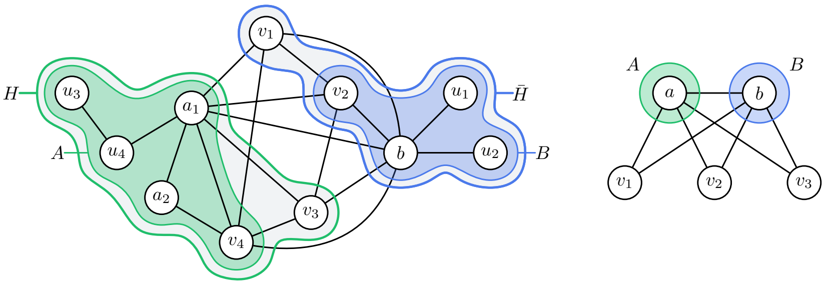


Figure 5 Two cases where A and B are linked, convex and saturated. On the left (follow-up of Figure 3), A and B can be separated (two half-spaces are drawn). On the right, any bipartition of the vertices will contain one of the forbidden pair v_1v_2, v_1v_3 or v_2v_3. Thus, A and B are not separable.

3.3 Testing bipartiteness

Let A, B be two linked, disjoint and saturated subsets of $V(G)$. By definition of saturation, A and B are convex. We characterize the separability of A and B using an equivalence relation $\equiv_{AB}$ on $\overline{A \cup B}$ and a graph G_{AB} defined from $\equiv_{AB}$. More precisely, we prove in Theorem 31 that A and B are separable if and only if G_{AB} is bipartite and no two $\equiv_{AB}$-equivalent vertices form a forbidden pair of $\mathrm{MFS}(A, B)$.

As a preliminary step though, we give properties of G and $N(A \cup B)$ in terms of A and B. We start with a statement that holds for every convex set.

▶ **Proposition 17.** *Let $C \subseteq V(G)$ be a convex set, and let u, v be two distinct vertices of $V(G) \setminus C$. Then:*

(1) *if u, v are not adjacent, then $h(uv) \cap C \neq \emptyset$ if and only if there exists $u', v' \in N(C)$ such that $u'v' \notin E(G)$ and $u', v' \in h(uv)$;*

(2) *if u, v are adjacent, then $F(C, u) \setminus F(C, v) \neq \emptyset$ entails $u \in h(C \cup v)$.*

Leveraging from the fact that A, B are linked and saturated, we use Proposition 17 to show that every vertex in $N(A \cup B)$ is adjacent to both A and B.

▶ **Lemma 18.** *For every $v \in N(A \cup B)$, $F(A, B) \cup F(B, A) \subseteq N(v)$. Therefore, the following properties hold for A (and symmetrically for B):*

(1) *$N(A) = F(B, A) \cup N(A \cup B)$;*

(2) *$F(A, \overline{A}) = F(A, N(A \cup B))$ is a clique.*

Proof. Assume for contradiction there exists $v \in N(A \cup B)$ such that $F(A,B) \cup F(B,A) \nsubseteq N(v)$. We have two cases:

(1) $F(A,B) \nsubseteq N(v)$ and $F(B,A) \nsubseteq N(v)$. Suppose w.l.o.g. that $v \in N(A)$. There exists $b \in F(B,A)$ such that $b \notin N(v)$. Then, we deduce by Proposition 17 that $h(bv) \cap A \neq \emptyset$ and $v \in A/B$.

(2) $F(A,B) \subseteq N(v)$ and $F(B,A) \nsubseteq N(v)$ (w.l.o.g.). Since $F(A,B) \subseteq N(v)$, $v \in N(A)$ holds. Thus, $v \in A/B$ again follows from Proposition 17.

In both cases, we obtain $v \in A/B$ with $v \notin A$. This contradicts A being saturated. We derive $F(A,B) \cup F(B,A) \subseteq N(v)$. Therefore, every $v \in N(A) \setminus B$ also lies in $N(B) \setminus A$ so that $N(A) \cap N(B) = N(A \cup B)$ holds along with $N(A) = F(B,A) \cup N(A \cup B)$ and $F(A, \overline{A}) = F(A, N(A \cup B))$. To see that $F(A, \overline{A})$ is a clique, observe that $B \cup N(A \cup B)$ is connected since B is convex. We deduce that $B \cup N(A \cup B)$ is included in a connected component of $G - A$. Since $F(A, \overline{A}) = F(A, N(A \cup B))$ and A is convex as it saturated, we obtain from Theorem 8 that $F(A, \overline{A})$ is a clique. ◀

Proposition 17 and Lemma 18 have two consequences. First, we can characterize $\mathrm{MFS}(A,B)$ as the set of pairs uv the closure of which contains non-adjacent vertices of $N(A \cup B)$, or in other words, a forbidden pair within $N(A \cup B)$.

▶ **Lemma 19.** *Let G be a connected graph and let A, B be two linked, disjoint and saturated subsets of $V(G)$. The following equality holds:*

$$\mathrm{MFS}(A,B) = \{uv \subseteq \overline{A \cup B} : h(uv) \cap N(A \cup B) \text{ is not a clique}\}$$

In particular, $X \subseteq \overline{A \cup B}$ is forbidden if and only if it includes a forbidden pair of $\mathrm{MFS}(A,B)$.

As another consequence, we can describe $N(A \cup B)$ and its interactions with A and B depending on whether it is a clique or not.

▶ **Lemma 20.** *Let G be a connected graph and let A, B be two linked, disjoint and saturated subsets of $V(G)$. Then either $N(A \cup B)$ is a clique or for every $u, v \in N(A \cup B)$, $F(A,v) = F(A,u)$ and $F(B,v) = F(B,u)$.*

Proof. Suppose that $N(A \cup B)$ is not a clique and let u, v be two non-adjacent vertices of $N(A \cup B)$. We first prove that $F(A,v) = F(A,u)$ and $F(B,v) = F(B,u)$. Assume for contradiction that $F(A,v) \neq F(A,u)$. We have $F(A,v) \setminus F(A,u) \neq \emptyset$ or $F(A,u) \setminus F(A,v) \neq \emptyset$. By Proposition 17, we deduce $v \in h(A \cup u)$ or $u \in h(A \cup v)$. Since u, v are not adjacent, $uv \in \mathrm{MFS}(A,B)$ by Lemma 19 and we obtain $h(A \cup u) \cap B \neq \emptyset$ or $h(A \cup v) \cap B \neq \emptyset$. Thus, either $u \in A/B$ or $v \in A/B$. This contradicts A being saturated. We obtain $F(A,v) = F(A,u)$, and $F(B,v) = F(B,u)$ using the same argument on B.

Now, let $w \in N(A \cup B)$ such that $w \neq u, v$. If w is not adjacent to u or v, then $F(A,w) = F(A,u) = F(A,v)$ and $F(B,w) = F(B,u) = F(B,v)$ readily holds by previous argument. Therefore, suppose that w is adjacent to both u and v. We prove that: (1) $F(A,w) \setminus F(A,u) = \emptyset$ and (2) $F(A,u) \setminus F(A,w) = \emptyset$.

(1) Assume for contradiction that $F(A,w) \setminus F(A,u) \neq \emptyset$. Then, $w \in h(A \cup u)$ by Proposition 17. But since, $F(A,u) = F(A,v)$, we deduce $F(A,w) \setminus F(A,v) \neq \emptyset$ and hence $w \in h(A \cup v)$. Because $uv \in \mathrm{MFS}(A,B)$ and $w \in h(A \cup u) \cap h(A \cup v)$, $w \notin A$ is a contradiction with A being saturated. We deduce that $F(A,w) \setminus F(A,u) = \emptyset$ must hold.

(2) Again, suppose for contradiction that $F(A,u) \setminus F(A,w) \neq \emptyset$. By Proposition 5, we obtain $u \in h(A \cup w)$ and since $F(A,u) = F(A,v)$, $v \in h(A \cup w)$ also holds. Since $uv \in \mathrm{MFS}(A,B)$, we obtain $w \in B/A$, a contradiction with B being saturated.

We conclude that $F(A,w) = F(A,u)$ holds, and similarly $F(B,w) = F(B,u)$. This concludes the proof. ◀

The two situations obtained from Lemma 20 are illustrated in Figure 6. In the case where

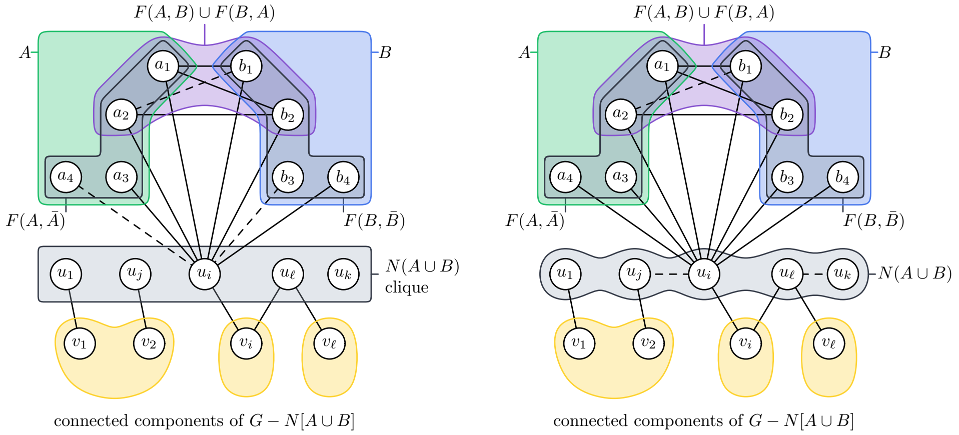


Figure 6 The two possible situations of Lemma 20. On the left, $N(A \cup B)$ is a clique. Each vertex of $N(A \cup B)$, is connected to each vertex of $F(A,B) \cup F(B,A)$ (circled in purple), modelled by u_i. However, it needs not be adjacent to all the vertices of the cliques $F(A,\bar{A})$ and $F(B,\bar{B})$ (the dotted line $u_i a_4$ indicates a non-edge). On the right, $N(A \cup B)$ is not a clique (for instance, u_i, u_j are not adjacent). Each vertex of $N(A \cup B)$ is complete to $F(A,\bar{A}) \cup F(B,\bar{B})$, including $F(A,B) \cup F(B,A)$.

$N(A \cup B)$ is not a clique, Lemma 20 together with Lemma 18 yields the subsequent corollary that will be useful later on.

▶ **Corollary 21.** *If $N(A \cup B)$ is not a clique, then for every clique $K \subseteq N(A \cup B)$, $F(A,\overline{A}) \cup K$ (resp. $F(B,\overline{B}) \cup K$) is a clique.*

Thanks to Lemmas 19 and 20, we are in position to relate the separability of A, B with (co)bipartiteness. We first address the case where all the vertices left to assign lie in $N(A \cup B)$, i.e., when $\overline{A \cup B} = N(A \cup B)$. Although restricted, this case gives some insights for the general one.

If $N(A \cup B)$ is a clique, then $\mathrm{MFS}(A,B) = \emptyset$ by Lemma 19. Hence every bipartition X, Y of $N(A \cup B)$ readily satisfies $h(A \cup X) \cap B = \emptyset$ and $h(B \cup Y) \cap A = \emptyset$. Therefore, X and Y need only satisfy $h(A \cup X) \cap Y = \emptyset$ and $h(B \cup X) \cap Y = \emptyset$. The trivial bipartition $X = \emptyset$ and $Y = N(A \cup B)$ vacuously obeys this requirement.

On the other hand, when $N(A \cup B)$ is not a clique, the subsequent lemma implies that for any bipartition X, Y of $N(A \cup B)$ into cliques, $A \cup X$ and $B \cup Y$ are convex.

▶ **Lemma 22.** *Let G be a connected graph and let A, B be two linked, disjoint and saturated subsets of $V(G)$ such that $N(A \cup B)$ is not a clique. Then for every clique $K \subseteq N(A \cup B)$, both $A \cup K$ and $B \cup K$ are convex.*

Proof. To check that $A \cup K$ is convex, we verify that $J[u,v] \subseteq A \cup K$ for every $u, v \in A \cup K$. If $u, v \in A$ or $u, v \in K$, then the result holds since A is convex and K is a clique. Consider instead $u \in A, v \in K$. Assume for contradiction $J[u,v] \nsubseteq K \cup A$. There exists a chordless uv-path $u = v_1, \ldots, v_k = v$ such that $v_i \notin K \cup A$ for some $1 < i < k$. Consider the least such i. By assumption $v_i \in N(A)$ and $v_{i-1} \in F(A, v_i)$. Morever, since A, B are saturated, $v_i \in N(A \cup B)$ must hold. As $N(A \cup B)$ is not a clique, we obtain by Lemma 20 that $F(A, v_i) = F(A, v)$, meaning that v_{i-1} is adjacent to v. This contradicts v_i being on a chordless uv-path. We deduce that $J[u,v] \subseteq A \cup K$ and $A \cup K$ is convex. ◀

We finally arrive at the following intermediate claim.

▶ **Lemma 23.** *Let G be a connected graph and let A, B be two disjoint, linked and saturated subsets of $V(G)$. If $\overline{A \cup B} = N(A \cup B)$, then A and B are separable if and only if $N(A \cup B)$ is cobipartite.*

Proof. We start with the only if part. Let $H = A \cup X$, $\overline{H} = B \cup Y$ be half-spaces separating A and B. By assumption, X contains no forbidden pair of $\mathrm{MFS}(A, B)$. Since $X \subseteq N(A \cup B)$, we deduce from Lemma 19 that X is a clique. In the same way, we deduce that Y is a clique. As X, Y is a bipartition of $\overline{A \cup B} = N(A \cup B)$, we deduce that $N(A \cup B)$ is cobipartite.

We proceed to the if part. If $N(A \cup B)$ is cobipartite, we have two cases: either $N(A \cup B)$ is a clique or it is not. If $N(A \cup B)$ is a clique, then (resp. $A \cup N(A \cup B)$ and B) are half-spaces separating A and B. If $N(A \cup B)$ is not a clique, the fact that $A \cup X$ and $B \cup Y$ are half-spaces for all bipartitions X, Y of $N(A \cup B)$ into cliques follows from Lemma 22. ◀

Let us consider now that there are vertices outside of $N(A \cup B)$, i.e., $N(A \cup B) \subset \overline{A \cup B}$. First, if $N(A \cup B)$ is a clique, $\mathrm{MFS}(A, B) = \emptyset$ still holds by Lemma 19. In this case, the same reasoning as before applies, and $A, B \cup \overline{A \cup B}$ is a half-space separation of A, B. Suppose on the other hand that $N(A \cup B)$ is not a clique. If it is not cobipartite, then any bipartition of $N(A \cup B)$ will contain a pair of non-adjacent vertices, and hence a forbidden pair, again due to Lemma 19. In other words, if $N(A \cup B)$ is not cobipartite, A and B are not separable. However, there are also cases where $N(A \cup B)$ is cobipartite, yet A and B are not separable. This is the case for the graphs of Figure 7, that we will use to illustrate the steps of the upcoming discussion. This happens because when picking an element v in a connected component S of $G - N[A \cup B]$, $h(A \cup v)$ and $h(B \cup v)$ will share elements from $N(S)$, regardless of the structure of $N(A \cup B)$ (clique or not).

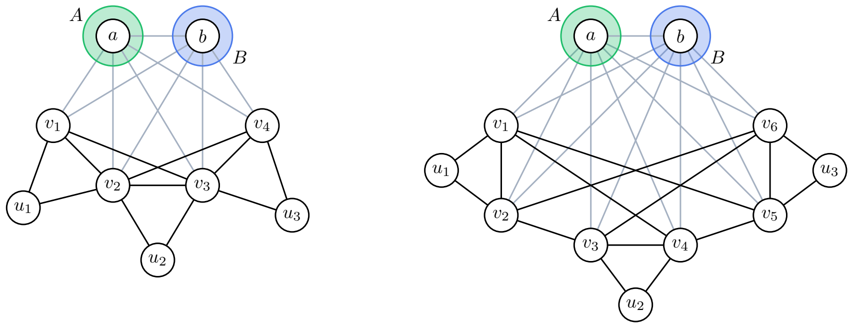


Figure 7 Two examples where A and B are linked and saturated, yet not separable despite $N(A \cup B)$ being cobipartite. For readability, the edges incident to a and b are clearer. Remark that since $N(A \cup B)$ is not a clique, both a and b are complete to $N(A \cup B)$ in virtue of Lemma 20.

▶ **Lemma 24.** *Let G be a connected graph and let A, B be two linked, disjoint and saturated subsets of $V(G)$. Let S be a connected component of $G - N[A \cup B]$. Then, for every $v \in S$, $N(S) \subseteq h(A \cup v) \cap h(B \cup v) \cap N(A \cup B)$.*

Using Lemma 24, we define an equivalence relation on $\overline{A \cup B}$ that will help us characterize the separability of A and B. Every half-space separation H, $\overline{H}$ of A and B, if any, can be written as $H = A \cup X$ and $\overline{H} = B \cup Y$ where X, Y is a bipartition of $\overline{A \cup B}$. Since $H \cap \bar{H} = \emptyset$, we have $H \cap Y = h(A \cup X) \cap Y = \emptyset$ and similarly $\overline{H} \cap X = h(B \cup Y) \cap X = \emptyset$. As a direct application of Lemma 24, we deduce:

(1) For each connected component S of $G - N[A \cup B]$, either $N[S] \subseteq X$ or $N[S] \subseteq Y$;

(2) If $S_1, \dots, S_k$ is a sequence of (not necessarily distinct) connected components of $G - N[A \cup B]$ such that $N(S_i) \cap N(S_{i+1}) \neq \emptyset$ for each $1 \leq i < k$, then $\bigcup_{i=1}^{k} N[S_i]$ must be included in one of X or Y. We call such a sequence an intersecting sequence of connected components.

Given an intersecting sequence $S_1, \dots, S_k$ of connected components of $G - N[A \cup B]$, we say for brevity that u, v belongs to the sequence $S_1, \dots, S_k$ if there exists $1 \leq i, j \leq k$ such that $u \in N[S_i]$ and $v \in N[S_j]$. Let us define the equivalence relation $\equiv_{AB}$ on $\overline{A \cup B}$ such that, for all $u, v \in \overline{A \cup B}$:

$$u \equiv_{AB} v \iff u = v \text{ or } u, v \text{ belong to an intersecting sequence of}$$
$$\text{connected components of } G - N[A \cup B]$$

▶ **Proposition 25.** *The relation $\equiv_{AB}$ is an equivalence relation.*

▶ Remark 26. The definition of $\equiv_{AB}$ encompasses the vertices that do not belong to the closed neighborhood of any connected component of $G - N[A \cup B]$, i.e., those vertices v in $N(A \cup B)$ such that $N[v] \subseteq N[A \cup B]$. By definition of $\equiv_{AB}$, they are equivalent to themselves only.

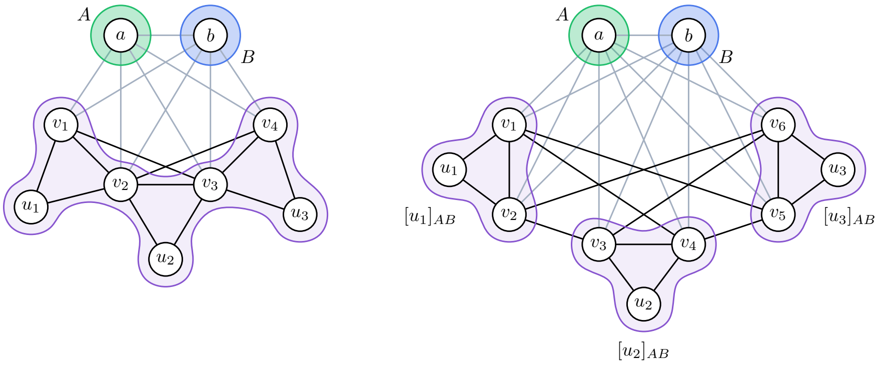


Figure 8 The equivalence relation $\equiv_{AB}$ applied to the graphs of Figure 7. The classes are circled (purple). On the left, there is a unique equivalence class. Remark that, as a consequence, v_1v_4 is a forbidden pair all the while $v_1 \equiv_{AB} v_4$. On the right, there are three classes, $[u_1]_{AB}$, $[u_2]_{AB}$, and $[u_3]_{AB}$.

For $u \in \overline{A \cup B}$, let $[u]_{AB}$ be the equivalence class of u: $[u]_{AB} = \{v \in \overline{A \cup B} : u \equiv_{AB} v\}$. In Figure 8, we give the equivalence classes induced by $\equiv_{AB}$ on the graphs of Figure 7. The next lemma is a direct yet important consequence of the above discussion and Lemma 24.

▶ **Lemma 27.** *Let G be a connected graph and let A, B be two disjoint, linked and saturated subsets of $V(G)$. For every bipartition X, Y of $\overline{A \cup B}$, we have $h(A \cup X) \cap Y = \emptyset$ and $h(B \cup Y) \cap X = \emptyset$ only if for each $v \in \overline{A \cup B}$, either $[v]_{AB} \subseteq X$ or $[v]_{AB} \subseteq Y$.*

We consider $\equiv_{AB}$ together with $\mathrm{MFS}(A, B)$. Remind that $\mathrm{MFS}(A, B)$ consists in pairs of vertices only, thanks to Lemma 19. Hence, a forbidden pair $uv \in \mathrm{MFS}(A, B)$ falls into exactly one of the following cases regarding equivalence classes:

(1) Either $u \equiv_{AB} v$ so that the equivalence class $[u]_{AB}$ prevents separation of A and B on its own (see Proposition 28 below). This case happens for instance in the graph on the left of Figure 8: $v_1 \equiv_{AB} v_4$ yet $v_1v_4 \in \mathrm{MFS}(A, B)$.

(2) Or $u \not\equiv_{AB} v$, so that $[u]_{AB}$ and $[v]_{AB}$ cannot be taken together in any separation of A and B. For example in the graph on the right of Figure 8 we have $v_1 \not\equiv_{AB} v_3$ and $v_1v_3 \in \mathrm{MFS}(A, B)$, which makes $[u_1]_{AB}$ and $[u_2]_{AB}$ incompatible for separating A and B. In this example, all equivalence classes are incompatible, so that A and B not separable.

As for the first case, we have the direct property:

▶ **Proposition 28.** *If $uv \in \mathrm{MFS}(A, B)$ and $u \equiv_{AB} v$, then A, B are not separable.*

For the second case, we can build a graph G_{AB} on the equivalence classes of $\equiv_{AB}$ that makes adjacent every two distinct equivalence classes sharing a forbidden pair. More formally:

$$V(G_{AB}) = \{[v]_{AB} : v \in \overline{A \cup B}\}$$
$$E(G_{AB}) = \{[u]_{AB}[v]_{AB} : u \not\equiv_{AB} v \text{ and } uv \in \mathrm{MFS}(A, B)\}.$$

For the graph on the right of Figure 8, the corresponding graph G_{AB} will be a clique. Figure 9 illustrates the graph G_{AB} on an other example.

▶ Remark 29. In the case where $\overline{A \cup B} = N(A \cup B)$, the equivalence classes $[v]_{AB}$ are precisely the singletons $\{v\}$ for all $v \in N(A \cup B)$. Identifying $[v]_{AB}$ with v, G_{AB} turns out to be precisely the complement of $G[N(A \cup B)]$.

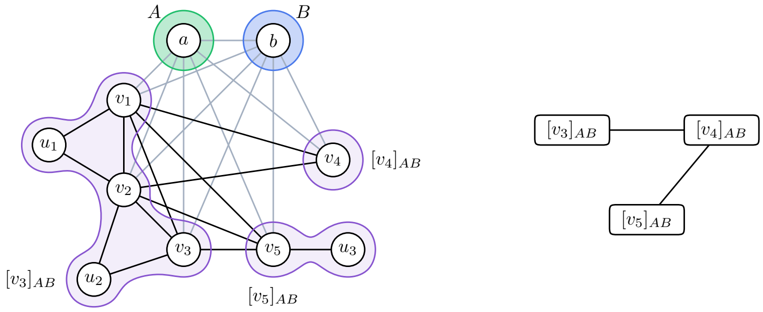


Figure 9 On the left, a graph with linked A and B where the equivalence class are highlighted. Again, the edges incident to a and b are clearer for readability. On the right, the corresponding graph G_{AB}.

Before characterizing the separability of A and B we give a lemma extending Lemma 22.

▶ **Lemma 30.** *Let G be a connected graph and let A, B be two disjoint, linked and saturated subsets of $V(G)$ such that $N(A \cup B)$ is not a clique. Then, for every collection $\mathcal{X}$ of equivalence classes of $\equiv_{AB}$, if $\bigcup \mathcal{X} \cap N(A \cup B)$ is a clique, then both $A \cup \bigcup \mathcal{X}$ and $B \cup \bigcup \mathcal{X}$ are convex.*

Proof. Let $\mathcal{X}$ be a collection of equivalences classes such that $\bigcup \mathcal{X} \cap N(A \cup B)$ is a clique and let $C = A \cup \bigcup \mathcal{X}$. We show that C is convex. We put $K = F(A, \bar{A}) \cup (N(A \cup B) \cap C)$. Now, by assumption, $N(A \cup B) \cap C$ is a clique, A and B are linked and saturated and $N(A \cup B)$ is not a clique. Therefore, K is a clique by Corollary 21. In view of Lemma 9, we show that K is a clique separator of G and that $C \setminus K$ is a union of connected components of $G - K$. First, since $F(A, \bar{A}) \subseteq K$ and K is a clique, we have that $G - K$ disconnects $A \setminus K$ from $\overline{A \cup B} \setminus K$. Hence K is a clique separator of G and moreover, $A \setminus K$ is indeed a union of connected components of $G - K$ since $F(A, \bar{A}) \subseteq K$. Now we consider $C \setminus (K \cup A)$. If $C \setminus (K \cup A) = \emptyset$, we deduce $C \subseteq A \cup K$ and the result holds by Lemma 22. Assume that $C \setminus (K \cup A) \neq \emptyset$ and let $S_1, \ldots, S_k$ be the connected components of $G - N[A \cup B]$ such that $C \cap S_i \neq \emptyset$ for each $1 \leq i \leq k$. We have $C \setminus (A \cup K) \subseteq \bigcup_{i=1}^{k} S_i$. We show that $\bigcup_{i=1}^{k} S_i \subseteq C \setminus (A \cup K)$. Let $v \in S_i$ for some $1 \leq i \leq k$. By definition of $\equiv_{AB}$, $S \subseteq [v]_{AB}$ and since $\mathcal{X}$ is a collection of equivalence classes, we obtain $S \subseteq [v]_{AB} \subseteq C \setminus (K \cup A)$. We deduce $C \setminus (A \cup K) \subseteq \bigcup_{i=1}^{k} S_i$ and hence $C \setminus (A \cup K) = \bigcup_{i=1}^{k} S_i$. It remains to show that S_i is a connected component of $G - K$. Since S_i is a connected component of $G - N[A \cup B]$, it is a connected component of

$G - N(S_i)$. Moreover, $N(S_i) \subseteq N(A \cup B)$ by construction. Finally, again by definition of C and $\equiv_{AB}$, $N(S_i) \subseteq C$. Henceforth, $N(S_i) \subseteq N(A \cup B) \cap C$ from which we deduce that S_i is a connected component of $G - K$. Hence, $C = K \cup (A \setminus K) \cup (C \setminus (K \cup A))$ is the union of a clique separator K of G and connected components of $G - K$. Applying Lemma 9, we deduce that C is convex, which concludes the proof. ◀

We can characterize the separability of A, B by generalizing Lemma 23.

▶ **Theorem 31.** *Let G be a connected graph and let A, B be two disjoint, linked and saturated subsets of V. Then A and B are separable if and only if the next conditions hold:*

(1) *for every $v \in \overline{A \cup B}$, $[v]_{AB}$ contains no forbidden pairs;*

(2) *G_{AB} is bipartite.*

Proof. We start with the only if part. Assume A and B are separable and let $H, \bar{H}$ be a half-space separation of A and B with $A \subseteq H$ and $B \subseteq \bar{H}$. Put $X = H \setminus A$ and $Y = \overline{H} \setminus B$. By assumption, $H \cap Y = h(A \cup X) \cap Y = \emptyset$. Hence, by Lemma 27, for each $v \in \overline{A \cup B}$, either $[v]_{AB} \subseteq X$ or $[v]_{AB} \subseteq Y$. Let $\mathcal{X} = \{[v]_{AB} \in V(G_{AB}) : [v]_{AB} \subseteq X\}$ and $\mathcal{Y} = \{[v]_{AB} \in V(G_{AB}) : [v]_{AB} \subseteq Y\}$. Since $H, \overline{H}$ are half-spaces separating A and B, and $X \subseteq H$, $Y \subseteq \overline{H}$, we deduce that neither X nor Y contain a forbidden pair of $\mathrm{MFS}(A, B)$. We derive:

(1) for each $[v]_{AB}$, $[v]_{AB}$ contains no forbidden pair, i.e., item (1) holds;

(2) for each pair of distinct classes $[u]_{AB}, [v]_{AB}$ in X (resp. Y), $[u]_{AB}$ and $[v]_{AB}$ are not adjacent in G_{AB}, i.e., that $\mathcal{X}$ (resp. $\mathcal{Y}$) is an independent set of G_{AB}. Since $\mathcal{X}$, $\mathcal{Y}$ is a partition of G_{AB} into two independent sets, we conclude that G_{AB} is bipartite, and that item (2) of the theorem holds.

We move to the if part. Assume both items (1) and item (2) are satisfied. In particular, if $N(A \cup B)$ is a clique, $\mathrm{MFS}(A, B) = \emptyset$ by Lemma 19. Hence, $A \cup N(A \cup B)$ and B (resp. $B \cup N(A \cup B)$ and A) are half-spaces separating A and B. Assume $N(A \cup B)$ is not a clique and let $\mathcal{X}, \mathcal{Y}$ be any bipartition of $V(G_{AB})$ into two independent sets. We show that $\bigcup \mathcal{X}$ contains no forbidden pair. Assume for contradiction there exists a forbidden pair $uv \in \bigcup \mathcal{X}$. We have two cases:

(1) $u \equiv_{AB} v$, but this would contradict item (1) of the statement;

(2) $u \not\equiv_{AB} v$, but this would contradict $\mathcal{X}_A$ being an independent set of G_{AB} by definition of G_{AB}.

By Lemma 19 we deduce that $\bigcup \mathcal{X}$ contains no forbidden pair, and hence that $\bigcup \mathcal{X} \cap N(A \cup B)$ is a clique. Applying Lemma 30, $A \cup \bigcup \mathcal{X}$ is convex. The same reasoning on $B \cup \mathcal{Y}$ yields that $A \cup \bigcup \mathcal{X}$ and $B \cup \mathcal{Y}$ are half-spaces separating A and B. This concludes the proof. ◀

Figure 10 illustrate the conditions of Theorem 31 on the example of Figure 9. We finally argue that the conditions of Theorem 31 can be checked in polynomial time. Since $\mathrm{MFS}(A, B)$ consists in pairs only, it can be computed in polynomial time. Then, we identify the connected components of $G - N[A \cup B]$ in polynomial time by traversing $G - N[A \cup B]$. We then identify the equivalence relation $\equiv_{AB}$ and build G_{AB} accordingly. Testing that no equivalent class contains a forbidden pair can be done in polynomial as well as checking that G_{AB} is bipartite. We deduce:

▶ **Theorem 32.** *Let G be a connected graph and let A, B be two disjoint, linked and saturated subsets of V. Whether A, B can be separated by half-spaces can be checked in polynomial time in the size of G.*

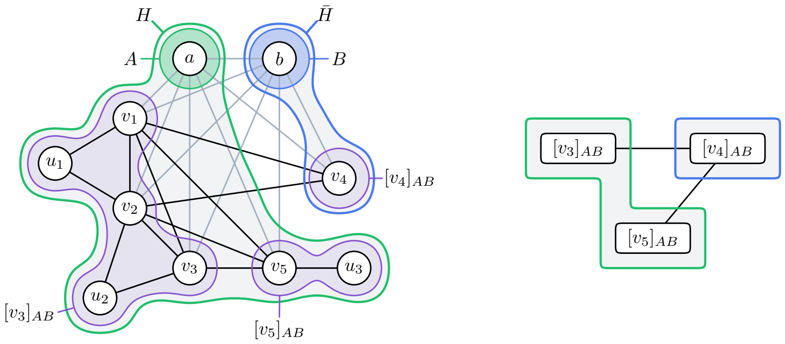


Figure 10 Illustration of Theorem 31 on the graph of Figure 9. A half-space separation of A and B is drawn. Observe that it corresponds to a bipartition of G_{AB} into independent sets.

4 Conclusion

We proved that half-space separability can be tested in polynomial time for monophonic convexity. Using Lemma 30, the algorithm we propose can be adapted to generate a pair of half-spaces separating two sets of vertices, if any. Moreover, we deduce as a corollary that the 2-partition problem can be solved in polynomial time for monophonic convexity, thus answering an open problem in [17].

To decide separability, we used the underlying graph together with the fact that the Carathéodory number is constant for monophonic convexity (Theorem 6, [12]). A natural question is then to investigate to what extent the Carathéodory number can be used to decide separability. However, relying on the problem of 2-coloring 3-uniform hypergraphs [19], we can show that already with Carathéodory number 3, half-space separation in general convexity spaces is out of reach.

▶ **Theorem 33.** *Half-space separation is NP-complete for convexity spaces $(V, \mathcal{C})$ given by V and a hull operator h that computes $h(X)$ in polynomial time in the size of V for all $X \subseteq V$, even if $(V, \mathcal{C})$ has Carathéodory number 3.*

Theorem 33 together with Theorem 1 motivate the next intriguing open problem.

▶ **Open Problem 34.** *Find a natural (graph) convexity with Carathéodory number 2 (e.g. triangle-path convexity [5]) where half-space separation is hard, or show that for all such convexities, half-space separation is tractable.*

References

1 Liliana Alcón, Boštjan Brešar, Tanja Gologranc, Marisa Gutierrez, Tadeja Kraner Šumenjak, Iztok Peterin, and Aleksandra Tepeh. Toll convexity. *European Journal of Combinatorics*, 46:161–175, 2015. doi:10.1016/j.ejc.2015.01.002.

2 Hans-Jürgen Bandelt. Graphs with intrinsic S_3 convexities. *Journal of graph theory*, 13(2):215–228, 1989. doi:10.1002/jgt.3190130208.

3 Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992. doi:10.1145/130385.130401.

4 Marco Bressan, Emmanuel Esposito, and Maximilian Thiessen. Efficient algorithms for learning monophonic halfspaces in graphs. *arXiv preprint arXiv:2405.00853*, 2024. doi:10.48550/arXiv.2405.00853.

5 Manoj Changat and Joseph Mathew. On triangle path convexity in graphs. *Discrete Mathematics*, 206(1-3):91–95, 1999. doi:10.1016/S0012-365X(98)00394-X.

6 Victor Chepoi. Some properties of domain finite convexity structures (in russian), res. *Algebra, Geometry and Appl.(Moldova State University)*, pages 142–148, 1986.

7 Victor Chepoi. Separation of two convex sets in convexity structures. *Journal of Geometry*, 50:30–51, 1994. doi:10.1007/BF01222661.

8 Victor Chepoi. Separation axiom S_3 for geodesic convexity in graphs. *arXiv preprint arXiv:2405.07512*, 2024. doi:10.48550/arXiv.2405.07512.

9 Mitre C. Dourado, Marisa Gutierrez, Fábio Protti, and Silvia Tondato. Weakly toll convexity and proper interval graphs. *Discrete Mathematics & Theoretical Computer Science*, vol. 26:2, April 2024. doi:10.46298/dmtcs.9837.

10 Mitre C Dourado, Fábio Protti, and Jayme L Szwarcfiter. Complexity results related to monophonic convexity. *Discrete Applied Mathematics*, 158(12):1268–1274, 2010. doi:10.1016/j.dam.2009.11.016.

11 Feodor F Dragan, Falk Nicolai, and Andreas Brandstädt. Convexity and hhd-free graphs. *SIAM Journal on Discrete Mathematics*, 12(1):119–135, 1999. doi:10.1137/S0895480195321718.

12 Pierre Duchet. Convex sets in graphs, ii. minimal path convexity. *Journal of Combinatorial Theory, Series B*, 44(3):307–316, 1988. doi:10.1016/0095-8956(88)90039-1.

13 Mohammed Elaroussi, Lhouari Nourine, and Simon Vilmin. Half-space separation in monophonic convexity. *arXiv preprint arXiv:2404.17564*, 2024. doi:10.48550/arXiv.2404.17564.

14 JW Ellis. A general set-separation theorem. *Duke Math. J.*, 19(1):417–421, 1952. doi:10.1215/S0012-7094-52-01941-8.

15 Martin Farber and Robert E Jamison. Convexity in graphs and hypergraphs. *SIAM Journal on Algebraic Discrete Methods*, 7(3):433–444, 1986. doi:10.1137/0607049.

16 Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 209–217, 1998. doi:10.1145/279943.279985.

17 Lucía M González, Luciano N Grippo, Martín D Safe, and Vinicius F dos Santos. Covering graphs with convex sets and partitioning graphs into convex sets. *Information Processing Letters*, 158:105944, 2020. doi:10.1016/j.ipl.2020.105944.

18 David Kay and Eugene W Womble. Axiomatic convexity theory and relationships between the carathéodory, helly, and radon numbers. *Pacific Journal of Mathematics*, 38(2):471–485, 1971. doi:10.2140/pjm.1971.38.471.

19 László Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeastern Conference of Combinatorics, Graph Theory, and Computing*, pages 3–12. Utilitas Mathematica Publishing, 1973.

20 Ignacio M Pelayo. *Geodesic convexity in graphs.* Springer, 2013. doi:10.1007/978-1-4614-8699-2.

21 Florian Seiffarth, Tamás Horváth, and Stefan Wrobel. Maximal closed set and half-space separations in finite closure systems. *Theoretical Computer Science*, 973:114105, 2023. doi:10.1016/j.tcs.2023.114105.

22 Maximilian Thiessen and Thomas Gärtner. Active learning of convex half-spaces on graphs. *Advances in Neural Information Processing Systems*, 34:23413–23425, 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/c4bf1e24f3e6f92ca9dfd9a7a1a1049c-Paper.pdf.

23 Marcel van de Vel. Binary convexities and distributive lattices. *Proceedings of the London Mathematical Society*, 3(1):1–33, 1984. doi:10.1112/plms/s3-48.1.1.

24 Marcel van De Vel. *Theory of convex structures.* Elsevier, 1993.

25 Vladimir Vapnik. *Statistical learning theory.* Wiley New York, 1998.

Structural Parameters for Dense Temporal Graphs

Jessica Enright
School of Computing Science, University of Glasgow, UK

Samuel D. Hand
School of Computing Science, University of Glasgow, UK

Laura Larios-Jones
School of Computing Science, University of Glasgow, UK

Kitty Meeks
School of Computing Science, University of Glasgow, UK

Abstract

Temporal graphs provide a useful model for many real-world networks. Unfortunately, the majority of algorithmic problems we might consider on such graphs are intractable. There has been recent progress in defining structural parameters which describe tractable cases by simultaneously restricting the underlying structure and the times at which edges appear in the graph. These all rely on the temporal graph being sparse in some sense. We introduce temporal analogues of three increasingly restrictive static graph parameters – cliquewidth, modular-width and neighbourhood diversity – which take small values for highly structured temporal graphs, even if a large number of edges are active at each timestep. The computational problems solvable efficiently when the temporal cliquewidth of the input graph is bounded form a subset of those solvable efficiently when the temporal modular-width is bounded, which is in turn a subset of problems efficiently solvable when the temporal neighbourhood diversity is bounded. By considering specific temporal graph problems, we demonstrate that (up to standard complexity theoretic assumptions) these inclusions are strict.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability

Keywords and phrases Graph algorithms, Parameterized Algorithms, Temporal Graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.52

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2404.19453`

Funding *Jessica Enright*: Supported by EPSRC grant EP/T004878/1.
Samuel D. Hand: Supported by an EPSRC doctoral training account.
Kitty Meeks: Supported by EPSRC grant EP/T004878/1.

1 Introduction

Temporal graphs, in which the set of active edges changes over time, are a useful formalism for modelling numerous real-world phenomena, from social networks in which friendships evolve over time to transport networks in which a particular connection is only available on particular days and times. This has inspired a large volume of research into the algorithmic aspects of such graphs in recent years [8, 26, 34], but unfortunately in many cases even problems which admit polynomial-time algorithms on static graphs become intractable in the temporal setting.

This has motivated the study of computational problems on restricted classes of temporal graphs, with mixed success: in a few cases, restricting the structure of the underlying static graph (e.g. to be a path or a tree) is effective, but numerous natural temporal problems remain intractable even when the underlying graph is very strongly restricted (e.g. when it is required to be a path [33] or a star [2]). Recently, several promising new parameters have been introduced that simultaneously restrict properties of the static underlying graph and the times at which edges are active in the graph; these include several analogues of treewidth for temporal graphs [18, 30], the temporal feedback edge/connection number [23], the timed vertex feedback number [7] and the (vertex-)interval-membership-width of the temporal graph [6]. However, all of these new temporal parameters are only small for temporal graphs that are, in some sense, sparse: none of them can be bounded on a temporal graph which has a superlinear (in the number of vertices) number of active edges at every timestep.

In this paper, we attempt to fill this gap in the toolbox of parameters for temporal graphs by introducing three new parameters which can take small values on temporal graphs which are dense but are sufficiently highly structured. Specifically, we define natural temporal analogues of cliquewidth, modular-width and neighbourhood diversity, all of which have proved highly effective in the design of efficient algorithms for static graphs.

Importantly, the neighbourhood diversity of a static graph upper bounds its modular-width, which upper bounds its cliquewidth. Both cliquewidth (introduced by Courcelle et al. [13]) and modular-width (introduced by Gajarský et al. [19], using the long-standing notion of modular decompositions [20]) can be defined in terms of width measures over composition trees allowing particular operations. Cliquewidth constructions have greater flexibility due to the fact that we are allowed to use an additional "relabelling" operation; this makes it possible, for example, to build long induced paths, which cannot exist in graphs of small modular width. Courcelle et al. [11] show that any graph property expressible in monadic second order is solvable in linear time on graphs of bounded cliquewidth. Gajarský et al. [19] provide examples of problems (Hamilton Path and Colouring) that are hard with respect to cliquewidth but tractable with respect to the more restrictive parameter modular-width. Neighbourhood diversity is a highly restrictive parameter, introduced by Lampis [28], which requires that large sets of vertices have identical neighbourhoods. Cordasco [9] demonstrated that Equitable Colouring is hard with respect to modular-width but tractable with respect to neighbourhood diversity.

These three static parameters are the inspiration for our temporal parameters. Informally, our new parameters are defined as follows:

- A temporal graph has *temporal neighbourhood diversity (TND)* at most k if its vertices can be partitioned into at most k classes such that each class induces either an independent set or a clique in which all edges are active at exactly the same times, and any two vertices in the same class have exactly the same neighbours outside the class at each timestep.
- *Temporal modular-width (TMW)* is a generalisation of TND: a temporal graph has TMW at most k if its vertices can be partitioned into modules such that two vertices in the same module must have the exactly the same neighbours outside the class at each timestep, but now each module need only be a temporal graph which itself has TMW at most k, rather than a clique or independent set.
- Like the static version, *temporal cliquewidth (TCW)* is defined to be the minimum number of labels needed to construct a temporal graph using four operations (create a vertex with a new label; take a disjoint union of two graphs; add all edges between vertices of two specified labels; relabel all occurrences of one label to another); the difference from the static case is that when adding edges between two sets of vertices these edges must all be active at exactly the same times.

We note that in every case we will recover the corresponding static parameter if all edges are active at the same times. It is immediate that the TND of a temporal graph is an upper bound on its TMW, and it is straightforward to show (see Section 3) that the TMW is an upper bound on the TCW. Thus, the most general algorithmic result we hope to obtain is to show that a problem is tractable when the TCW of the input temporal graph is bounded, but we expect to be able to show tractability for more problems as we impose stronger restictions on the input by bounding respectively the TMW and the TND.

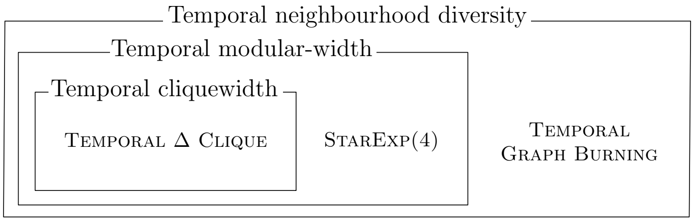


Figure 1 A diagram of our parameters and the problems we show to be tractable with respect to each. A problem is in a rectangle if it is tractable with respect to the parameter it is labelled with. Assuming P≠NP, each problem is in the rectangle for the most general of the three parameters for which it is tractable.

To illustrate the value of considering this hierarchy of parameters, we provide examples of problems which can be solved efficiently when each of our three new parameters is bounded, but which (in the case of TND and TMW) remain intractable when we restrict only the next most restrictive parameter, as illustrated in Figure 1. Specifically, we prove that:

- Temporal Clique is solvable in linear time when a temporal cliquewidth decomposition of constant width is given (see Section 2).
- StarExp(4) (the problem of deciding whether there is a closed temporal walk visiting all vertices in a star when each edge is active at no more than four times) remains NP-hard on graphs with TCW at most three, but is solvable in polynomial time when the TMW of the input graph is bounded by a constant; in fact we provide an fpt-algorithm[1] with respect to TMW (see Section 3).
- Graph Burning is NP-hard on temporal graphs with constant TMW, but is solvable in polynomial time when the TND of the input graph is bounded by a constant; again this is an fpt-algorithm with respect to TND (see Section 4).

We also (in Section 4) provide an fpt-algorithm to solve SingMinReachDelete parameterised by TND (when the number of appearances of each edge is bounded), in order to illustrate additional techniques that may be used when working with this new temporal parameter. This problem asks, for a given source vertex, what the cardinality of the smallest set of time-edges is such that their deletion leaves at most r vertices temporally reachable from the source. We conjecture that SingMinReachDelete is another example of a problem that is tractable with respect to TND but intractable when only the TMW is restricted.

The remainder of the paper is organised as follows. We conclude this section by introducing some key notation and definitions used throughout the paper. The following three sections are devoted to TCW, TMW and TND respectively, with each section containing the formal definition of a parameter as well as results about problems which can be solved efficiently when that parameter is bounded. Many proof details are omitted due to space constraints. A full version of the paper can be found on arXiv. Statements with proofs omitted are highlighted with a ($\star$).

[1] We use fpt (lowercase) as a descriptor of algorithms witnessing the inclusion of a problem in the parameterised complexity class FPT.

1.1 Notation and definitions

We use a number of standard notations for temporal graphs and related notions. A *temporal graph* $\mathcal{G} = (G, \lambda)$ consists of an underlying static graph $\mathcal{G}_{\downarrow} = G$, and a time-labeling function $\lambda : E \to 2^{\mathbb{N}}$, assigning to each edge a set of timesteps at which it is active. We refer to a pair (e, t) consisting of an edge $e \in E(G)$ and time $t \in \lambda(e)$ as a *time-edge*. The set of all time-edges of a temporal graph is denoted by $\varepsilon(\mathcal{G})$ and the *lifetime* Λ of a temporal graph refers to the final time at which any edge is active, i.e. $\Lambda = \max\{\max \lambda(e) : e \in E(G)\}$. The *snapshot* $\mathcal{G}_t$ of a temporal graph $\mathcal{G}$ at time t is the static graph $G = (V, E_t)$ where E_t is the set of edges active at time t.

A *temporal path* on the temporal graph $\mathcal{G} = (G, \lambda)$ is a sequence of edge, time pairs $(e_1, t_1), ..., (e_\ell, t_\ell)$, such that $e_1, ..., e_\ell$ is a path on G, and $t_i \in \lambda(e_i)$ for every $i \in [\ell]$, and $t_1, ..., t_\ell$ is a strictly increasing sequence of times. Given a temporal path $(e_1, t_1), ..., (e_\ell, t_\ell)$ we refer to the time t_1 as its *departure time*, and t_ℓ as its *arrival time*.

We refer the reader to [15] for standard definitions from the field of parameterised complexity.

2 Tractability with respect to Temporal Cliquewidth

In this section we give the formal definition of the first of our new parameters, temporal cliquewidth, and demonstrate that the problem of finding a temporal clique admits an fpt-algorithm parameterised by temporal cliquewidth. Before defining temporal cliquewidth, we start by recalling the definition of cliquewidth in the static setting, as introduced by Courcelle and Olariu [14].

▶ **Definition 1** (Cliquewidth). *The* cliquewidth *of a static graph $G = (V, E)$ is the number of labels required to construct G using only the following operations:*

1. *Creating a new vertex with label i.*
2. *Taking the disjoint union of two labeled graphs.*
3. *Adding edges to join all vertices labeled i to all vertices labeled j, where $i \neq j$.*
4. *Renaming label i to label j.*

We refer to an algorithm which constructs a graph G using the above operations as a cliquewidth construction *of G.*

Computing the cliquewidth of a graph is NP-hard [17], although there exists a polynomial-time algorithm to recognise graphs of cliquewidth at most three [10].

Translating this definition into the temporal setting, and preserving the idea that vertices with the same label should be indistinguishable when we add edges – which imposes additional restrictions on the times at which new edges are active – we obtain our definition of temporal cliquewidth.

▶ **Definition 2** (Temporal Cliquewidth). *The* temporal cliquewidth *of a temporal graph $\mathcal{G} = (G, \lambda)$ is the number of labels required to construct $\mathcal{G}$ using only the following operations:*

1. *Creating a new vertex with label i.*
2. *Taking the disjoint union of two labeled graphs.*
3. *Adding edges to join all vertices labeled i to all vertices labeled j, where $i \neq j$, such that all the added edges are active at the same set of times T.*
4. *Renaming label i to label j.*

We refer to an algorithm which constructs a temporal graph $\mathcal{G}$ using the above operations as a temporal cliquewidth construction *of $\mathcal{G}$.*

We note that, if temporal graph $\mathcal{G}$ has bounded temporal cliquewidth k, then the underlying graph $G_{\downarrow}$ of $\mathcal{G}$ has cliquewidth at most k. The construction of the underlying graph is found by adding a static edge (if one does not already exist) whenever a time-edge is added in the construction of the temporal graph. In addition, the snapshot of $\mathcal{G}$ at any time t, G_t, also has cliquewidth at most k. This follows from a similar argument. If we have a temporal graph where the edges all appear at the same times, the cliquewidth of the underlying graph is the same as the temporal cliquewidth and the cliquewidth of any snapshots where edges are active. It follows immediately that it is NP-hard to compute temporal cliquewidth, as the NP-hard problem of computing the cliquewidth of a static graph is a special case.

The remainder of this section is devoted to proving that the problem Temporal Δ Clique is in FPT parameterised by the temporal cliquewidth of the input graph. This problem was introduced by Viard et al. [37] and asks, in any interval of times of size Δ, whether there is a set of at least h vertices such that there is an appearance of an edge between every pair of vertices in every sub-interval of Δ times. Hermelin et al. [25] investigate the variant of this problem where the interval in question is the entire lifetime of the temporal graph. More formally, it asks if there exists a set V' of cardinality at least h such that for each pair of distinct vertices $u, v \in V'$ and each time $0 \leq i \leq \Lambda - \Delta$ where Λ is the lifetime of $\mathcal{G}$, there exists a time-edge at time $t' \in [i, \Delta + i]$.

Hermelin et al. note that this is the case if and only if there is a set of vertices of size at least k such that they form a clique on the static graph consisting of edges that appear in every interval of Δ timesteps. They name this static graph a Δ-*association graph.* It is more formally defined as $G = \left(V, \bigcap_{i=1}^{\Lambda(\mathcal{G})-\Delta+1} \bigcup_{j=i}^{i+\Delta-1} E_j\right)$ where E_j is the set of edges active at time j. This reformulates the problem as follows.

Temporal Δ Clique

Input: A temporal graph $\mathcal{G} = (V, E, \lambda)$ and two integers Δ and h where $\Delta \leq T(\mathcal{G})$.
Output: Is there a set $V' \subseteq V$ of vertices such that $|V'| \geq h$ and V' is a clique in the Δ-association graph G of $\mathcal{G}$?

Note that a temporal graph $\mathcal{G}$ with integers Δ and h is a yes-instance of Temporal Δ Clique if and only if its Δ-association graph G and h are a yes-instance of Clique.

▶ **Proposition 1** (⋆). *If a temporal graph $\mathcal{G}$ has temporal cliquewidth k, then the Δ-association graph G of $\mathcal{G}$ has cliquewidth at most k.*

Gurski [22, Theorem 4.4] shows that Clique is solvable in linear time in graphs of bounded cliquewidth if its construction is given. This gives us the following result.

▶ **Theorem 2.** *Given a cliquewidth construction of the Δ-association graph of a temporal graph $\mathcal{G}$, Temporal Δ Clique can be solved in linear time.*

3 Tractability with respect to Temporal Modular-width

We now introduce a more restrictive parameter, temporal modular-width, and show (in Section 3.1) that there exist problems which are efficiently solvable when this parameter is bounded even though they remain intractable on temporal graphs with constant temporal cliquewidth.

We begin with the formal definition of the parameter. Again, we start by recalling the definition of the corresponding static parameter on which our definition is based.

▶ **Definition 3** (Modular-width, Section 2.5 [19]). *Suppose a static graph G can be constructed by the algebraic expression A which uses the following operations:*

1. *Creating an isolated vertex.*
2. *Taking the disjoint union of two graphs.*
3. *Taking the complete join of two graphs. That is, for graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, $V(G_1 \otimes G_2) = V(G_1) \cup V(G_2)$ and $E(G_1 \otimes G_2) = E(G_1) \cup E(G_2) \cup \{(v, w) : v \in V(G_1)$ and $w \in V(G_2)\}$.*
4. *The substitution of graphs $G_1, \ldots, G_n$ into a graph G' with vertices $v_1, \ldots, v_n$. This gives the graph $G'(G_1, \ldots, G_n)$ with vertex set $\bigcup_{1 \leq i \leq n} V(G_i)$ and edge set $\bigcup_{1 \leq i \leq n} E(G_i) \cup \{(v, w) : v \in V(G_i), w \in V(G_j),$ and $(v_i, v_j) \in E(G')\}$.*

The width of an expression A is the maximum number of operands in an occurrence of the operation 4 in A. The modular-width *of G, written $MW(G)$, is this minimum width of an expression A which constructs G.*

We refer to the graphs $G_1, \ldots, G_n$ which we substitute into G' as *modules*. It is known that for any graph G an algebraic expression of modular-width $MW(G)$ can be found in linear time [31, 36]. Observe that operations 2 and 3 are special cases of operation 4.

We now define our temporal analogue of this parameter. For simplicity we do not explicitly include the disjoint union and complete join operations, noting that once again these are special cases of the substitution operation.

▶ **Definition 4** (Temporal Modular-width). *Suppose a temporal graph $\mathcal{G}$ can be constructed by the algebraic expression A which uses the following operations:*

1. *Creating an isolated vertex.*
2. *Taking the disjoint union of two temporal graphs.*
3. *Taking the complete join of two temporal graphs at a set of times T. That is, for graphs $\mathcal{G}_1 = ((V_1, E_1), \lambda_1)$ and $\mathcal{G}_2 = ((V_2, E_2), \lambda_2)$, $V(\mathcal{G}_1 \otimes \mathcal{G}_2) = V(\mathcal{G}_1) \cup V(\mathcal{G}_2)$ and $\varepsilon(\mathcal{G}_1 \otimes \mathcal{G}_2) = \varepsilon(\mathcal{G}_1) \cup \varepsilon(\mathcal{G}_2) \cup \{(vw, t) : v \in V(\mathcal{G}_1), w \in V(\mathcal{G}_2)$ and $t \in T\}$.*
4. *The substitution of temporal graphs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ into a temporal graph $\mathcal{G}'$ with vertices $v_1, \ldots, v_n$. This gives the graph $\mathcal{G}'(\mathcal{G}_1, \ldots, \mathcal{G}_n)$ with vertex set $\bigcup_{1 \leq i \leq n} V(\mathcal{G}_i)$ and time-edge set $\bigcup_{1 \leq i \leq n} \varepsilon(\mathcal{G}_i) \cup \{(vw, t) : v \in V(\mathcal{G}_i), w \in V(\mathcal{G}_j),$ and $(v_i v_j, t) \in \varepsilon(\mathcal{G}')\}$.*

The width of an expression A is the maximum number of operands in an occurrence of the operation 4 in A. The temporal modular-width *of G is this minimum width of an expression A which constructs G.*

As for temporal cliquewidth, we observe that the temporal modular-width of a temporal graph is equal to the modular-width of the underlying graph if all edges have the same temporal assignment. It follows that, as in the static case, the temporal modular-width bounds the length of the longest induced path in the underlying graph.

We now argue that, as claimed, bounding the temporal modular-width of a temporal graph is a strictly stronger restriction than bounding the temporal cliquewidth.

▶ **Theorem 3** ($\star$). *For any temporal graph $\mathcal{G} = (G, \lambda)$, the temporal cliquewidth is upper bounded by the temporal modular-width.*

Habib and Paul [24] describe a simple algorithm for finding the modular decomposition of a static graph. This operates by finding and repeatedly adding *splitters* to a candidate module - intuitively, splitters are vertices outside of a module that distinguish between vertices inside of a candidate module. We can use a similar splitter-closure method to find the unique maximal temporal modular decomposition:

▶ **Theorem 4** (⋆). *We can find the maximal temporal modular decomposition in time $O(n^4\Lambda)$, where n is the number of vertices in the temporal graph.*

3.1 Star Exploration

In this section we consider the following problem, demonstrating that it remains NP-hard even on temporal graphs with temporal cliquewidth at most three, but that it is solvable in constant time on graphs with bounded temporal modular-width (provided we are given the temporal modular-width of the graph). This problem was first introduced by Akrida et al. [2].

StarExp(τ)

Input: A temporal star (S_n, λ) where $|\lambda(e)| \leq \tau$ for every edge e in the star S_n.
Output: Is there a strict temporal walk, starting and finishing at the centre of the star, which visits every vertex of S_n?

We begin with a simple observation about the temporal cliquewidth of temporal graphs whose underlying graph is a star.

▶ **Lemma 5** (⋆). *A temporal star S_n has temporal cliquewidth at most 3.*

StarExp(τ) is known to be NP-hard even for constant τ [2, 6]. Then, by Lemma 5, StarExp(τ) is an example of a problem which is NP-hard on graphs of bounded temporal cliquewidth. We now show that StarExp(τ) is tractable on graphs of bounded temporal modular-width. We begin with the following lemma.

▶ **Lemma 6** (⋆). *If a temporal star S_n has temporal modular-width at most k, the leaves of S_n can be partitioned into $k-1$ subsets such that, if u and v are in the same subset and c is the central vertex in the star, the edges uc and cv are active at the same times.*

▶ **Lemma 7.** *If there are strictly more than $\tau/2$ leaves of S_n whose incident edges are active at the same times, we have a no-instance of StarExp(τ).*

Proof. By definition of the problem StarExp(τ), each edge in the star is active at most τ times. Therefore, if there are $\lfloor\tau/2\rfloor + 1$ vertices $u_1, \ldots, u_{\lfloor\tau/2\rfloor+1}$ such that $\lambda(u_1c) = \cdots = \lambda(u_{\lfloor\tau/2\rfloor+1}c)$ where c is the central vertex in the star, there is no temporal walk which starts at c and visits all of $u_1, \ldots, u_{\lfloor\tau/2\rfloor+1}$. To see this, note that visiting a leaf and returning requires the use of two distinct time-edges. Therefore, a walk visiting any $\lfloor\tau/2\rfloor + 1$ leaves which departs from and returns to c must consist of at least $\tau + 1$ distinct time-edges. This is not possible if these vertices have incident edges are active at the same times and $|\lambda(uc)| \leq \tau$. ◀

▶ **Theorem 8.** *StarExp(τ) is solvable in $(k\tau)!(k\tau)^{O(1)}$ time when the temporal modular-width of the graph is at most k.*

Proof. Given that the temporal modular-width of the graph is at most k, we check whether the number of leaves is more than $(k-1)\lfloor\tau/2\rfloor$. If the number of leaves is at least $(k-1)\lfloor\tau/2\rfloor+1$ then, by the pigeon-hole principle and Lemma 6, there must be at least $\lfloor\tau/2\rfloor + 1$ leaves whose edges to the central vertex are active at exactly the same times. In this case, by Lemma 7, we conclude that we have a no-instance of StarExp(τ).

Else, we have at most $(k-1)\lfloor\tau/2\rfloor < k\tau$ leaves. Note that, given an ordering of leaves to visit, we can check whether such a walk is valid in time polynomial in k and τ: we need only check for each leaf in order whether there are two appearances of its incident edge following the time-edges used to visit the previous leaf (and if so, greedily use the first two such appearances). Since there are fewer than $(k\tau)!$ possible orderings of the leaves, we can check each possibility in turn in time $(k\tau)!(k\tau)^{O(1)}$. ◀

4 Tractability with respect to Temporal Neighbourhood Diversity

We now turn our attention to our final parameter, temporal neighbourhood diversity, which is the most restrictive and hence allows for the most problems to be solved efficiently. In Section 4.1 we demonstrate that Temporal Graph Burning is solvable in polynomial time when the temporal neighbourhood diversity is bounded by a constant (in fact we give an fpt-algorithm with respect to this parameterisation), even though the problem remains NP-hard when restricted to temporal graphs with constant temporal modular-width. To illustrate further techniques that may be used to design efficient algorithms on graphs of bounded temporal neighbourhood diversity, in Section 4.2 we also give an fpt-algorithm for the problem SingMinReachDelete with a single source vertex.

We begin with the formal definition of temporal neighbourhood diversity. Once again, the definition is modelled on that for static graphs, which was first introduced by Lampis [28] and adpated by Ganian [21] to describe uncoloured graphs. In a static graph, we define the *neighbourhood* $N(v)$ of a vertex v as the set of vertices which share an edge with v.

▶ **Definition 5** (Type, Definition 2.2 [21])**.** *Two vertices v, v' have the same* type *if and only if $N(v) \setminus \{v'\} = N(v') \setminus \{v\}$.*

▶ **Definition 6** (Neighbourhood Diversity, Definition 2 [28])**.** *A graph $G = (V, E)$ has* neighbourhood diversity *at most k if and only if there exists a partition of $V(G)$ into at most k sets where all vertices in each set have the same type. We refer to this partition as a* neighbourhood partition.

We note that the neighbourhood diversity of a graph can be computed in linear time [28].

We now define the analogous temporal parameter, where we require that the edges between sets are all active at the same times.

▶ **Definition 7** (Temporal Neighbourhood)**.** *The* temporal neighbourhood *of a vertex v in a temporal graph (G, λ) is the set $TN(v)$ of vertex time pairs (w, t) where $(w, t) \in TN(V)$ if and only if $vw \in E(G)$ and $t \in \lambda(vw)$.*

▶ **Definition 8** (Temporal Type)**.** *Two vertices u, v have the same* temporal type *if and only if $\{(w, t) \in TN(v) : w \neq v\} = \{(w, t) \in TN(v') : w \neq u\}$.*

▶ **Definition 9** (Temporal Neighbourhood Diversity)**.** *A graph $\mathcal{G}$ has* temporal neighbourhood diversity *at most k if and only if there exists a partition of $V(\mathcal{G})$ into at most k sets where all vertices in each set have the same temporal type. We refer to this partition as a* temporal neighbourhood partition.

It is immediate from this definition that, when all edges are assigned the same times, the temporal neighbourhood diversity of the graph is the same as the neighbourhood diversity of the underlying graph. We now argue that, as in the static case, the subgraph induced by any class must form a clique or independent set; moreover, in the temporal setting, this must be true at every timestep.

▶ **Lemma 9** ($\star$)**.** *At any snapshot $\mathcal{G}_t$ of $\mathcal{G}$, the subgraph induced by the vertices in a class X of a temporal neighbourhood partition of $\mathcal{G}$ either forms an independent set or a clique.*

As stated, temporal neighbourhood diversity is the most restrictive parameter in our hierarchy. Observe that for any temporal graph $\mathcal{G}$ the temporal modular-width is upper bounded by the temporal neighbourhood diversity: each class in the temporal neighbourhood partition forms a module, and it follows from Lemma 9 that each of these modules can be constructed by operations 1, 2, and 3 of temporal modular-width construction.

Finally, we argue that we can compute the temporal neighbourhood diversity efficiently.

▶ **Proposition 10** ($\star$). *The temporal neighbourhood diversity of a temporal graph (G, λ) can be calculated in $O(\Lambda n^3)$ time.*

4.1 Temporal Graph Burning

In this section we define Temporal Graph Burning, a temporal analogue of the static Graph Burning problem first proposed by Bonato et al.[5]. Informally, a fire is placed at a new vertex in each time-step and the fire spreads along incident active edges. The problem asks if, after h placements of fire, all vertices are burning. Static Graph Burning is NP-hard on general graphs [4] and was recently shown to be in FPT parameterised by static modular width [27]. In contrast, we prove that Temporal Graph Burning remains NP-hard on graphs with constant temporal modular-width. This difference arises from the fact that, in the static setting, the length of a longest induced path in the graph (which is upper bounded by the modular-width) gives an upper bound on the time taken to burn the graph. In the temporal setting, on the other hand, the times assigned to edges mean that even graphs with small diameter may take many steps to burn. In contrast, we show that Temporal Graph Burning can be solved in time $O(n^5 \Lambda k! 4^k)$ on temporal graphs with n vertices, lifetime Λ and temporal neighbourhood diversity k.

The Temporal Graph Burning problem asks how quickly a fire can be spread over the vertices of a temporal graph in the following discrete time process, where a fire is placed at a vertex of a graph on each timestep.

1. At time $t = 0$ a fire is placed at a chosen vertex. All other vertices are unburnt.
2. At all times $t \geq 1$, the fire spreads, burning all vertices u adjacent to an already burning vertex v where the edge between u and v is active at time t. Then, another fire is placed at a chosen vertex.
3. This process ends once all vertices are burning.

We refer to a sequence of vertices at which fires are placed as a strategy.

▶ **Definition 10** (Burning Strategy). *A* burning strategy *for a temporal graph (G, λ) is a sequence of vertices $S = s_1, s_2, ..., s_h$ such that $s_i \in V(G)$ for all $i \leq h$, and on each timestep i a fire is placed at s_i.*

We say that a strategy $S = s_1, s_2, ..., s_h$ has length h. For convenience, we allow for strategies that place fires at already burning vertices, although it is worth noting that such moves may be omitted. If every vertex in the graph is burning after a strategy is played, we say that strategy is successful.

▶ **Definition 11** (Successful Burning Strategy). *A burning strategy $S = s_1, s_2, ..., s_h$ for a temporal graph (G, λ) is* successful *if every vertex in G is burning on timestep h when the moves from S are played.*

The decision problem asks how many timesteps it takes to burn a given temporal graph.

Temporal Graph Burning

Input: A temporal graph (G, λ) and an integer h.
Output: Does there exist a successful burning strategy for (G, λ) of length less than or equal to h?

This problem is in NP, with a strategy providing a certificate. Given a strategy it can be checked in polynomial time if it is successful and of length less than or equal to h by simulating temporal graph burning on the input graph.

We show that Temporal Graph Burning is NP-hard even on graphs of bounded temporal modular-width. This is achieved by reducing from (3, 2B)-SAT, an NP-hard variant of the Boolean satisfiability problem in which each variable appears exactly twice both positively and negatively [3]. Our reduction produces a graph where each edge is active on exactly one timestep, and furthermore every connected component has a bounded temporal neighbourhood diversity, and hence the graph has bounded temporal modular-width by our earlier observation.

▶ **Theorem 11** ($\star$). *Temporal Graph Burning is NP-hard even when restricted to graphs with constant temporal modular-width.*

We now show that Temporal Graph Burning is solvable efficiently when the temporal neighbourhood diversity of the input graph is bounded. Throughout we assume that the lifetime Λ of the input temporal graph is at most the number of vertices n, as it is possible to burn any temporal graph in n timesteps by placing a fire at every vertex in turn. We begin by defining notation for the burning set of vertices on a given timestep when a strategy is played.

▶ **Definition 12** (Burning Set). *Given a strategy S the* burning set *$B_t(S)$ at timestep t is the set of burning vertices immediately after a fire is placed on timestep t when S is played.*

We now give a number of results about how successful strategies can be modified to have certain desirable properties; these results allow us to bound the number of possible strategies we must check to determine whether there is a successful strategy of the desired length.

▶ **Lemma 12** ($\star$). *Let S be a successful strategy for (G, λ). Suppose that there is some timestep $t_1 < |S|$ and strategy R with $|R| = |S|$ such that $B_{t_1}(S) \subseteq B_{t_1}(R)$ and on every timestep after t_1, R places a fire at the same vertex as S. Then R is also successful.*

▶ **Lemma 13** ($\star$). *Let (G, λ) be a temporal graph with temporal neighbourhood partition $(X_i)_{i \in I}$. Now let S be a strategy that burns this graph, and let u be a vertex at which S places a fire on a timestep t_1, and X_i be the class to which this vertex belongs. Let S' be a strategy which plays as follows until t_2, for any timestep $t_2 > t_1$:*

$$s'_t = \begin{cases} s_t & \text{if } t < t_1 \\ s_{t+1} & \text{if } t_1 \leq t < t_2 \\ u & \text{if } t = t_2 \end{cases}$$

Providing there exists a vertex $w \in X_i$ which is burning before the end of timestep t_1 when S' is played, we have that $B_{t_2}(S) \subseteq B_{t_2}(S')$.

▶ **Lemma 14** ($\star$). *Let (G, λ) be a temporal graph with temporal neighbourhood partition $(X_i)_{i \in I}$. Let S and S' both be strategies with $|S'| = |S|$, such that on every timestep, S and S' both place fires in the same class, that is, for any $i \leq |S|$ we have that $\{s_i, s'_i\} \subseteq X_j$.*

Furthermore, assume that S' places a fire at an already burning vertex on a timestep i if and only if S also places a fire at an already burning vertex on timestep i. Then S is successful if and only if S' is.

▶ **Definition 13** (Placement Classes). *The placement classes for a strategy S denoted $C(S)$ is the set of classes from the temporal neighbourhood partition in which S places fires.*

▶ **Lemma 15** ($\star$). *Given a temporal graph (G, λ), let S be any successful strategy. There is then a successful strategy S' with $|S'| = |S|$, and $C(S') = C(S)$, such that the first $|C(S')|$ burns are in distinct equivalence classes in the temporal neighbourhood partition.*

Finally we show that, given a strategy S that places fires only in distinct classes for the first $|C(S)|$ moves, we can arbitrarily reorder all subsequent moves made after timestep $|C(S)|$.

▶ **Lemma 16** ($\star$). *Let (G, λ) be a temporal graph, and S a successful strategy such that the first $|C(S)|$ fires placed by S are placed in distinct classes from the temporal neighbourhood partition. Let $f : [|C(S)| + 1, |S|] \to [|C(S)| + 1, |S|]$ be any bijection. Then the strategy S' given by*

$$s'_t = \begin{cases} s_t & \text{if } t \leq |C(S)| \\ s_{f(t)} & \text{otherwise} \end{cases}$$

is successful, and burns the graph in the same or less time as S.

We now present an algorithm for Temporal Graph Burning (Algorithm 1), and show that this algorithm is an fpt-algorithm with respect to temporal neighbourhood diversity.

Algorithm 1 TND Graph Burning Algorithm.

```
Input: A temporal graph $\mathcal{G}$, and an integer $k$.
Output: True if and only if there exists a successful burning strategy of length at most $h$.
Compute the temporal neighbourhood partition $\Theta$ of $(G, \lambda)$.
for all possible subsets $A \subseteq \Theta$ do
    for all possible orderings of $A$ do
        for all possible subsets $B \subseteq A$ do
            Compute a strategy that first places a fire in order in every class from $A$, and
            then places fires at every unburnt vertex in $B$ in any order.
            if this strategy is successful and consists of $k$ or fewer moves then
                return true.
If no such strategy is found, return false.
```

▶ **Lemma 17** ($\star$). *The TND Graph Burning Algorithm returns true for a temporal graph (G, λ) and integer h if and only if there exists a strategy S that burns the graph in h or fewer timesteps.*

This allows us to obtain fixed parameter tractability, as bounding the temporal neighbourhood diversity bounds the number of such strategies that we have to check.

▶ **Theorem 18** ($\star$). *Temporal Graph Burning is solvable in time $O(n^5 \Lambda k! 4^k)$, where n is the size of the input temporal graph $\mathcal{G}$, Λ the lifetime, and k the temporal neighbourhood diversity. If the temporal neighbourhood partition is given, we obtain a runtime of $O(n^2 \Lambda k! 4^k)$.*

4.2 Minimum Reachability Edge Deletion

Here we give another problem which is tractable with respect to temporal neighbourhood diversity. Given a specified source vertex, we seek a minimum set of edge appearances that can be deleted to limit the number of vertices reachable from that source.

We say a vertex v is *temporally reachable* from a vertex u in $\mathcal{G}$ if there exists a temporal path from u to v. We say a vertex v is temporally reachable from a set S if there is a vertex in S from which v is temporally reachable. The *reachability set* $\text{reach}(v)$ of a vertex v is the set of vertices temporally reachable from v. We can now give the formal problem definition; this is a special case of the problem MinReachDelete studied by Molter et al. [35] in which multiple sources are allowed.

Singleton Minimum Temporal Reachability Edge Deletion (SingMinReachDelete)

Input: A temporal graph $\mathcal{G} = (G, \lambda)$, a vertex $v_s \in V(G)$ and positive integer r.
Output: What is the cardinality of the smallest set of time-edges E such that the vertex v_s has temporal reachability at most r after their deletion from $\mathcal{G}$?

SingMinReachDelete was shown by Enright et al. [16] to be NP-hard (and W[1]-hard parameterised by the maximum number of vertices that are allowed to be reached following deletion) even when the lifetime of the input temporal graph is 2 and every edge is active at exactly one timestep. In their problem, they ask if there exists a deletion such that no vertex in the resulting temporal graph reaches more than r vertices. While the result of [16] is for a version of the problem when the source set S is the entire vertex set, it is clear from the construction that hardness also holds with a single source vertex.

We show that this problem is in FPT when parameterised by temporal neighbourhood diversity and the *temporality* of the input graph $\tau(\mathcal{G})$, which was defined by Mertzios et al. [32] to be the maximum number of times any edge appears. When the temporal graph in question is clear from context, we just refer to τ. We note that it remains open whether the problem belongs to FPT parameterised by temporal neighbourhood diversity alone, or indeed parameterised by temporal modular width or temporal cliquewidth; the techniques we use here do not extend naturally to these less restrictive settings.

We now give a formal statement of our result.

▶ **Theorem 19** ($\star$). *SingMinReachDelete is solvable in time $g(k,\tau)\log^{O(1)} r + \Lambda n^3$, where g is a computable function. If a temporal neighbourhood decomposition is given, we can solve the problem in time $g(k,\tau)\log^{O(1)} r$.*

Note that, given two vertices of the same temporal type, their reachability sets must consist of the same vertices except for the vertices themselves; as a result, the reachability sets of vertices in a given class all have the same cardinality. Moreover, all vertices of the same type are first reached from the source at the same time (where we say a vertex is "first reached" at time t if the final time-edge in an earliest-arriving temporal path from the source is at time t). Our strategy for solving SingMinReachDelete is then as follows. We partition each class according to the time at which the vertices are first reached after the deletion of time-edges, and consider all possibilities for which of these subclasses are non-empty. Given a function ϕ telling us which subclasses are non-empty, we argue that we can determine efficiently whether there is indeed a deletion such that precisely these subclasses are non-empty, and if so compute exactly the pairs of subclasses between which we must delete time-edges to achieve this. For a fixed ϕ, we then encode the problem as an instance of Integer Quadratic Programming, where the variables are the sizes of the subclasses and the objective function seeks to minimise the number of time-edges we must delete, and use the algorithm of Lokshtanov [29].

5 Conclusion and Open Questions

We have described three temporal parameters that form a hierarchy mirroring the one formed by their static analogues, and that can all be small when the temporal graph is dense at every timestep. We provide examples of problems demonstrating that there is a separation between the classes of problems admitting efficient algorithms when each of the parameters is bounded. As is the case for the corresponding static parameters, we expect that there will be many problems for which these temporal parameters give fixed-parameter tractability, and suggest exploration of temporal extensions of static problems for which there are known fpt-algorithms as future work. From a practical perspective, it would also be interesting to investigate the values of these new parameters on dense real-world temporal networks.

One of the most celebrated results involving static cliquewidth is a metatheorem due to Courcelle et al. [11] which guarantees the existence of a linear-time algorithm for any problem expressible in a suitable fragment of logic (MSO_1) on graphs of bounded cliquewidth. It is a natural question whether an analogous metatheorem exists for temporal cliquewidth. A promising approach might be to encode a temporal graph as an arbitrary relational structure (as has been done for a temporal version of treewidth [18]). A major challenge here, however, is that to the best of our knowledge there is no single notion of cliquewidth for relational structures: several alternatives have been introduced [1, 12], but none has all of the desirable properties. Moreover, we believe that any encoding of a temporal graph of bounded temporal cliquewidth as a relational structure that preserves all the information in the original is unlikely to have bounded width for any cliquewidth-style measure unless we also bound the lifetime of the temporal graph. Nevertheless, this general direction merits further investigation, and there is potential for a useful metatheorem even if it is necessary to further restrict the fragment of logic considered or the structure of the temporal graph.

References

1 Hans Adler and Isolde Adler. A note on clique-width and tree-width for structures. *CoRR*, abs/0806.0103, 2008. arXiv:0806.0103.

2 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos Raptopoulos. The temporal explorer who returns to the base. *Journal of Computer and System Sciences*, 120:179–193, September 2021. doi:10.1016/j.jcss.2021.04.001.

3 Piotr Berman, Marek Karpinski, and Alex D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electron. Colloquium Comput. Complex.*, TR03-049, 2003. arXiv:TR03-049.

4 Stéphane Bessy, Anthony Bonato, Jeannette C. M. Janssen, Dieter Rautenbach, and Elham Roshanbin. Burning a graph is hard. *Discret. Appl. Math.*, 232:73–87, 2017. doi:10.1016/J.DAM.2017.07.016.

5 Anthony Bonato, Jeannette C. M. Janssen, and Elham Roshanbin. Burning a graph as a model of social contagion. In Anthony Bonato, Fan Chung Graham, and Pawel Pralat, editors, *Algorithms and Models for the Web Graph - 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings*, volume 8882 of *Lecture Notes in Computer Science*, pages 13–22. Springer, 2014. doi:10.1007/978-3-319-13123-8_2.

6 Benjamin Merlin Bumpus and Kitty Meeks. Edge Exploration of Temporal Graphs. *Algorithmica*, 85(3):688–716, March 2023. doi:10.1007/s00453-022-01018-7.

7 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding Temporal Paths Under Waiting Time Constraints. *Algorithmica*, 83(9):2754–2802, September 2021. doi:10.1007/s00453-021-00831-w.

8 Arnaud Casteigts, Kitty Meeks, George B. Mertzios, and Rolf Niedermeier. Temporal Graphs: Structure, Algorithms, Applications (Dagstuhl Seminar 21171). *Dagstuhl Reports*, pages 16–46, 2021. Publisher: Schloss Dagstuhl – Leibniz Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/DagRep.11.3.16, doi:10.4230/DagRep.11.3.16.

9 Gennaro Cordasco, Luisa Gargano, and Adele A. Rescigno. Iterated Type Partitions. In Leszek Gąsieniec, Ralf Klasing, and Tomasz Radzik, editors, *Combinatorial Algorithms*, Lecture Notes in Computer Science, pages 195–210, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-48966-3_15.

10 Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Applied Mathematics*, 160(6):834–865, April 2012. doi:10.1016/j.dam.2011.03.020.

11 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory of Computing Systems*, 33(2):125–150, April 2000. doi:10.1007/s002249910009.

12 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Cambridge University Press, 1 edition, June 2012. doi:10.1017/CBO9780511977619.

13 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218–270, April 1993. doi:10.1016/0022-0000(93)90004-G.

14 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114, April 2000. doi:10.1016/S0166-218X(99)00184-5.

15 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21275-3.

16 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, August 2021. doi:10.1016/j.jcss.2021.01.007.

17 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-Width is NP-Complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, January 2009. doi:10.1137/070687256.

18 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. As Time Goes By: Reflections on Treewidth for Temporal Graphs. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms: Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 49–77. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-42071-0_6.

19 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized Algorithms for Modular-Width. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, Lecture Notes in Computer Science, pages 163–176, Cham, 2013. Springer International Publishing. doi:10.1007/978-3-319-03898-8_15.

20 T. Gallai. Transitiv orientierbare Graphen. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(1):25–66, March 1967. doi:10.1007/BF02020961.

21 Robert Ganian. Using Neighborhood Diversity to Solve Hard Problems, January 2012. arXiv:1201.3091 [cs]. doi:10.48550/arXiv.1201.3091.

22 Frank Gurski. A comparison of two approaches for polynomial time algorithms computing basic graph parameters, June 2008. arXiv:0806.4073 [cs]. doi:10.48550/arXiv.0806.4073.

23 Roman Haag, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Feedback edge sets in temporal graphs. *Discrete Applied Mathematics*, 307:65–78, January 2022. doi:10.1016/j.dam.2021.09.029.

24 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Comput. Sci. Rev.*, 4(1):41–59, 2010. doi:10.1016/J.COSREV.2010.01.001.

25 Danny Hermelin, Yuval Itzhaki, Hendrik Molter, and Rolf Niedermeier. Temporal interval cliques and independent sets. *Theoretical Computer Science*, 961:113885, June 2023. doi:10.1016/j.tcs.2023.113885.

26 Petter Holme and Jari Saramäki. Temporal networks. *Physics Reports*, 519(3):97–125, October 2012. URL: https://www.sciencedirect.com/science/article/pii/S0370157312000841, doi:10.1016/j.physrep.2012.03.001.

27 Yasuaki Kobayashi and Yota Otachi. Parameterized complexity of graph burning. *Algorithmica*, 84(8):2379–2393, 2022. doi:10.1007/S00453-022-00962-8.

28 Michael Lampis. Algorithmic Meta-theorems for Restrictions of Treewidth. *Algorithmica*, 64(1):19–37, September 2012. doi:10.1007/s00453-011-9554-x.

29 Daniel Lokshtanov. Parameterized Integer Quadratic Programming: Variables and Coefficients, April 2017. arXiv:1511.00310 [cs]. doi:10.48550/arXiv.1511.00310.

30 Bernard Mans and Luke Mathieson. On the treewidth of dynamic graphs. *Theoretical Computer Science*, 554:217–228, October 2014. doi:10.1016/j.tcs.2013.12.024.

31 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1):189–241, April 1999. doi:10.1016/S0012-365X(98)00319-7.

32 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal Network Optimization Subject to Connectivity Constraints. *Algorithmica*, 81(4):1416–1449, April 2019. doi:10.1007/s00453-018-0478-6.

33 George B. Mertzios, Hendrik Molter, Rolf Niedermeier, Viktor Zamaraev, and Philipp Zschoche. Computing maximum matchings in temporal graphs. *Journal of Computer and System Sciences*, 137:1–19, November 2023. doi:10.1016/j.jcss.2023.04.005.

34 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016. ISBN: 1542-7951 Publisher: Taylor & Francis.

35 Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal Reachability Minimization: Delaying vs. Deleting. In Filippo Bonchi and Simon J. Puglisi, editors, *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 76:1–76:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.MFCS.2021.76.

36 Marc Tedder, Derek Corneil, Michel Habib, and Christophe Paul. Simpler Linear-Time Modular Decomposition Via Recursive Factorizing Permutations. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 634–645, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-70575-8_52.

37 Tiphaine Viard, Matthieu Latapy, and Clémence Magnien. Computing maximal cliques in link streams. *Theoretical Computer Science*, 609:245–252, January 2016. doi:10.1016/j.tcs.2015.09.030.

A Robust Measure on FDFAs Following Duo-Normalized Acceptance

Dana Fisman
The Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Emmanuel Goldberg
The Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Oded Zimerman
The Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

Families of DFAs (FDFAs) are a computational model recognizing ω-regular languages. They were introduced in the quest of finding a Myhill-Nerode theorem for ω-regular languages and obtaining learning algorithms. FDFAs have been shown to have good qualities in terms of the resources required for computing Boolean operations on them (complementation, union, and intersection) and answering decision problems (emptiness and equivalence); all can be done in non-deterministic logarithmic space. In this paper we study FDFAs with a new type of acceptance condition, *duo-normalization*, that generalizes the traditional *normalization* acceptance type. We show that duo-normalized FDFAs are advantageous to normalized FDFAs in terms of succinctness as they can be exponentially smaller. Fortunately this added succinctness doesn't come at the cost of increasing the complexity of Boolean operations and decision problems — they can still be preformed in NLOGSPACE.

An important measure of the complexity of an ω-regular language is its position in the Wagner hierarchy (aka the Rabin Index). It is based on the inclusion measure of Muller automata, and for the common ω-automata there exist algorithms computing their position. We develop a similarly robust measure for duo-normalized (and normalized) FDFAs, which we term the *diameter measure*. We show that the diameter measure corresponds one-to-one to the position in the Wagner hierarchy. We show that computing it for duo-normalized FDFAs is PSPACE-complete, while it can be done in NLOGSPACE for traditional FDFAs.

2012 ACM Subject Classification Theory of computation; Theory of computation → Formal languages and automata theory; Theory of computation → Automata over infinite objects

Keywords and phrases Omega-Regular Languages, Families of DFAs, Complexity Measure, Wagner Hierarchy, Rabin Index

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.53

Related Version *Full Version*: `https://arxiv.org/abs/2310.16022` [12]

Funding *Oded Zimerman*: Supported by ISF grant 2507/21.

1 Introduction

Regular languages of finite words possess a natural canonical representation — the unique minimal DFA. The essence of the representation lies in a right congruence relation for a language L saying that two words x and y are *equivalent*, denoted $x \sim_L y$, if and only if $xz \in L \iff yz \in L$ for every finite word $z \in \Sigma^*$. The famous Myhill-Nerode theorem [20, 21] relates the equivalence classes of $\sim_L$ to the set of words reaching a state of the minimal DFA.

For regular languages of infinite words the situation is more complex. First, there is no unique minimal automaton for any of the common ω-automata acceptance conditions (Büchi, Muller, Rabin, Streett and parity). Second, one can indeed define two finite words x and y to be *equivalent* with respect to an ω-regular language L, denoted $x \sim_L y$, if $xz \in L \iff yz \in L$

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 53; pp. 53:1–53:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for every infinite word $z \in \Sigma^\omega$. However, there is no one-to-one correspondence between these equivalence classes and a minimal ω-automaton for L. Consider for instance the language L_1 stipulating that aab occurs infinitely often. The right congruence relation $\sim_{L_1}$ has only one equivalence class, yet clearly an automaton for L_1 needs more than one state.

A quest for a characterization of an ω-regular language L, relating equivalence classes of a semantic definition of L to states of an automaton for L, has led to the development of families of right concurrences (FORCs) [19] and families of DFAs (FDFAs) [14, 2]. Several canonical FDFAs were introduced over the years, the periodic FDFA [7], the syntactic FDFA [19], the recurrent FDFA [2], and the limit FDFA [17]. All these representations have a one-to-one correspondence between the equivalence classes of semantic right congruence relations and the states of the respective automata. This is very satisfying in the sense that they induce a semantic canonical representation, ie one that is agnostic to a particular automaton; and this is a beneficial property when it comes to learning [2, 18]. FDFAs have additional good qualities — computing Boolean operations on them (complementation, union, and intersection) and answering decision problems (emptiness and equivalence) can all be done cheaply, in non-deterministic logarithmic space [1].

Loosely speaking, an FDFA is composed of a *leading automaton* $\mathcal{Q}$ and a family of *progress DFAs* $\{\mathcal{P}_q\}$, one for each state q of $\mathcal{Q}$. FDFAs consider only ultimately periodic words, ie words of the form $u(v)^\omega$ for $u \in \Sigma^*$ and $v \in \Sigma^+$. Since two ω-regular languages recognize the same language if and only if they agree on the set of ultimately periodic words [6, 7], this is not really a limitation. Exact acceptance of an ultimately periodic word (u, v) representing uv^ω is determined by checking acceptance of v in the progress DFA corresponding to the state reached in the leading automaton by reading u. Normalized acceptance is done by first *normalizing* the word wrt the leading automaton — this means considering a decomposition (uv^i, v^j) of uv^ω such that v^j loops on the state of the leading automaton reached by reading uv^i. This normalization was introduced as it leads to an exponential save in the number of states [2]. In this paper we consider a new acceptance condition for FDFAs, which we term *duo-normalization*, which considers decompositions (uv^i, v^j) where in addition v^j closes a loop on the state it arrives at in the respective progress DFA. We term FDFAs with this new type of acceptance *duo-normalized* FDFAs. The notion of duo-normalization has appeared in the literature before [27, 9, 1, 5] and was suggested as an acceptance condition for FDFAs in the future work of [5].

We show that duo-normalized FDFAs also enjoy the good qualities of computing Boolean operations and answering decision problems in non-deterministic logarithmic space. In terms of succinctness we show that they can be exponentially smaller than normalized FDFAs.

We are also interested in the problem of finding their position in the *Wagner hierarchy*, a hierarchy reflecting the complexity of ω-regular properties, that often correlates to the complexity of algorithms on ω-regular languages and games [29, 13, 3]. It is noted in [10, Sec. 5] that while for ω-automata there are algorithms for computing their position in the Wagner hierarchy, there is no clear way to relate the structure of a particular FDFA to its Wagner position.

In [26] Wagner defined a complexity measure on Muller automata: the *inclusion measure*. Wagner showed that the inclusion measure is robust in the sense that any two Muller automata for the same language (minimal or not) have the same inclusion measure. This is thus a semantic property of the language. Since the inclusion measure is unbounded it induces an infinite hierarchy. The position on the Wagner hierarchy has been shown to be tightly related to the minimal number of colors required in a parity automaton, and the minimal number of pairs required in a Rabin/Street automaton. Deterministic Büchi and coBüchi (which are less expressive than deterministic Muller/Rabin/Streett/parity automata, that are capable of recognizing all the ω-regular languages) lie in the bottom levels of the hierarchy. Given a deterministic ω-automaton (Büchi, coBüch, Muller, Rabin, Streett, or parity), its position in the Wagner hierarchy can be computed in polynomial time [28, 8, 22].

We develop a syntactic notion of a measure on FDFAs, that we term *the diameter measure*. Loosely speaking it relates to chains of prefixes $v_1 \prec v_2 \prec \ldots \prec v_k$ such that $u(v_i)^\omega \in L$ iff $u(v_{i+1})^\omega \notin L$, and moreover, each of the words v_i is *persistent* in the progress DFA of some $u \in \Sigma^*$. The precise definition of the term *persistent* and *persistent chains* is deferred to § 4. We show there that this measure is robust in the sense that computing it on two FDFAs for the same language will give the same result. The proof is by relating it to the position on the Wagner hierarchy. We show that computing the Wagner position of a duo-normalized FDFA can be done in PSPACE and it is PSPACE-complete, whereas for normalized FDFAs this computation can be done in NLOGSPACE. So this is one place where the added succinctness of duo-normalized FDFAs comes at a price.

The rest of the paper is organized as follows. We give some basic definitions and explain the Wagner hierarchy in § 2. We introduce duo-normalized FDFAs in § 3 where we show that it is not more expensive to compute the Boolean operations on them, or to answer emptiness and equivalence about them. § 4 is devoted to defining the *diameter measure* and proving that its computation is PSPACE-complete. § 5 provides several succinctness results relating duo-normalized FDFAs and normalized FDFAs, including results regarding succinctness of previously studied canonical FDFAs and the *Colorful FDFA*, a canonical duo-normalized FDFA. We conclude with a short discussion in § 6. Due to space limitations, proofs are deferred to the appendix of the full version [12].

2 Preliminaries

We use $[i..j]$ for the set $\{i, i+1, \ldots, j\}$. A (complete deterministic) *automaton structure* is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ consisting of an alphabet Σ, a finite set Q of states, an initial state q_0, and a complete transition function $\delta : Q \times \Sigma \to Q$. A run of an automaton on a finite word $v = a_1 a_2 \cdots a_n$ is a sequence of states $q_0, q_1, \ldots, q_n$, starting with the initial state, such that for each $i \geq 0$, $q_{i+1} = \delta(q_i, a_i)$. A run on an infinite word is defined similarly and results in an infinite sequence of states. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta)$ be an automaton structure. We say that a word $w \in \Sigma^*$ reaches state q if the run of $\mathcal{A}$ on w ends in q, and use $\mathcal{A}(w)$ to denote q.

By augmenting an automaton structure with an acceptance condition α, obtaining a tuple $(\Sigma, Q, q_0, \delta, \alpha)$, we get an *automaton*, a machine that accepts some words and rejects others. An automaton accepts a word if the run on that word is accepting. For finite words the acceptance condition is a set $F \subseteq Q$ and a run on a word v is accepting if it ends in an accepting state, ie a state $q \in F$. For infinite words, there are various acceptance conditions in the literature. The common ones are Büchi, coBüchi, Muller, Rabin, Streett and parity. They are all defined with respect to the set of states visited infinitely often during a run. For a run $\rho = q_0 q_1 q_2 \ldots$ we define $\mathit{inf}(\rho) = \{q \in Q \mid \forall i \in \mathbb{N}.\ \exists j > i.\ q_j = q\}$. We focus here on the most common types — Büchi, coBüchi, Muller and parity.

- A *Büchi* (resp. *coBüchi*) acceptance condition is a set $F \subseteq Q$. A run of a Büchi (resp. coBüchi) automaton is accepting if it visits F infinitely (resp. finitely) often. That is, if $\mathit{inf}(\rho) \cap F \neq \emptyset$ (resp. $\mathit{inf}(\rho) \cap F = \emptyset$).
- A *parity* acceptance condition is a mapping $\kappa : Q \to \{0, 1, \ldots, k\}$ of a state to a number (referred to as a *color*). For a subset $Q' \subseteq Q$, we use $\kappa(Q')$ for the set $\{\kappa(q) \mid q \in Q'\}$. A run ρ of a parity automaton is accepting if the **minimal** color in $\kappa(\mathit{inf}(\rho))$ is **even**.

- A *Muller* acceptance condition is a set $\alpha = \{F_1, \dots, F_k\}$ where $F_i \subseteq Q$ for all $1 \leq i \leq k$. A run ρ of a Muller automaton is accepting if $\mathit{inf}(\rho) \in \alpha$. That is, if the set of states visited infinitely often by the run ρ is exactly one of the sets F_i specified in α.

We use DBA, DCA, DPA, and DMA as acronyms for deterministic (complete) Büchi, coBüchi, parity, and Muller automata, respectively. We use $[\![\mathcal{A}]\!]$ to denote the set of words accepted by a given automaton $\mathcal{A}$. Two automata $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if $[\![\mathcal{A}]\!]$=$[\![\mathcal{B}]\!]$. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ be a DFA and $q \in Q$. We use $\mathcal{A}_{[q}$ for $(\Sigma, Q, q, \delta, F)$, namely a DFA for the words exiting state q of $\mathcal{A}$.

The *syntactic right congruence relation* for an ω-language L relates two finite words x and y if there is no infinite suffix z differentiating them, that is, for $x, y \in \Sigma^*$, $x \sim_L y$ if $\forall z \in \Sigma^\omega (xz \in L \iff yz \in L)$. We use $[u]_{\sim_L}$ (or simply $[u]$) for the equivalence class of u induced by $\sim_L$. A right congruence $\sim$ can be naturally associated with an automaton structure (Σ, Q, q_0, δ) as follows: the set of states Q are the equivalence classes of $\sim$. The initial state q_0 is the equivalence class $[\epsilon]$. The transition function δ is defined by $\delta([u], \sigma) = [u\sigma]$. We use $\mathcal{A}[\sim]$ to denote the automaton structure induced by $\sim$.

2.1 The Wagner Hierarchy

Let $\mathcal{M} = (\Sigma, Q, q_0, \delta, \alpha)$ be a complete deterministic Muller automaton, where all states are reachable. We use the term *strongly connected component (SCC)* for a set of states $S \subseteq Q$ such that there is a non-empty path between every pair of states in S. Thus, if S is a singleton $\{q\}$ we require a self-loop on q for S to be an SCC. We use the term *MSCC* for a *maximal SCC*, that is, an SCC S such that no set $S' \supset S$ is an SCC. We say that an SCC $S \subseteq Q$ is *accepting* iff $S \in \alpha$. Otherwise we say that S is *rejecting.* We define the *positive inclusion measure* of $\mathcal{M}$, denoted $|\mathcal{M}|^+_\subset$, to be the maximal length of an inclusion chain $S_1 \subset S_2 \subset S_3 \subset \cdots \subset S_k$ of SCCs with alternating acceptance where S_1 is accepting. (Therefore for each $1 \leq i \leq k$ if i is odd then S_i is accepting, and if it is even then S_i is rejecting.) Likewise, we define the *negative inclusion measure* of $\mathcal{M}$, denoted $|\mathcal{M}|^-_\subset$, to be the maximal length of an inclusion chain where the first SCC is rejecting. Note that for any $\mathcal{M}$ the difference between $|\mathcal{M}|^+_\subset$ and $|\mathcal{M}|^-_\subset$ may be at most one, since by omitting the first element of a chain we remain with a chain shorter by one, and of the opposite sign. We use $L_{\infty aa \wedge \neg \infty bb}$ in Ex. 2.2 to illustrate the concepts explained throughout this section.

Wagner [26] showed that this measure is robust in the sense that any two DMAs that recognize the same language have the same positive and negative inclusion measures.

▶ **Theorem 2.1** (Robustness of the inclusion measures [26]). *Let $\mathcal{M}_1$, $\mathcal{M}_2$ be two DMAs where $[\![\mathcal{M}_1]\!] = [\![\mathcal{M}_2]\!]$. For $i \in \{1, 2\}$, let $|\mathcal{M}_i|^+_\subset = p_i$ and $|\mathcal{M}_i|^-_\subset = n_i$. Then $p_1 = p_2$ and $n_1 = n_2$.*

Since this measure is robust and since one can construct DMAs with arbitrarily long inclusion chains, the inclusion measure yields an infinite hierarchy of ω-regular languages. Formally, the classes of the Wagner hierarchy are defined as follows for a positive integer k:

$$\begin{aligned}
\mathbb{DM}^+_k &= \{L \mid \exists \text{ DMA } \mathcal{M} \text{ s.t. } [\![\mathcal{M}]\!] = L \text{ and } |\mathcal{M}|^+_\subset \leq k \text{ and } |\mathcal{M}|^-_\subset < k\} \\
\mathbb{DM}^-_k &= \{L \mid \exists \text{ DMA } \mathcal{M} \text{ s.t. } [\![\mathcal{M}]\!] = L \text{ and } |\mathcal{M}|^+_\subset < k \text{ and } |\mathcal{M}|^-_\subset \leq k\} \\
\mathbb{DM}^\pm_k &= \{L \mid \exists \text{ DMA } \mathcal{M} \text{ s.t. } [\![\mathcal{M}]\!] = L \text{ and } |\mathcal{M}|^+_\subset \leq k \text{ and } |\mathcal{M}|^-_\subset \leq k\}
\end{aligned}$$

The hierarchy is depicted in Fig. 3.1 (left). Note that if $\mathcal{A}$ is an ω-automaton, for any of the ω-automata types, then it can be recognized by a Muller automaton on the same structure. Transforming a Büchi $\mathcal{B}$ automaton with accepting states F to a Muller automaton $\mathcal{M}_\mathcal{B}$ yields an acceptance condition $\alpha_\mathcal{B} = \{F' \mid F' \cap F \neq \emptyset\}$. Note that for any $F' \in \alpha_\mathcal{B}$ and $F'' \supseteq F'$ it holds that $F'' \in \alpha_\mathcal{B}$. Therefore $|\mathcal{M}_\mathcal{B}|^+_\subset = 1$ and $|\mathcal{M}_\mathcal{B}|^-_\subset = 2$ (unless $[\![\mathcal{B}]\!] = \Sigma^\omega$ or

$[\![\mathcal{B}]\!] = \emptyset$). Hence all languages recognized by a DBA are in $\mathbb{DM}_2^-$. Dually, one can see that all languages recognized by a DCA are in $\mathbb{DM}_2^+$. It can be shown that a parity automaton for a language in $\mathbb{DM}_k^-$ can suffice with colors $\{1,\dots,k\}$ if k is odd and $\{0,\dots,k-1\}$ if it is even [22]. Likewise, a DPA for a language in $\mathbb{DM}_k^+$ can suffice with colors $\{0,\dots,k-1\}$ if k is odd and with $\{1,\dots,k\}$ otherwise. A DPA in $\mathbb{DM}_k^\pm$ requires $k+1$ colors starting with 0.

▶ **Example 2.2.** Fig. 3.1 (middle) shows a Muller automaton $\mathcal{M}$ for the language $L_{\infty aa \wedge \neg \infty bb}$. The inclusion chain $\{q_1,q_2\} \subset \{q_1,q_2,q_3\} \subset \{q_1,q_2,q_3,q_4\}$ is a negative inclusion chain of length 3 (since $\{q_1,q_2\}$ is rejecting, $\{q_1,q_2,q_3\}$ is accepting, and $\{q_1,q_2,q_3,q_4\}$ is rejecting). There are no negative inclusion chains of length 4, and there are no positive inclusion chains of length 3. (Note that $\{q_3\} \subset \{q_2,q_3\} \subset \{q_1,q_2,q_3\}$ is not an inclusion chain since $\{q_2,q_3\}$ is not an SCC.) We can thus conclude that $L_{\infty aa \wedge \neg \infty bb} \in \mathbb{DM}_3^-$. Consider the parity automaton $\mathcal{P}$ for $L_{\infty aa \wedge \neg \infty bb}$ defined on the same structure as $\mathcal{M}$. It uses the three colors $\{1,2,3\}$ in accordance with our conclusion that $L_{\infty aa \wedge \neg \infty bb} \in \mathbb{DM}_3^-$.

3 FDFAs with duo-normalized acceptance condition

As already mentioned, none of the common ω-automata has a unique minimal automaton, and the number of states in the minimal automaton may be bigger than the number of equivalence classes in $\sim_L$. For example, $L_2 = (\Sigma^* abc)^\omega$ has one equivalence class under $\sim_{L_2}$, since for any finite word x, an infinite extension xw for $w \in \Sigma^\omega$ is in the language iff $w \in L_2$.

The quest for finding a correspondence between equivalence classes of the language and an automaton model led to the development of *Families of Right Congruences* (FORCs) [19] and *Families of DFAs* (FDFAs) [2]. These definitions consider only ultimately periodic words (ie words of the form uv^ω) building on the well-known result that two ω-regular languages are equivalent if and only if they agree on the set of ultimately periodic words [6, 7]. We also consider only such words, and represent them as pairs (u,v) for $u \in \Sigma^*$ and $v \in \Sigma^+$.

Several canonical FDFAs were introduced over the years, the periodic FDFA [7], the syntactic FDFA [19], the recurrent FDFA [2], and the limit FDFA [17]. We do not go into the details of their definition but summarize the succinctness relations among them. It was shown in [2] that the syntactic and recurrent FDFAs can be exponentially more succinct than the periodic FDFA, while the translations in the other direction are at most polynomial. Further, the recurrent FDFA is never bigger, and can be quadratically more succinct, than the syntactic FDFA [2]. Limit FDFAs are the duals of recurrent FDFAs and similarly can also be at most quadratically bigger than the syntactic; and there are examples of quadratic blowups in the transformation from the recurrent to the limit and vice versa [17].

The gain in succinctness in going from the syntactic to the recurrent (or limit) FDFAs originates from removing the requirement that $x \approx^u y$ implies that $ux \sim uy$, which comes from the definition of a FORC.[1] The gain in succinctness of the syntactic/recurrent/limit FDFAs compared to the periodic FDFA is due to the use of a different type of acceptance condition.

An FDFA is a pair $\mathcal{F} = (\mathcal{Q}, \{\mathcal{P}_q\}_{q\in Q})$ consisting of a *leading* automaton structure $\mathcal{Q}$ and of a *progress* DFA $\mathcal{P}_q$ for each state q of $\mathcal{Q}$. There are a few ways to define acceptance on FDFAs. They differ on what decompositions (u,v) of an infinite word w are considered. We

[1] A FORC is a pair $\mathcal{F} = (\sim, \{\approx^u\})$ where $\sim$ is a right congruence, $\approx^u$ is a right congruence for every equivalence class u of $\sim$, and it satisfies that $x \approx^u y$ implies $ux \sim uy$. An ω-language L is recognized by $\mathcal{F}$ if it can be written as a union of sets of the form $[u]([v]_u)^\omega$ s.t. $uv \sim_L u$. Every FORC corresponds to an FDFA, but the other direction many not hold. This is since there is no requirement on the relation between the progress DFAs and the leading automaton in an FDFA, while there is in a FORC.

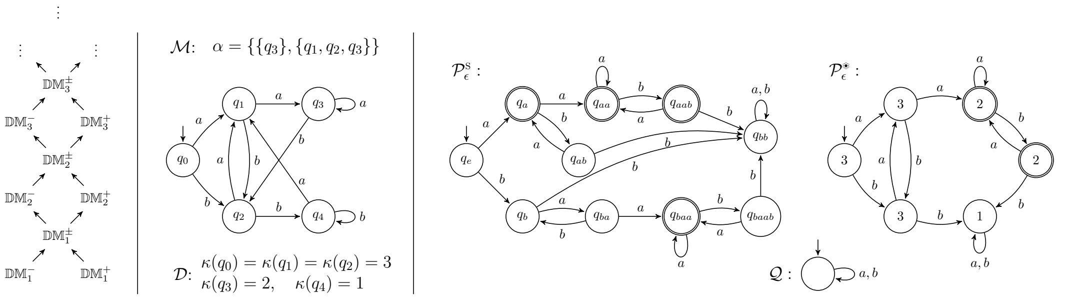


Figure 3.1 Left: The Wagner hierarchy. **Middle:** A DMA $\mathcal{M}$ and a DPA $\mathcal{D}$ for the language $L_{\infty aa \wedge \neg \infty bb}$. **Right:** Two FDFAs $\mathcal{F}^{\mathrm{S}} = (\mathcal{Q}, \{\mathcal{P}_\epsilon^{\mathrm{S}}\})$ and $\mathcal{F}^{\circledast} = (\mathcal{Q}, \{\mathcal{P}_\epsilon^{\circledast}\})$ for the language $L_{\infty aa \wedge \neg \infty bb}$ where $\mathcal{Q}$ is a one-state leading automaton. $\mathcal{F}^{\mathrm{S}}$ uses normalized acceptance, $\mathcal{F}^{\circledast}$ uses duo-normalized acceptance.

provide a definition for such decompositions in Def. 3.1. Once an α-decomposition is defined, an ω-word w is accepted by an FDFA using α-acceptance if there exists an α-decomposition (u, v) of w which is accepted. That is, the word v is accepted by $\mathcal{P}_{\mathcal{Q}(u)}$ where $\mathcal{P}_{\mathcal{Q}(u)}$ is the progress DFA corresponding to the state $\mathcal{Q}(u)$ reached by the leading automaton after reading u. We henceforth use $\mathcal{P}_u$ for $\mathcal{P}_{\mathcal{Q}(u)}$.

In exact acceptance, that is used in the periodic FDFA, any decomposition of the ω-word into an ultimately periodic word is considered. In normalized acceptance, used by the other three canonical FDFAs, only decompositions (u, v) in which the periodic part v loops in the leading automaton (ie $\mathcal{Q}(u) = \mathcal{Q}(uv)$) are considered.

As shown in [2] this acceptance condition, termed *normalization*, can yield an exponential save in the number of states. The intuition is that some periods are easier to verify as good periods if one considers some repetitions of them. For instance, in the language $(121 + 212)^\omega$ it is harder to figure out that $(\epsilon, 12)$ should be accepted than it is for $(\epsilon, 121{\cdot}212)$ though both represent the same ω-word.

For similar reasons, one may wonder if considering only decompositions that also close a loop in the progress automaton might lead to an exponential save as well. In the following we define FDFAs with such an acceptance condition, which we term *duo-normalization*. The notion of duo-normalization has appeared in the literature before. In particular, it resembles the notion of a linked-pair in ω-semigroups and Wilke-algebras [27, 9], it is used in [1, Proof of Thm. 5.8] and it is termed *idempotent* in [5].

▶ **Definition 3.1** (ω-words decomposition wrt an FDFA). *Let $u \in \Sigma^*$, $v \in \Sigma^+$ and $w \in \Sigma^\omega$. Let $\mathcal{F} = (\mathcal{Q}, \{\mathcal{P}_q\}_{q \in Q})$ be an FDFA.*

- *(u, v) is a decomposition of w if $uv^\omega = w$.*
- *A decomposition (u, v) is* normalized *if $\mathcal{Q}(u) = \mathcal{Q}(uv)$.*
- *A normalized decomposition is* duo-normalized *if $\mathcal{P}_u(v) = \mathcal{P}_u(vv)$.*

▶ **Definition 3.2** (Exact, Normalized, and Duo-Normalized acceptance). *Let $\mathcal{F} = (\mathcal{Q}, \{\mathcal{P}_q\}_{q \in Q})$ be an FDFA, $u \in \Sigma^*$, $v \in \Sigma^+$. We define three types of acceptance conditions: We say that $(u, v) \in [\![\mathcal{F}]\!]$ using* exact acceptance *if $v \in [\![\mathcal{P}_u]\!]$. We say that $(u, v) \in [\![\mathcal{F}]\!]$ using* normalized *(resp. duo-normalized) acceptance if there exists a normalized (resp. duo-normalized) decomposition (x, y) of uv^ω such that $y \in [\![\mathcal{P}_x]\!]$.*

An FDFA $\mathcal{F}$ is said to be α-*saturated* if for every ultimately periodic word w, all its α-decompositions agree on membership in $\mathcal{F}$. Assuming saturation, and an efficient α-normalization process (as suggested by Claim 3.3) we can alternatively define α-acceptance as in [2] using the efficient procedure that given any (u, v) returns a particular (x, y) that is α-normalized and satisfies $uv^\omega = xy^\omega$. Henceforth, all FDFAs are presumed to be saturated.

▷ Claim 3.3. Let $x \in \Sigma^*$ and $y \in \Sigma^+$. The word xy^ω has an α-decomposition of the form (xy^i, y^j) for i and j quadratic in the size of $\mathcal{F}$ for all $\alpha \in \{\text{exact, normalized, duo-normalized}\}$.

Clearly, if $\mathcal{F}$ is saturated using exact acceptance and it recognizes L then it is also saturated and recognizes L when using normalized acceptance instead. The same is true when going from normalized acceptance to duo-normalized acceptance.

▶ **Corollary 3.4.** *Duo-normalized FDFAs recognize all ω-regular languages.*

▶ **Example 3.5.** Fig. 4.1 (left) shows two FDFAs. The FDFA $\mathcal{F}_1 = (\mathcal{Q}, \{\mathcal{P}_\epsilon, \mathcal{P}_b\})$ has a leading automaton with two states $[\epsilon]$ and $[b]$, and the corresponding progress automata are $\mathcal{P}_\epsilon$ and $\mathcal{P}_b$. Consider the ultimately periodic word a^ω; since (ϵ, a) is a normalized decomposition of a^ω and $a \in [\![\mathcal{P}_\epsilon]\!]$, the word a^ω is accepted by $\mathcal{F}_1$ using normalized acceptance. The FDFA $\mathcal{F}_2 = (\mathcal{Q}, \{\mathcal{P}'_\epsilon, \mathcal{P}_b\})$ uses duo-normalization and $\mathcal{P}'_\epsilon$ as the progress DFA for $[\epsilon]$. The pair (ϵ, aa) is a duo-normalized decomposition of a^ω wrt $\mathcal{F}_2$ and it holds that $a \in [\![\mathcal{P}'_\epsilon]\!]$ thus the word a^ω is accepted by $\mathcal{F}_2$. Observe that the normalized (rather than duo-normalized) decomposition (ϵ, a) is not accepted by $\mathcal{P}'_\epsilon$. In this example the FDFA using duo-normalization has more states. Later on we provide an example where an FDFA using duo-normalization has fewer states, and even exponentially fewer.

▶ **Theorem 3.6.** *The following holds for saturated FDFAs using duo-normalized acceptance: complementation can be computed in constant space; intersection, union and membership can be computed in logarithmic space; emptiness, universality, containment and equivalence can be computed in non-deterministic logarithmic space.*

From now on, unless stated otherwise, we work with duo-normalization. That is, when we say $(u, v) \in [\![\mathcal{F}]\!]$ or $w \in [\![\mathcal{F}]\!]$ we mean according to duo-normalized acceptance condition.

As we later show duo-normalized FDFAs can be exponentially more succinct than all previously defined FDFAs. This is essentially because it considers fewer or more specific decompositions. One might wonder if considering even more specific decompositions will lead to more succinct FDFAs, and can this still be done in the same complexity as for normalized and duo-normalized FDFAs. We come back to this point in the next section, see Prop. 4.3.

4 The Diameter Measure — A Robust Measure on FDFAs

In the following section we define a measure on FDFAs that is tightly related to the inclusion measure of the Wagner hierarchy. The defined measure is robust among FDFAs in the same way that the inclusion measure is robust among DMAs. That is, every pair of FDFAs $\mathcal{F}_1$ and $\mathcal{F}_2$ recognizing the same language agree on this measure.

To devise this measure we would like to understand what the inclusion measure entails on an FDFA for the language. If a DMA has an inclusion chain $S_1 \subset S_2 \subset S_3$ then there is a state q_u in S_1 reachable by some word u from which there are words v_1, v_2, v_3 looping on q_u while traversing the states of S_1, S_2 and S_3, respectively (all and only these states). We term this state a *pivot* state. Assume S_1 is rejecting, then $u(v_1)^\omega \notin L$, $u(v_2)^\omega \in L$ and

$u(v_3)^\omega \notin L$. Since they all loop back to q_u, we have that $u(v_1v_2)^\omega$ also loops in S_2 and is thus accepted and $u(v_1v_2v_3)^\omega$ also loops in S_3 and is thus rejected. Since $v_1 \prec v_1v_2 \prec v_1v_2v_3$ (where $\prec$ denotes the prefix relation) tracing the run on $v_1v_2v_3$ in a progress DFA for u we expect the state reached after v_1 to be rejecting, the one after v_1v_2 to be accepting and the one after $v_1v_2v_3$ to be rejecting. To be precise, we should expect this only if the words v_1, v_1v_2 and $v_1v_2v_3$ are α-normalized, where α is the normalization used by the FDFA.

Let's inspect this on our running example $L_{\infty aa\wedge\neg\infty bb}$ with the DMA in Fig. 3.1 (middle) and the inclusion chain $S_1 = \{q_1, q_2\}$, $S_2 = \{q_1, q_2, q_3\}$, and $S_3 = \{q_1, q_2, q_3, q_4\}$. We can choose q_1 for the pivot state q_u of S_1 and the words $v_1 = ba$, $v_2 = aba$ and $v_3 = abba$, that loop respectively in S_1, S_2 and S_3. Fig. 3.1 (right) provides two FDFAs for this language. The FDFA $\mathcal{F}^{\text{s}}$ uses normalization and $\mathcal{F}^{\circledast}$ uses duo-normalization. Looking at $\mathcal{P}^{\text{s}}_\epsilon$, we see that v_1, v_1v_2 and $v_1v_2v_3$ are normalized and the states reached after reading them (q_{ba}, q_{baa}, q_{bb}) are rejecting or accepting as expected. The same is true for $\mathcal{P}^{\circledast}_\epsilon$.

Should we entail from this discussion and example that the maximal number of alternations between rejecting and accepting states along any path in an FDFA for a language in $\mathbb{DM}^-_k$ is at most $k-1$? This is true in the progress DFA $\mathcal{P}^{\circledast}_\epsilon$, but the progress DFA $\mathcal{P}^{\text{s}}_\epsilon$ clearly refutes it, since it has strongly connected accepting and rejecting states (eg, q_a and q_{ab}) and so we can create paths with an unbounded number of alternations between them.

Take such a path with say $k+1$ prefixes $z_1 \prec z_2 \prec \ldots \prec z_{k+1}$ alternating between accepting and rejecting states. Are the words z_i normalized? They can be. Take for instance $z_1 = a$, $z_2 = ab$, and so on ($z_{k+1} = z_k \cdot b$ if k is even and $z_{k+1} = z_k \cdot a$ otherwise).

Can they all be duo-normalized? They can be as we show in Fig. 4.1 (right). It shows a progress DFA $\mathcal{P}_\epsilon$ for an FDFA using duo-normalization and a one-state leading automaton. The FDFA recognizes the language ∞aa. The words $a \prec ab \prec abaa$ are all duo-normalized and reach alternating accepting/rejecting states though the language is in $\mathbb{DM}^-_2$. To fix this issue we introduce the notion of a *persistent decomposition.*

▶ **Definition 4.1** (persistent decomposition wrt an FDFA). *Let $u \in \Sigma^*$, $v \in \Sigma^+$. Let $\mathcal{F} = (\mathcal{Q}, \{\mathcal{P}_q\}_{q\in Q})$ be an FDFA. A duo-normalized decomposition (u, v) is* persistent *if for every $z \in \Sigma^*$ there exists $i > 1$ such that $\mathcal{P}_u(zv) = \mathcal{P}_u(zv^i)$.*

As in Claim 3.3 there exist i and j quadratic in the size of $\mathcal{F}$ such that (uv^i, v^j) is persistent.

▷ Claim 4.2. For every $x\in\Sigma^*$ and $y\in\Sigma^+$ the word xy^ω has a persistent decomposition of the form (xy^i, y^j) where i and j are of size quadratic in $\mathcal{F}$.

One might try to define an additional acceptance condition using the persistent decomposition as was done above, but as stated by the following claim this is futile.

▶ **Proposition 4.3.** *Every FDFA defined with persistent acceptance recognizes the same language when defined with duo-normalized acceptance instead.*

Back to the issue of finding the position on the Wanger hierarchy, we can look for chains of prefixes with alternating acceptance such that each prefix in the chain is persistent.

▶ **Definition 4.4** (Persistent Chain). *Let $\mathcal{F}$ be an FDFA and $u \in \Sigma^*$. We say that $v_1 \prec v_2 \prec \ldots \prec v_k$ for $v_i \in \Sigma^*$ is a u-*persistent chain *of length k wrt $\mathcal{F}$ if (u, v_i) is persistent for every $1 \leq i \leq k$ and $(u, v_{i+1}) \in [\![\mathcal{F}]\!]$ iff $(u, v_i) \notin [\![\mathcal{F}]\!]$ for every $1 \leq i < k$. We say the chain is* positive *(resp.* negative*) if $(u, v_1) \in [\![\mathcal{F}]\!]$ (resp. $(u, v_1) \notin [\![\mathcal{F}]\!]$). We use simply* persistent chain *when u is clear from the context.*

We can now state the measure on FDFAs that relates them to the Wagner hierarchy.

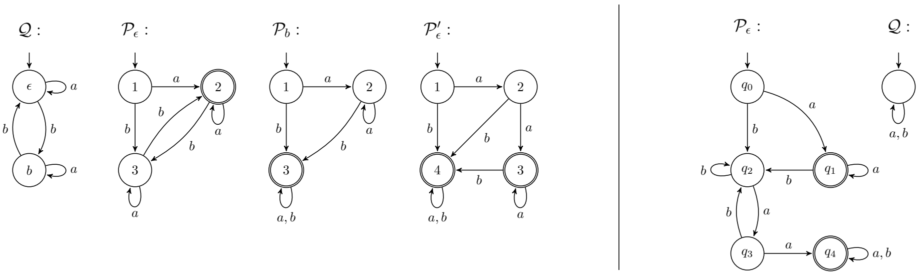


Figure 4.1 Left: Two FDFAs $\mathcal{F}_1 = (\mathcal{Q}, \{\mathcal{P}_\epsilon, \mathcal{P}_b\})$ and $\mathcal{F}_2 = (\mathcal{Q}, \{\mathcal{P}'_\epsilon, \mathcal{P}_b\})$ for the language $(\Sigma^* b)^\omega \cup (bb)^* a^\omega$ using normalized and duo-normalized acceptances, respectively. **Right:** The progress DFA $\mathcal{P}_\epsilon$ for an FDFA accepting ∞aa that uses duo-normalization and a one-state leading automaton.

▶ **Definition 4.5** (The Diameter Measure). *Let $\mathcal{F}$ be an FDFA and $u \in \Sigma^*$. We define the* positive *(resp.* negative*)* diameter measure *of the progress DFA $\mathcal{P}_u$, denoted $|\mathcal{P}_u|^+_\rightsquigarrow$ (resp. $|\mathcal{P}_u|^-_\rightsquigarrow$), as the maximal k for which there exists a* positive *(resp. negative) persistent chain of length k in $\mathcal{P}_u$. We define $|\mathcal{F}|^+_\rightsquigarrow$ as* $\max\{|\mathcal{P}_q|^+_\rightsquigarrow : q \in Q\}$ *and $|\mathcal{F}|^-_\rightsquigarrow$ as* $\max\{|\mathcal{P}_q|^-_\rightsquigarrow : q \in Q\}$.

We show that the diameter measure is robust among all FDFAs for the language by relating it to the Wagner hierarchy as formally stated below.

▶ **Theorem 4.6** (Correlation to Wagner's hierarchy). *Let $\mathcal{F}$ be an FDFA using any of the acceptance types $\alpha \in \{$exact, normalized, duo-normalized$\}$.*

- $[\![\mathcal{F}]\!] \in \mathbb{DM}^\pm_k$ *iff* $|\mathcal{F}|^+_\rightsquigarrow \leq k$ *and* $|\mathcal{F}|^-_\rightsquigarrow \leq k$
- $[\![\mathcal{F}]\!] \in \mathbb{DM}^+_k$ *iff* $|\mathcal{F}|^+_\rightsquigarrow \leq k$ *and* $|\mathcal{F}|^-_\rightsquigarrow < k$
- $[\![\mathcal{F}]\!] \in \mathbb{DM}^-_k$ *iff* $|\mathcal{F}|^-_\rightsquigarrow < k$ *and* $|\mathcal{F}|^-_\rightsquigarrow \leq k$

In the proof of Thm. 4.6, given a persistent chain $v_1 \prec v_2 \prec \ldots \prec v_k$ in $\mathcal{P}_u$ in some FDFA, we would like to find an inclusion chain of length k in some DMA recognizing the same language. The SCCs visited infinitely often by the words $u(v_1)^\omega, \ldots, u(v_k)^\omega$ might not correspond to an inclusion chain in the DMA. Roughly speaking, to obtain a persistent chain for which this does hold, we make sure every element of the chain has already reached its final SCC and traversed it. Using the following lemma we can create such a persistent chain.

▶ **Lemma 4.7** (Pumping Persistent Periods). *Let $\mathcal{A}$ be an automaton and let $v \in \Sigma^+$ be $\mathcal{A}$-persistent.*[2] *For every $n \in \mathbb{N}$ there exists $l \geq n$ such that for every extension $z \in \Sigma^*$, if vz is $\mathcal{A}$-persistent then $v^l z$ is also $\mathcal{A}$-persistent.*

Following [10] a word $v \in \Sigma^*$ is said to be a *suffix-invariant* of $u \in \Sigma^*$ (in short *u-invariant*) with respect to L if $u \sim_L uv$. That is, no suffix distinguishes between u and the word obtained by concatenating v to u.

Proof of Thm. 4.6. We prove the claim regarding the positive measure. The claim regarding the negative measure is proven symmetrically. We show that

1. $|L|^+_\subset \geq k$ implies $|\mathcal{F}|^+_\rightsquigarrow \geq k$ and
2. $|\mathcal{F}|^+_\rightsquigarrow \geq k$ implies $|L|^+_\subset \geq k$.

The two claims together entail that $|\mathcal{F}|^+_\rightsquigarrow = |L|^+_\subset$.

[2] We say that v is $\mathcal{A}$-persistent if $\mathcal{A}(v) = \mathcal{A}(vv)$ and for every $z \in \Sigma^*$ there exists an $i > 1$ such that $\mathcal{A}(zv) = \mathcal{A}(zv^i)$.

1. We start by showing that if $|L|^{+}_{\subset} \geq k$ then $|\mathcal{F}|^{+}_{\rightsquigarrow} \geq k$. Let $\mathcal{M}$ be a DMA for L. From $|L|^{+}_{\subset} \geq k$ we know that there exists an MSCC of $\mathcal{M}$ subsuming SCCs $S_1, S_2, \ldots, S_k$ such that $S_1 \subsetneq S_2 \subsetneq \ldots \subsetneq S_k$ and S_i is an accepting component if and only if i is odd. Pick a state s in S_1. For $1 \leq i \leq k$ let v_i be a word that loops on s while traversing all the states of S_i and no other states. Let u be a word reaching s from the initial state. Consider the progress DFA of u, $\mathcal{P}_u$. By Claim 4.2 there exists l_1 such that $y_1 = (v_1)^{l_1}$ is $\mathcal{P}_u$-persistent. Similarly, there exists l_2 such that $y_2 = (y_1 v_2)^{l_2}$ is $\mathcal{P}_u$-persistent. In the same way we define $y_i = (y_{i-1} v_i)^{l_i}$ for all $i \in [2..k]$. Consider the words $w_i = u(y_i)^{\omega}$ for $i \in [1..k]$. Since the set of states visited infinitely often when reading w_i is exactly S_i, it follows that w_i is in L if and only if i is odd. Since all the infixes y_i loop back to s it follows that all the y_i's are u-invariants and thus the (u, y_i) are persistent in $\mathcal{F}$. Since $y_1 \prec y_2 \prec \ldots \prec y_k$ we have found a positive alternating persistent chain in $\mathcal{P}_u$ of length k. Hence, $|\mathcal{P}_u|^{+}_{\rightsquigarrow} \geq k$, which in turn gives that $|\mathcal{F}|^{+}_{\rightsquigarrow} \geq k$.
2. Next we show that if $|\mathcal{F}|^{+}_{\rightsquigarrow} \geq k$ then $|L|^{+}_{\subset} \geq k$. Let u be such that $|\mathcal{P}_u|^{+}_{\rightsquigarrow} \geq k$. Then there exists a persistent chain $v_1 \prec v_1 v_2 \prec \ldots \prec v_1 v_2 \cdots v_k$ of length k in $\mathcal{P}_u$, starting with an accepting state. For $i \in [1..k]$ let q_i be the state reached after reading $v_1 v_2 \cdots v_i$. Note that q_i is accepting iff i is odd.

   Let $\mathcal{M}$ be a DMA for L and let n be its number of states. Let $l_1 \geq n$ be the number promised by Lemma 4.7 for v_1. Consider $(v_1)^{l_1} v_2$. As v_1 and $v_1 v_2$ are both $\mathcal{P}_u$-persistent, it follows from the lemma that $(v_1)^{l_1} v_2$ is $\mathcal{P}_u$-persistent as well. Since v_1 is $\mathcal{P}_u$-persistent it loops on q_1 and it holds that $(v_1)^{l_1} v_2$ reaches and loops on q_2. Continuing in the same manner, let $y_1 = v_1$ and $y_i = (y_{i-1})^{l_{i-1}} \cdot v_i$ for $i \in [2..k]$ where $l_{i-1} \geq n$ is the number from Lemma 4.7 for y_{i-1}. Then y_i is $\mathcal{P}_u$-persistent, reaching and looping on q_i. Moreover $u(v_1 v_2 \cdots v_i)^{\omega} \in L$ iff $u(y_i)^{\omega} \in L$ iff $x(y_i)^{\omega} \in L$ for any $x \sim_L u$. Let $x_i = u \cdot y_k^n \cdot y_{k-1}^n \cdots y_i^n$ for $i \in [1..k]$. As the v_i's are u-invariant it holds that the x_i's are $\sim_L u$. For $i \in [1..k]$ let $w_i = x_i (y_i)^{\omega}$. Thus, w_i is in L iff i is odd. For every such i, consider the run of $\mathcal{M}$ on w_i, and let $\mathit{inf}(w_i) = S_i$ be the states of the SCC that $\mathcal{M}$ eventually traverses in. Since n bounds the number of states of $\mathcal{M}$ it follows that after reading x_i the automaton already traversed the SCC S_i and reading y_i again, it will stay in S_i. Because $x_{i-1} = x_i \cdot y_{i-1}^n$ and $y_{i-1}^n \prec y_i$ it holds that $S_1 \subseteq S_2 \subseteq \ldots \subseteq S_k$.

   From the acceptance of the words w_i we conclude S_i is accepting iff i is odd. Therefore the inclusions are strict, namely $S_1 \subsetneq S_2 \subsetneq \ldots \subsetneq S_k$. This proves $|L|^{+}_{\subset} \geq k$. ◀

With Thm. 4.6 in place we conclude the robustness of the diameter measure.

▶ **Corollary 4.8.** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two FDFAs recognizing the same language. Then $|\mathcal{F}_1|^{+}_{\rightsquigarrow} = |\mathcal{F}_2|^{+}_{\rightsquigarrow}$ and $|\mathcal{F}_1|^{-}_{\rightsquigarrow} = |\mathcal{F}_2|^{-}_{\rightsquigarrow}$.*

Next we show that given an FDFA we can compute its diameter measure in polynomial space. The proof shows that with each progress DFA $\mathcal{P}_u$ we can associate a DFA $\mathbb{P}_u$, which we term the *persistent DFA*. Broadly speaking, $\mathbb{P}_u$ is the product of the leading automaton and copies of the progress automaton, starting from each of its states. The states of $\mathbb{P}_u$ can be classified into *significant* and *insignificant* where significant states are those that are reached by persistent words and only such words. The classification can easily be done by inspecting the "state vector". The persistent DFA need not be built, instead a persistent chain can be non-deterministically guessed and verified in polynomial space.

▶ **Theorem 4.9.** *The diameter measure of a duo-normalized FDFA can be computed in PSPACE.*

Since normalized and exact FDFAs are also duo-normalized, this upper bound holds for them as well. However, in Prop. 4.12 we show that for normalized (and exact) FDFAs the diameter measure can be computed in NLOGSPACE. For the case of duo-normalized FDFAs, we provide a matching lower bound.

▶ **Theorem 4.10.** *The problem of determining whether the diameter measure of a duo-normalized FDFA is at least k is PSPACE-hard.*

Proof sketch. The proof uses a reduction from non-emptiness of intersection of DFAs, which is known to be PSPACE-hard [15]. Let $D_1, D_2, \ldots, D_k$ be k DFAs over Σ. We construct an FDFA with a one-state leading automaton and a progress DFA $\mathcal{P}$ over $\Sigma' = \Sigma \cup \{1, \ldots, k\} \cup \{\sharp\}$ as depicted in Fig. 5.1 (left), see the full version for a complete description. To see that the FDFA is saturated, we show in the full version that not only every two duo-normalized decompositions (u, v) and (u', v') of the same ultimately periodic word w agree on acceptance, but their periods also traverse the same MSCC in $\mathcal{P}$.

We claim that if there is a word $v \in \Sigma$ in the intersection of all D_i's, then $\sharp v \prec \sharp v1 \prec \sharp v12 \prec \sharp v123 \prec \ldots \prec \sharp v12 \cdots k \prec \sharp v12 \cdots k\sharp\sharp$ is a persistent chain in $\mathcal{P}$ of length $k+2$. Let $y_0 = \sharp v$, $y_i = \sharp v12 \cdots i$ for $i \in [1..k]$, and $y_{k+1} = y_k\sharp\sharp$. Then y_i reaches s_i^i, and reading y_i from s_i^i reaches s_i^i again. Thus y_i is duo-normalized and y_i is accepted iff i is even. To see that y_i is persistent it remains to show that from any state q reading y_i and reading $(y_i)^2$ the automaton reaches the same state q' for some q'. Observe that reading y_i from any state s_j^j for $j \leq i$ will still reach s_i^i. If y_i is read from s_j^j for $i < j \leq k+1$ or any other state (as there are no other outgoing $\sharp$-transitions) it will reach s_{k+1}^{k+1} and stay there forever. Thus y_i is persistent for any i. Hence, $y_0 \prec y_1 \prec y_2 \prec \ldots \prec y_k \prec y_{k+1}$ is a persistent chain in $\mathcal{P}$ of length $k+2$.

For the other direction, we claim that if there exists a persistent chain of length $k+2$ in $\mathcal{P}$, then the intersection of all D_i's is non-empty. Let $w_0 \prec w_2 \prec \ldots \prec w_{k+1}$ be such a chain. First, we note that from the structure of $\mathcal{P}$, since the MSCCs corresponding to the D_i's are of alternating acceptance, it follows that w_0 reaches s_0^0, w_{k+1} reaches s_{k+1}^{k+1} and all other w_i reach some state in the i-th MSCC. Second, if reading w_i reaches a state in the i-th MSCC, then since w_i is duo-normalized, reading w_i for the second time must loop back to the same state. For $i \in [1..k]$ this can only occur if w_i is a rotation of $v_1 12 \cdots i\sharp v_2 12 \cdots i\sharp \cdots v_m 12 \cdots i\sharp$ for some $v_1, \ldots, v_m \in D_i$. This is since w_i must contain the letter 1 and it can only be read from final states of D_i. Note that there can't be an infix of any w_i for $1 \leq i \leq k$ where the letter $\sharp$ appears after a letter lower than i, as this would lead from the i-th MSCC to s_{k+1}^{k+1}. As the w_i's form a chain, w_1 is a prefix of all the w_i's for $1 < i \leq k$. It follows that w_1 is exactly of the form $\sharp v1$, and as v is common to all the following w_i's, v is in the intersection of all the D_i's. ◀

The following proposition shows that there exists FDFAs where the smallest elements of a persistent chain are inevitably of exponential length. The idea is to take the construction of the FDFA in the proof of Thm. 4.10 and let the DFA D_i count modulo the i-th prime.

▶ **Proposition 4.11.** *There exists a family of languages $\{L_n\}$ with an FDFA with number of states polynomial in n where any persistent chain of maximal length must include a word of length exponential in n.*

Computing the Wagner position on normalized (rather than duo-normalized) FDFAs, can be done more efficiently, specifically in NLOGSPACE.

▶ **Proposition 4.12.** *The position in the Wagner hierarchy of an FDFA using normalized acceptance can be computed in NLOGSPACE.*

For the sake of the proof we define two sub-classes of normalized FDFAs. The smallest one is a generalization studied in [5] of the syntactic FDFA for a language L [19].

▶ **Definition 4.13** (∼-syntactic FDFA). *Let $L \subseteq \Sigma^\omega$ and let $\sim$ be a right congruence refining the syntactic right congruence $\sim_L$. The* ∼-syntactic FDFA*, denoted $(\mathcal{Q}^\sim, \mathcal{P}_u^\sim)$, is defined as follows. The leading automaton $\mathcal{Q}^\sim$ is $\mathcal{A}[\sim]$, and the progress automaton $\mathcal{P}_u^\sim$ is $\mathcal{A}[\approx_u]$ where $x \approx_u y$ if (a) $ux \sim uy$ and (b) for every $z \in \Sigma^*$ it holds that $uxz \sim u$ implies $u(xz)^\omega \in L$ iff $u(yz)^\omega \in L$.*[3]

It is shown in [5, Lemma 21] that if x is duo-normalized wrt $\sim$ and $\approx_u$ then for every $y \approx_u x$ we have that y is also duo-normalized. We can thus refer to a state as being duo-normalized. It is then showed that two duo-normalized states in the same SCC of a ∼-syntactic progress DFA agree on acceptance [5, Lemma 23]. These properties allow defining a polynomial procedure that associates with every state of a *∼-syntactic FDFA* a color that tightly correlates to position on the Wagner hierarchy [5].

It follows that on *∼-syntactic FDFAs* the Wagner position can be determined in polynomial time. The proof can be generalized to any FDFA using normalization in which the right congruence $x \approx_u y$ implies $x \sim_L y$. We call such FDFAs *projective FDFAs*.

▶ **Definition 4.14.** *An FDFA $(\mathcal{A}[\sim], \{\mathcal{A}[\approx_u]\})$ is termed* projective *if for every progress DFA, the respective right congruence $\approx_u$ satisfies that $x \approx_u y$ implies $x \sim_L y$.*

Fig. 5.1 (left) depicts the inclusions among these classes. (In the meantime ignore the text in blue, and the arrow; we will come back to this in the next section.) Since any FDFA using normalization can be transformed with a quadratic blowup into a *projective FDFA* (by multiplying the progress DFAs by the leading automaton) we have that the Wagner position on normalized FDFAs can be computed in polynomial time. In the full version of the paper we show that it can also be computed in NLOGSPACE.

5 Succinctness Results

We turn to provide some succinctness results regarding FDFAs with duo-normalized acceptance. The results compare duo-normalized FDFAs with normalized FDFAs, as well as canonical FDFAs using these acceptance conditions. We already mentioned four canonical FDFAs: the periodic FDFA [7] that uses exact acceptance; and the syntactic [19], recurrent [2] and limit FDFAs [17] that use normalized acceptance. A canonical FDFA that uses duo-normalization can be extracted from notions of [5]. We term it the *Colorful FDFA*, since it relies on the notion of natural colors [10].

Loosely speaking, [10] shows that given an ω-regular language L one can associate with every word $w \in \Sigma^\omega$ a natural color. If w is given color k wrt L, then there is no parity automaton for L that would visit a color lower than k infinitely often when reading w. Consider again the language $L_{\infty aa \wedge \neg\infty bb}$ requiring infinitely many aa and finitely many bb for which a DPA is given in Fig. 3.1 (middle). The colors of $(ab)^\omega$, $(a)^\omega$, $(aab)^\omega$, $(aabb)^\omega$ and $(b)^\omega$ are 3, 2, 2, 1 and 1, resp. The intuition is that the color is 1 if bb occurs infinitely often, it is 2 if aa occurs infinitely often but bb occurs only finitely often, and it is 3 if neither aa nor bb occur infinitely often. We use $\mathsf{Color}(w)$ for the natural color of w. Note that the language L is implicit in the notation.

A related definition provided in [5, Def 2.] can be viewed as giving colors for finite words v wrt to an ω-regular language L and an equivalence class $[u]$. We use $\mathsf{Color}_u(v)$ to denote the color of the finite word $v \in \Sigma^+$ wrt a finite word $u \in \Sigma^*$. The definition satisfies that

[3] *Canonical FORC* is the term used in [5] for the FORC underlying the ∼-syntactic FDFA.

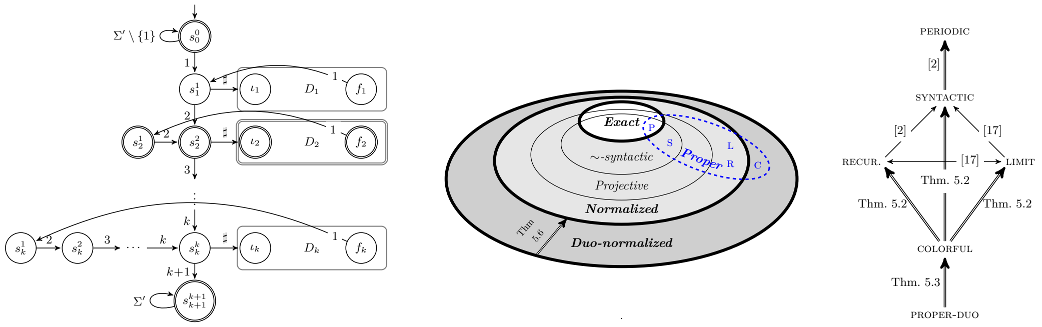


Figure 5.1 Left: The progress DFA $\mathcal{P}$ used in the proof of Thm. 4.10. Its alphabet is $\Sigma' = \Sigma \cup \{1, \ldots, k, k+1\} \cup \{\sharp\}$ where Σ is the alphabet of the DFAs $D_1, \ldots, D_k$. Transitions to the sink state s_{k+1}^{k+1} are omitted. The figure assumes k is odd; for $j \in [1..k]$, ι_j is initial and f_j is accepting. **Middle:** The inclusions among these classes of FDFAs (in black), as well as the placement of the canonical FDFAs in these classes (in blue). The letters P,S,R,L,C abbreviate PERIODIC, SYNTACTIC, RECURRENT, LIMIT, and COLORFUL, resp. **Right:** Picture summarizing succinctness results on proper FDFAs. A double-line (resp. one-line) arrow from C to D indicates that C can be exponentially (resp. quadratically) more succinct than D.

$\mathsf{Color}_u(v)$ returns $\max\{\mathsf{Color}(u(vz)^\omega) \mid z \in \Sigma^*\}$. Note that it is possible that $u(v')^\omega = u(v'')^\omega$ for some $u \in \Sigma^*$ and $v', v'' \in \Sigma^+$, though $\mathsf{Color}_u(v') \neq \mathsf{Color}_u(v'')$. Indeed in the example of $L_{\infty aa \wedge \neg \infty bb}$ we have $(a)^\omega = (aa)^\omega$, yet $\mathsf{Color}_\epsilon(a) = 3$ while $\mathsf{Color}_\epsilon(aa) = 2$.

The reason is that if the period contains a followed by some z the resulting color may be 3 or 2 or 1 while if the period contains aa followed by some z the color can be 2 or 1 but it cannot be 3 since aa surely occurs infinitely often.

These colors can be used to define equivalence classes $\approx_u^{\circledast}$ for each word $u \in \Sigma^*$ by differentiating between words x and y if there is a word z such that the respective extensions xz and yz disagree on the color (wrt u).[4] The Colorful FDFA uses the automaton for $\sim_L$ for the leading automaton as the other canonical FDFAs do. For the progress DFA for u it takes a DFA whose automaton structure is derived by the equivalence relation $\approx_u^{\circledast}$, and the accepting states are those with an even color. The acceptance type of the Colorful FDFA is duo-normalization.

▶ **Definition 5.1** (The Colorful FDFA). *Let $u, x, y \in \Sigma^*$. We define $x \approx_u^{\circledast} y$ if for every $z \in \Sigma^*$ we have $\mathsf{Color}_u(xz) = \mathsf{Color}_u(yz)$. The* colorful FDFA *for a language L, denoted $\mathcal{F}^{\circledast}(L)$, uses duo-normalized acceptance and consists of $(\mathcal{A}[\sim_L], \{\mathcal{P}_u^{\circledast}\})$ where $\mathcal{P}_u^{\circledast}$ is a DFA with the automaton structure $\mathcal{A}[\approx_u^{\circledast}]$ where state q_v is accepting if $\mathsf{color}_u(v)$ is even.*

The Colorful FDFA $\mathcal{F}^{\circledast} = (\mathcal{Q}, \{\mathcal{P}_\epsilon^{\circledast}\})$ for our running example $L_{\infty aa \wedge \neg \infty bb}$ is given in Fig. 3.1 (right).

We are now ready to discuss the succinctness results. Recall that the canonical FDFAs using normalized acceptance have been shown to be exponentially more succinct than the canonical model using exact acceptance [2]. We first show that using duo-normalization a similar succinctness gain is achieved, namely that the Colorful FDFA can be exponentially more succinct than all other canonical representations.

[4] These equivalence classes correspond to the precise family of weak priority mappings wrt to $\sim_L$ from [5, Def. 17].

▶ **Theorem 5.2.** *The Colorful FDFA can be exponentially more succinct than the syntactic/recurrent/limit FDFA.*

The proof uses the family of languages $\{L_n\}_{n\in\mathbb{N}}$ over $\Sigma = \{a, b, \langle, \rangle\}$ defined as follows.

$$L_n = \left\{ w \in \{a,b,\langle,\rangle\}^\omega \;\middle|\; \begin{array}{l} w \text{ has inf. many occurrences of } \langle a^k b^m \rangle \text{ for some } k \in [1..n] \\ \text{and } m \text{ that is divisible by the } k\text{-th prime.} \end{array} \right\}$$

The idea is that using duo-normalization the Colorful FDFA can look only for prefixes of the form $\langle a^k b^m\rangle$, whereas the syntactic/recurrent/limit FDFA must also answer correctly for prefixes of the form $b^m\rangle\langle a^k$ due to using normalized acceptance. Recognizing prefixes of the latter form is much harder.

Next we show that the Colorful FDFA is not the most succinct among the duo-normalized FDFAs. The essence of the proof is that the Colorful FDFA must keep track of all infixes of interest seen in order to maintain the real color of the word. On the other hand, a duo-normalized FDFA, similarly to a DBA, can choose an arbitrary order to look for such infixes. The proof uses the family $\{L'_n\}_{n\in\mathbb{N}}$ over $\Sigma = \{a_1, \ldots, a_n\}$ defined as follows.[5]

$$L'_n = \{w \in \{a_1, \ldots, a_n\}^\omega \;\mid \text{for all } i \in [1..n] \text{ the letter } a_i \text{ appears inf. often in } w\}$$

▶ **Theorem 5.3.** *Duo-normalized FDFAs can be exponentially more succinct than the Colorful FDFA.*

An FDFA in general can use any leading automaton $\mathcal{A}[\sim]$ for a right congruence $\sim$ that refines $\sim_L$. We note that all the canonical models (the periodic, syntactic, recurrent, limit and colorful) use $\mathcal{A}[\sim_L]$ for the leading automaton, we term such FDFAs, *proper*.

▶ **Definition 5.4** (Proper FDFAs). *An FDFA recognizing a language L is termed* proper *if its leading automaton is $\mathcal{A}[\sim_L]$.*

The FDFA used in the proof of Thm. 5.3 is proper. We can thus strengthen the claim as follows.

▶ **Corollary 5.5.** *Proper duo-normalized FDFAs can be exponentially more succinct than the Colorful FDFA.*

Surprisingly, Klarlund has shown that non-proper normalized FDFAs may be exponentially more succinct than proper normalized FDFAs [14]. One may thus wonder if non-proper normalized FDFAs can be as succinct as duo-normalized FDFAs. That is, if duo-normalization adds succinctness when considering non-proper FDFAs. The following theorem shows that duo-normalization does add succinctness even relative to non-proper normalized FDFAs (and even when limiting duo-normalized FDFAs to proper ones).

▶ **Theorem 5.6.** *(Proper) duo-normalized FDFAs can be exponentially more succinct than (not necessarily proper) normalized FDFAs.*

The proof uses the following family of languages $\{L''_n\}_{n\in\mathbb{N}}$ over $\Sigma = \Sigma_a \cup \Sigma_s$ where $\Sigma_a = \{a_1, \ldots, a_n\}$ and $\Sigma_s = \{s_1, \ldots, s_n\}$.

$$L''_n = \left\{ w \in (\Sigma^*\Sigma_a)^\omega \;\middle|\; \begin{array}{l} \text{Let } m = \max\{j \mid a_j \in \Sigma_a \cap \mathit{inf}(w)\}. \\ \text{Then } s_m \in \Sigma_s \text{ appears inf. often in } w. \end{array} \right\}.$$

[5] This family is used in [4] to show DBAs can be exponentially more succinct than combinations of DFAs.

The challenge in the language can be observed in periods where s_m occurs before a_m was seen (for m being the maximal index of an a_i letter in the period). The duo-normalized FDFA has the privilege of looking for duo-normalized decompositions in which s_m is observed after the maximal a_m is seen. As the normalized FDFA has to consider all prefixes, specifically prefixes for every subset of Σ_s, it must grow to an exponential size.

6 Discussion

We have shown that FDFAs with duo-normalized acceptance can be exponentially more succinct than FDFAs using (standard) normalization. At the same time the common operations procedures and decision problems on them can still be done in NLOGSPACE. Fig. 5.1 (right) summarizes the results regarding succinctness among the canonical FDFAs suggested thus far. It shows that the Colorful FDFA can be exponentially more succinct than all other canonical models. At the same time, a minimal duo-normalized FDFA can be exponentially more succinct than the Colorful FDFA.

The figure might raise the question whether a duo-normalized FDFA can be doubly-exponentially more succinct than the periodic FDFA (the least succinct canonical representation). However this cannot be since a duo-normalized FDFA can be translated into a non-deterministic Büchi automaton (NBA) using exactly the same procedure as the one transforming a normalized FDFA into an NBA [1]. The reason is that the construction actually looks for a duo-normalized decomposition (which by saturation exists).

▶ **Proposition 6.1.** *If L has a duo-normalized FDFA $\mathcal{F}$ then it has an NBA of size polynomial in the number of states of $\mathcal{F}$.*

Since an NBA can be converted into a periodic FDFA in an exponential blowup [7, 16] we get an overall exponential translation from duo-normalized FDFAs to the periodic FDFA, showing no doubly-exponential lower bound can be achieved. Since NBAs can be converted to DPAs with an exponential blow up [24, 23, 25, 11] and DPAs can be polynomially converted into non-proper FDFAs [1] we can conclude the following.

▶ **Corollary 6.2.** *The periodic FDFA and the minimal (non-proper) normalized FDFAs and DPAs of a language L are at most exponentially larger than a duo-normalized FDFA for L.*

We also answer a question posed by [10] regarding the relation of the structure of an FDFA to its position in the Wagner hierarchy. Specifically, we have provided a measure on FDFAs that corresponds to the Wagner hierarchy. We have shown that its computation is PSPACE-complete for duo-normalized FDFAs, and is in NLOGSPACE for normalized FDFAs. The measure is based on the notion of a persistent chain. Since the Wagner hierarchy correlates to the minimal color required by a parity automaton, we can define a notion of chains that relates to natural colors of [10, 5], and is thus a semantic notion (defined wrt to the language regardless of a particular acceptor for it). In the full version we show that the existence of one type of chain implies the existence of the other type of chain, and vice versa.

The notion of duo-normalization decomposition seems to be related to the notion of a *linked-pair* in ω-semigroups and Wilke-algebras [27, 9], in which (s, e) is a linked pair if $se = s$ and e is idempotent. It seems that there are two main differences; the first is that a linked pair relates to one relation $\sim$, while a duo-normalized decomposition relates to a pair of relations $(\sim, \approx)$ (one for the leading automaton and one for the respective progress DFA). The second is that the relation $\sim$ used in Wilke-algebras is a two-sided congruence, while the relations used by FDFAs are one-sided. Both differences suggest that duo-normalized FDFAs would be more succinct, but this deserves further study.

References

1 D. Angluin, U. Boker, and D. Fisman. Families of DFAs as acceptors of ω-regular languages. *Log. Methods Comput. Sci.*, 14(1), 2018.

2 D. Angluin and D. Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016.

3 R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

4 L. Bohn and C. Löding. Passive learning of deterministic Büchi automata by combinations of DFAs. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, pages 114:1–114:20, 2022.

5 L. Bohn and C. Löding. Constructing deterministic parity automata from positive and negative examples. arXiv:2302.11043. *In print for TheoretiCS, accepted on: 2024-05-11*, 2024.

6 Büchi J. R. On a decision method in restricted second order arithmetic. In *Int. Congress on Logic, Method, and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

7 H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational w-languages. In *9th Inter. Conf. on Mathematical Foundations of Programming Semantics (MFPS)*, pages 554–566, 1993.

8 O. Carton and R. Maceiras. Computing the Rabin index of a parity automaton. *RAIRO Theor. Informatics Appl.*, 33(6):495–506, 1999.

9 O. Carton, D. Perrin, and J-E. Pin. Automata and semigroups recognizing infinite words. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, pages 133–168, 2008.

10 R. Ehlers and S. Schewe. Natural colors of infinite words. In *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, pages 36:1–36:17, 2022.

11 D. Fisman and Y. Lustig. A modular approach for Büchi determinization. In *26th International Conference on Concurrency Theory, CONCUR 2015*, pages 368–382, 2015.

12 Dana Fisman, Emmanuel Goldberg, and Oded Zimerman. A robust measure on FDFAs following duo-normalized acceptance. *arXiv*, 2023. doi:10.48550/arXiv.2310.16022.

13 M. Jurdzinski. Small progress measures for solving parity games. In *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 290–301, 2000.

14 N. Klarlund. A homomorphism concepts for omega-regularity. In *Computer Science Logic, 8th International Workshop, CSL*, pages 471–485, 1994.

15 D. Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.

16 D. Kuperberg, L. Pinault, and D. Pous. Coinductive algorithms for Büchi automata. *Fundam. Informaticae*, 180(4):351–373, 2021.

17 Y. Li, S. Schewe, and Q. Tang. A novel family of finite automata for recognizing and learning ømega-regular languages. In *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA*, pages 53–73, 2023.

18 Y. Li, X. Sun, A. Turrini, Y-F. Chen, and J. Xu. ROLL 1.0: ω -regular language learning library. In *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS*, pages 365–371, 2019.

19 O. Maler and L. Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997.

20 J. Myhill. Finite automata and the representation of events. Technical report, Wright Patterson AFB, Ohio, 1957.

21 A. Nerode. Linear automaton transformations. In *Proceedings of the American Mathematical Society, 9(4)*, pages 541–544, 1958.

22 D. Perrin and J-E Pin. *Infinite words – Automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series.* Elsevier Morgan Kaufmann, 2004.

23 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *21th IEEE Symposium on Logic in Computer Science LICS*, pages 255–264, 2006.

24 S. Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains*, pages 319–327, 1988.

25 S. Schewe. Büchi complementation made tight. In *26th International Symposium on Theoretical Aspects of Computer Science*, pages 661–672, 2009.

26 K. W. Wagner. A hierarchy of regular sequence sets. In *Mathematical Foundations of Computer Science 1975, 4th Symposium, MFCS*, pages 445–449, 1975.

27 T. Wilke. An Eilenberg theorem for infinity-languages. In *Automata, Languages and Programming, 18th International Colloquium, ICALP*, pages 588–599, 1991.

28 T. Wilke and H. Yoo. Computing the rabin index of a regular language of infinite words. *Inf. Comput.*, 130(1):61–70, 1996.

29 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Romeo and Juliet Is EXPTIME-Complete

Harmender Gahlawat
G-SCOP, Grenoble-INP, France

Jan Matyáš Křišťan
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Tomáš Valla
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Abstract

Romeo and Juliet is a two player Rendezvous game played on graphs where one player controls two agents, *Romeo* ($\mathcal{R}$) and *Juliet* ($\mathcal{J}$) who aim to meet at a vertex against k adversaries, called *dividers*, controlled by the other player. The optimization in this game lies at deciding the minimum number of dividers sufficient to restrict $\mathcal{R}$ and $\mathcal{J}$ from meeting in a graph, called the *dynamic separation number*. We establish that Romeo and Juliet is EXPTIME-complete, settling a conjecture of Fomin, Golovach, and Thilikos [Inf. and Comp., 2023] positively. We also consider the game for directed graphs and establish that although the game is EXPTIME-complete for general directed graphs, it is PSPACE-complete and co-W[2]-hard for directed acyclic graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Rendezvous Games on graphs, EXPTIME-completeness, Dynamic Separators

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.54

Funding This work was supported by the Czech Science Foundation Grant no. 24-12046S. This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS23/205/OHK3/3T/18.

1 Introduction

The study of *Rendezvous Games* was initiated by Alpern [2] where two agents, that are randomly placed in some known search region and move at unit speed, aim to meet each other in least expected time. Since then, several variants of rendezvous games have been considered on graphs [3, 10, 24]. Fomin, Golovach, and Thilikos [12] introduced the rendezvous game on graph with *adversaries* where a team of *dividers* aim to prevent the meeting of two passionate lovers, say *Romeo* and *Juliet*. We refer to this game as Romeo and Juliet.

Romeo and Juliet is played on finite, connected, and undirected graphs between two players: *facilitator* and *divider*. The facilitator has two agents, *Romeo*, denoted by $\mathcal{R}$, and *Juliet*, denoted by $\mathcal{J}$, that start the game at two designated vertices s and t of G, respectively. The divider has k agents, $D_1, \ldots D_k$, and their starting position is selected by the divider from vertices in $V(G) \setminus \{s, t\}$. Several divider agents can occupy the same vertex. Afterwards, the divider player and the facilitator player make alternate moves, starting with the facilitator. In a move, a player, for each of its agents, either moves the agent to an adjacent vertex not occupied by any agent of the other player or keeps it on the same vertex. A situation where $\mathcal{R}$ and $\mathcal{J}$ are on the same vertex is a *meet*. The facilitator wins if $\mathcal{R}$ and $\mathcal{J}$ meet, and the divider wins if it succeeds in preventing the meet of $\mathcal{R}$ and $\mathcal{J}$ forever. Accordingly, we have the following decision version of the problem. We define the game formally in Section 2.

Romeo and Juliet
Input: A graph G with two specified vertices s and t, and an integer $k \in \mathbb{N}$.
Question: Can the facilitator win on G starting from s and t against the divider with k agents?

We consider the computational complexity of this game and establish that Romeo and Juliet is EXPTIME-complete, resolving a conjecture of Fomin, Golovach, and Thilikos [12] positively. We further define the game for directed graphs and extend our EXPTIME-completeness result to directed graphs as well. We then establish that the game stays PSPACE-complete for directed acyclic graphs.

Rules of Romeo and Juliet are very similar to the rule of classical Cops and Robber game introduced by Nowakowski and Winkler [20], and Quilliot [21]. Cops and Robber fall in the broad range of *Graph Searching*, where a a set of agents, called *pursuers*, plan to catch one or multiple *evaders* in a graph under some movement rules. We refer to the annotated bibliography by Fomin and Thilikos [13] and recent monographs [6, 7] for further references on this topic.

Observe that if $s = t$ or s and t are adjacent vertices, then the facilitator wins trivially. For distinct non-adjacent vertices s and t, let s, t*-dynamic separation number* be the minimum k such that k dividers have a winning strategy against $\mathcal{R}$ and $\mathcal{J}$ starting at s and t, respectively. Since the dividers have a winning strategy by placing a divider on each vertex of a minimum size s, t-vertex cut (for distinct and non-adjacent vertices), the s, t-dynamic separation number is a well-defined graph invariant for s, t.

Fomin, Golovach, and Thilikos [12] proved that Romeo and Juliet is PSPACE-hard for general graphs. They conjectured that the game is, in fact, EXPTIME-complete. We resolve their conjecture positively by providing the following theorem.

▶ **Theorem 1.** *Romeo and Juliet is EXPTIME-complete for undirected graphs.*

Fomin, Golovach, and Thilikos [12] gave a backtracking based $n^{\mathcal{O}(k)}$ time algorithm for Romeo and Juliet, which is also a $2^{\mathcal{O}(n \log n)}$ time algorithm. Hence, to prove the EXPTIME-completeness, we only need to prove EXPTIME-hardness of Romeo and Juliet. To this end, we provide a non-trivial reduction from GuardUndir (a *guarding game* on undirected graphs), which is known to be EXPTIME-complete [22].

Several graph-searching games have a natural generalization to directed graphs and are well-studied [4, 5, 8, 9, 15, 16, 17]. Romeo and Juliet can also be considered on directed graphs where the agents can only move along the orientations of the arcs. We begin by establishing that the Romeo and Juliet game stays EXPTIME-complete on directed graphs. To this end, we provide a rather easy and straightforward reduction from Guard (a guarding game on directed graphs), which is known to be EXPTIME-complete [23], to Romeo and Juliet on directed graphs.

▶ **Theorem 2.** *Romeo and Juliet is EXPTIME-complete for oriented graphs.*

Next, we consider Romeo and Juliet on directed acyclic graphs (DAGs). Fomin, Golovach, and Thilikos [12] also considered a variant of this game, Romeo and Juliet in Time, where the question is whether $\mathcal{R}$ and $\mathcal{J}$ can meet in at most τ rounds. They established that this game is PSPACE-hard and co-W[2]-hard parameterized by k (for undirected graphs). We provide a general framework to establish computational complexity results for Romeo and Juliet on DAGs. In particular, we define a relaxation of the game –Relaxed Romeo and Juliet– where the dividers have an added relaxation that they can move to a vertex occupied by $\mathcal{R}$ or $\mathcal{J}$, but $\mathcal{R}$ or $\mathcal{J}$ cannot finish a move by occupying the same vertex as a divider. We present a reduction from Romeo and Juliet in Time on general graphs to Romeo and Juliet on directed acyclic graphs. This helps us to translate the hardness results proved for Romeo and Juliet in Time on general graphs to Romeo and Juliet on DAGs. In particular, this establishes that Romeo and Juliet remains PSPACE-hard and co-W[2]-hard parameterized by k even on DAGs. To prove PSPACE-completeness for DAGs, we also provide a polynomial-space algorithm for Romeo and Juliet on DAGs. We show the following results.

▶ **Theorem 3.** *Romeo and Juliet is PSPACE-complete when restricted to directed acyclic graphs.*

▶ **Theorem 4.** *Romeo and Juliet is co-W[2]-hard parameterized by k when restricted to directed acyclic graphs.*

Brief Survey. The study of rendezvous games with adversaries was initiated by Fomin, Golovach, and Thilikos [12] and they conducted an extensive study of the computational complexity of this problem and established the following results. Romeo and Juliet, as well as Romeo and Juliet in Time, are PSPACE-hard and co-W[2]-hard parameterized by k, and both of these problems admit a $n^{\mathcal{O}(k)}$ algorithm. This algorithm is optimal in the sense that, assuming ETH , none of these problems can be solved in $n^{o(k)}$ time. Moreover, Romeo and Juliet in Time is co-NP-complete even for $\tau = 2$, and admit a FPT algorithm parameterized by τ and the neighbourhood diversity of the graph, combined. Interestingly, for chordal graphs and P_5-free graphs, the s, t-dynamic separation number is same as the minimum size of a s, t-vertex cut, which establishes that Romeo and Juliet is polynomial time solvable for these classes.

Misra et al. [18] conducted further analysis of this game from a parameterized complexity perspective and established the following interesting results. Romeo and Juliet is co-para-NP-hard parameterized by the treewidth of the input graph. Further, Romeo and Juliet remains co-W[1]-hard when parameterized by the feedback vertex set number and the solution size (combined), and when parameterized by the pathwidth and the solution size (combined). On the positive side, they established that Romeo and Juliet is FPT when parameterized by the vertex cover number and solution size (combined) by the design of an exponential kernel, and complemented this result by proving that it is unlikely to obtain a polynomial kernel by these parameters. Finally, Romeo and Juliet can be solved in polynomial time for treewidth-2 graphs and grids.

An important part of our result is related to the so-called *guarding game*, introduced by Fomin et al. [11], is played on a graph G by two alternating players, the *cop-player* and the *robber-player*, each having their pawns (c cops and one robber, respectively). The vertex set $V(G)$ is partitioned into a *cop region* C and a *robber region* $R = V(G) \setminus C$, and the goal of the cops is to prevent the robber, who starts at some vertex of R, from entering a vertex of C. The computational complexity of the guarding game depends heavily on the chosen restrictions on the graph G. In particular, if Robber's region (R) is only a path, then the problem can be solved in polynomial time, and when robber moves in a tree (or even in a star), then the problem is NP-complete, and if Robber is moving in a DAG, the problem becomes PSPACE-complete [11]. Later Fomin, Golovach and Lokshtanov [14] studied the *reverse guarding game* with the same rules as in the guarding game, except that the cop-player plays first. They proved that the related decision problem is PSPACE-hard on undirected graphs. Nagamochi [19] has also shown that that the problem is NP-complete even if C induces a 3-star and that the problem is polynomially solvable if R induces a cycle. Also, Reddy, Krishna and Rangan [25] proved that if the robber-region is an arbitrary undirected graph, then the decision problem is PSPACE-hard. Šámal and Valla established that the guarding game is, in fact, ETIME-complete under log-space reductions for both directed [23] as well undirected graphs [22].

Organization. We begin with formal preliminaries and definitions in Section 2. In Section 3 we establish the EXPTIME-completeness of Romeo and Juliet for undirected graphs. In Section 4, we extend our EXPTIME-completeness result to directed graphs. We conclude in Section 5. To respect the space restrictions the section concerning Romeo and Juliet on DAGs has been omitted.

2 Preliminaries

For $\ell \in \mathbb{N}$, let $[\ell] = \{1, \ldots, \ell\}$.

Graph Theory. For a graph G, we denote its vertex set by $V(G)$ and edge set by $E(G)$. We denote the size of $V(G)$ by n and size of $E(G)$ by m. In this paper, we consider finite, connected, and simple graphs. Let v be a vertex of a graph G. Then, by $N(v)$ we denote the *open neighbourhood* of v, that is, $N(v) = \{u \mid uv \in E(G)\}$. By $N[v]$ we denote the *closed neighbourhood* of v, that is, $N[v] = N(v) \cup \{v\}$. For $X \subseteq V(G)$, we define $N_X(v) = N(v) \cap X$ and $N_X[v] = N[v] \cap X$. The *length* of a path or cycle is the number of edges in it. A *u, v-path* is a path with endpoints u and v. For $u, v \in V(G)$, let $d(u, v)$ denote the length of a shortest u, v-path. A path is *isometric* if it is a shortest path between its endpoints.

Computational complexity. The complexity class PSPACE is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space. The class EXPTIME (sometimes denoted EXP) is the set of all decision problems that are solvable by a deterministic Turing machine in the $O(2^{p(n)})$ time where $p(n)$ is a polynomial of n.

Romeo and Juliet. Romeo and Juliet is played on a graph G, where the input prescribes the number of dividers k, and the starting positions s_0 and t_0 of $\mathcal{R}$ and $\mathcal{J}$, respectively. The game starts with $\mathcal{R}$ and $\mathcal{J}$ occupying the initial vertices s_0, and t_0, respectively. Then, the divider player places its agents $D_1, \ldots, D_k$ on vertices $d_0^1, \ldots, d_0^k$, respectively, such that $\{d_0^1, \ldots, d_0^k\} \cap \{s_0, t_0\} = \emptyset$. We call this state $\mathcal{S}_0 = (s_0, t_0, d_0^1, \ldots, d_0^k)$. For $i \geq 0$, let $\mathcal{D}_i$ denote the set $\{d_i^1, \ldots, d_i^k\}$. Multiple dividers may occupy the same vertex, and $|\mathcal{D}_i|$ may be less than k. After this, the game proceed in *rounds*, where each round consists of a divider move, followed by a facilitator move. In round i, $i \geq 1$, first the facilitator moves $\mathcal{R}$ to a vertex $s_i \in N[s_{i-1}] \setminus \mathcal{D}_{i-1}$ and $\mathcal{J}$ to a vertex $t_i \in N[t_{i-1}] \setminus \mathcal{D}_{i-1}$. Then, the divider moves each divider D_p, $p \in [k]$, to a vertex $d_i^p \in N[d_{i-1}^p] \setminus \{s_i, t_i\}$. This gives us a game state $\mathcal{S}_i = (s_i, t_i, d_i^1, \ldots, d_i^k)$. If the facilitator can ensure that for some $i \geq 0$, $s_i = t_i$, we say that the facilitator has a winning strategy. On the other hand, if the divider player can ensure that for each $i \geq 0$, $s_i \neq t_i$, then the divider player has a winning strategy. For directed graphs, the rules are exactly the same with the only difference that the agents can only move along the orientations of the arcs. Fomin et al. [12] gave an algorithm for Romeo and Juliet with running time $\mathcal{O}(2^{\mathcal{O}(n \log n)})$. It is easy to see that the algorithm works for directed graphs as well. Hence, we have the following proposition.

▶ **Proposition 5** ([12]). *Romeo and Juliet is in class EXPTIME on directed as well as undirected graphs.*

We have the following trivial observation that shall be useful to us.

▶ **Observation 6.** *Let $\mathcal{S}_i = (s_i, t_i, d_i^1, \ldots, d_i^k)$ be a game state in some graph G, and let $y \in V(G)$. If there is a s_i, y-path (resp., t_i, y-path) P such that (i) P is an isometric path of length ℓ, (ii) and for each vertex $v \in V(P)$ and $u \in \mathcal{D}_i$, $d(s_i, v) \leq d(u, v)$ (resp., $d(t_i, v) \leq d(u, v)$), then the facilitator player can ensure that $\mathcal{S}_{i+\ell} = (s_{i+\ell} = y, t_{i+\ell}, d_{i+\ell}^1, \ldots, d_{i+\ell}^k)$ (resp., $\mathcal{S}_{i+\ell} = (s_{i+\ell}, t_{i+\ell} = y, d_{i+\ell}^1, \ldots, d_{i+\ell}^k)$).*

Guarding Game. The GuardUndir game is played on an undirected graph G, where $V(G)$ is partitioned into two regions: *Cop region* $C \subset V(G)$, and *Robber region* $R = V(G) \setminus C$. There is a prescribed vertex $r_0 \in R$, where the robber starts. The games begins with the robber occupying the vertex r_0. Then, k cops occupy vertices $c_0^1, \dots, c_0^k$ such that for each $j \in [k]$, $c_0^j \in C$. More than one cop may occupy the same vertex. This gives us game state $\mathcal{G}_0 = (r_0, c_0^1, \dots, c_0^k)$. Let $\mathcal{C}_i$ denote the set of vertices $\{c_0^i, \dots, c_k^i\}$. Then, the game proceeds in rounds. In round i, $i > 0$, first the robber moves to a vertex $r_i \in N[r_{i-1}] \setminus \mathcal{C}_{i-1}$. Then, the cop player moves the cop C_j, $j \in [k]$, to a vertex $c_i^j \in N_C[c_{i-1}^j] \setminus \{r_i\}$. If the robber can ensure that for some $i \geq 0$, $r_i \in C$, then the robber has a winning strategy. Otherwise, if the cops can ensure that for each $i \geq 0$, $r_i \notin C$, then the cop player has a winning strategy.

It is worth noting that the GuardUndir game where the starting position of the robber is not specified is also studied. But for our purposes, we consider the variant where the starting position of the robber, i.e., r_0 is fixed. Furthermore, we assume that $G[C]$ as well as $G[R]$ are connected subgraphs of G.

The Guard game is analog of GuardUndir on directed graphs. The rules of the game are exactly the same as the undirected case with the only change that the agents can only move along the orientations of the arcs. Šámal and Valla [22, 23] proved the following.

▶ **Proposition 7** ([23, 22]). *Guard and GuardUndir are EXPTIME-complete.*

3 Romeo and Juliet is EXPTIME-complete

In this section, we establish that Romeo and Juliet is EXPTIME-complete for undirected graphs. To prove this, we provide a non-trivial reduction from GuardUndir to Romeo and Juliet on undirected graphs. We begin by providing an overview of our reduction.

Overview. First, we make two copies of the cop region C as $C_\mathcal{R}$ and $C_\mathcal{J}$, and two copies of the robber region R as $R_\mathcal{R}$ and $R_\mathcal{J}$ such that $\mathcal{R}$ starts in $R_\mathcal{R}$ and $\mathcal{J}$ starts in $R_\mathcal{J}$. We will have $2k$ dividers. Moreover, we use some gadgets, that we call *secret gardens*, which ensure that after each round, if each of $C_\mathcal{R}$ and $C_\mathcal{J}$ does not host at least k dividers each, $\mathcal{R}$ and $\mathcal{J}$ will meet in one of the secret gardens after a finite number of rounds. Moreover, given that both $C_\mathcal{R}$ and $C_\mathcal{J}$ host at least k dividers each, if $\mathcal{R}$ and $\mathcal{J}$ are able to enter the vertices of $C_\mathcal{R}$ and $C_\mathcal{J}$, respectively, then they will be able to meet in the next round. Hence, for dividers to win, the game is, more or less, similar to restricting $\mathcal{R}$ and $\mathcal{J}$ to enter $C_\mathcal{R}$ and $C_\mathcal{J}$, respectively, where both $C_\mathcal{R}$ and $C_\mathcal{J}$ host exactly k dividers each. Below, we provide our construction in detail.

Construction. Let (H, k, r) be an instance of the GuardUndir, where r is the starting position of the robber and $V(H)$ consists of the cop region C and the robber region $R = V(H) \setminus C$. We assume that $H[C]$ as well as $H[R]$ is connected. We will construct an instance $(G, 2k)$ of Romeo and Juliet in the following manner. Since the construction has several components, we will define the components of our construction (reduction) individually. Our graph G will have following components.

1. $C_\mathcal{R}$, $C_\mathcal{J}$, $R_\mathcal{R}$, and $R_\mathcal{J}$. We begin by constructing two copies of the graph H as $G_\mathcal{R}$ and $G_\mathcal{J}$, corresponding to $\mathcal{R}$ and $\mathcal{J}$, respectively. See Figure 1 for an illustration. More specifically, $G_\mathcal{R}$ (resp., $G_\mathcal{J}$) contains a copy $C_\mathcal{R}$ (resp., $C_\mathcal{J}$) of C and $R_\mathcal{R}$ (resp., $R_\mathcal{J}$) of R. Formally, $V(G_\mathcal{R}) = \{u_\mathcal{R} \mid u \in V(H)\}$ and, $V(G_\mathcal{J}) = \{u_\mathcal{J} \mid u \in V(H)\}$. Moreover,

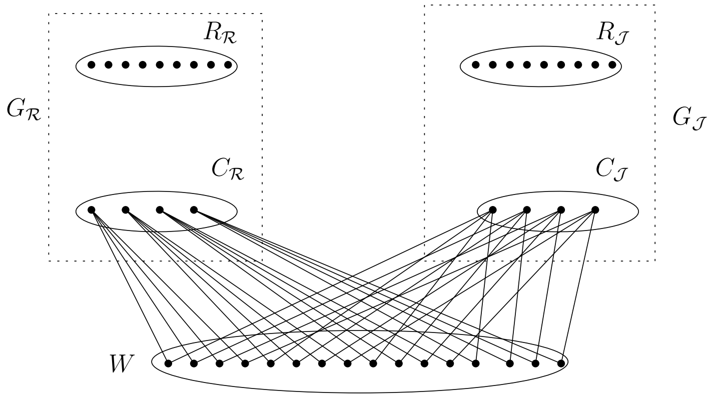


Figure 1 The graphs $G[R_{\mathcal{R}} \cup C_{\mathcal{R}}]$ (i.e., $G_{\mathcal{R}}$) as well as $G[R_{\mathcal{J}} \cup C_{\mathcal{J}}]$ (i.e., $G_{\mathcal{J}}$) are isomorphic to H. Here, we do not display edges in $G_{\mathcal{R}}$ and $G_{\mathcal{J}}$ to ease the illustration.

$E(G_{\mathcal{R}}) = \{u_{\mathcal{R}}v_{\mathcal{R}} \mid uv \in E(H)\}$ and $E(G_{\mathcal{J}}) = \{u_{\mathcal{J}}v_{\mathcal{J}} \mid uv \in E(H)\}$. Finally, the starting position of $\mathcal{R}$ is $r_{\mathcal{R}}$ and $\mathcal{J}$ is $r_{\mathcal{J}}$ (where r is the starting position of the robber in the GuardUndir), i.e., $s_0 = r_{\mathcal{R}}$ and $t_0 = r_{\mathcal{J}}$.

2. Connecting $C_{\mathcal{R}}$ to $C_{\mathcal{J}}$. For each vertex $x \in C_{\mathcal{R}}$ and each vertex $y \in C_{\mathcal{J}}$, we connect x and y using a path $P_{xy} = xw_{xy}y$ of length 2. Let W be the set of vertices that lie in the middle of these paths, i.e., $W = \{w_{xy} \mid x \in C_{\mathcal{R}}, y \in C_{\mathcal{J}}\}$. Moreover, each vertex of W has degree exactly 2 in G. See Figure 1 for an illustration. We have the following trivial observation, which follows from the fact that each vertex in W have degree exactly 2.

▶ **Observation 8.** *Consider a game state $S_i = \{s_i, t_i, d_i^1, \ldots, d_i^{2k}\}$, $i > 0$, such that $s_i \in C_{\mathcal{R}}$ and $t_i \in C_{\mathcal{J}}$, and $w_{s_i t_i} \notin \mathcal{D}_{i-1}$, then $\mathcal{R}$ and $\mathcal{J}$ can meet at $w_{s_i t_i}$ in the next round.*

Proof. The proof follows from the fact that if $w_{s_i t_i} \notin \mathcal{D}_{i-1}$, then $w_{s_i t_i} \notin \mathcal{D}_i$ since $w_{s_i t_i}$ is connected only to s_i and t_i and $\{s_i, t_i\} \cap \mathcal{D}_i = \emptyset$. ◀

3. *Secret Gardens*. Next, we construct $2k+2$ secret gardens $S_1, \ldots, S_{2k+2}$, the goal of which is to ensure that after each round, both $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$ must host exactly k dividers each. See Figure 2 for an illustration. Each S_i, $i \in [2k+2]$, consists of k independent vertices.

4. *Bridges*. Next, we construct $2k+2$ *divider bridges* $B_1, \ldots B_{2k+2}$, where each B_i ($i \in [2k+2]$) consists of k independent vertices, and $2k+2$ *lover bridges* $L_1, \ldots, L_{2k+2}$, where each L_i consists of two independent vertices a_i and b_i. See Figure 2 for an illustration.

5. Connecting bridges and gardens. For each $i \in [2k+2]$, $G[S_i \cup B_i]$ induces a complete bipartite graphs. Next, for $i \in [k+1]$, we connect each vertex of S_i to a_i via an edge and each vertex of b_i to S_i via a path of length 2. Symmetrically, for $k+2 \le i \le 2k+2$, we connect each vertex of S_i to a_i via a path of length 2 and each vertex of S_i to b_i via an edge.

6. Connecting bridges to rest of the graph. Let β_H, β_C, and β_R be the diameter of graph H, graph $H[C]$, and graph $H[R]$, respectively. Then, let $\alpha = \max(\beta_H, \beta_C, \beta_R)$. For each vertex $x \in C_{\mathcal{R}}$ and each vertex $y \in \bigcup_{i \in [k+1]} B_i$, we connect x and y using a path of length α. Similarly, for each vertex $x \in C_{\mathcal{J}}$ and each vertex $y \in \bigcup_{k+1 < i \le 2k+2} B_i$, we connect x and y using a path of length α. Next, for $i \in [k+1]$, we connect each vertex $x \in R_{\mathcal{R}}$ to

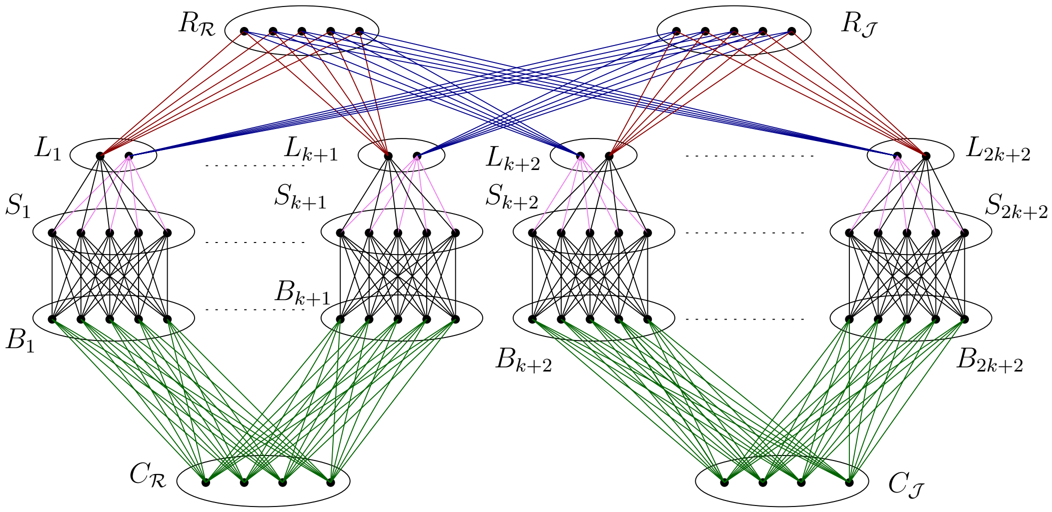


Figure 2 Illustration of secret gardens and bridges. Each black connection signifies an edge, blue, red, green, and violet connections signify paths of length $\alpha+2$, $\alpha+1$, α, and 2, respectively. Note that, due to our choice of α, no shortcuts are created in the original graph.

a_i using a path of length $\alpha+1$ and each vertex $y \in R_{\mathcal{J}}$ to b_i using a path of length $\alpha+2$. Symmetrically, for $k+2 \leq i \leq 2k+2$, we connect each vertex $x \in R_{\mathcal{R}}$ to a_i using a path of length $\alpha+2$ and each vertex $y \in R_{\mathcal{J}}$ to b_i using a path of length $\alpha+1$. See Figure 2 for an illustration. This completes our construction of G.

Subgraphs of G. For the ease of exposition, we name some of the induced subgraphs of G. Consider the induced subgraph $G' = G[V(G) \setminus (R_{\mathcal{R}} \cup R_{\mathcal{J}} \cup W)]$. Observe that G' has two connected components: one containing $C_{\mathcal{R}}$, say $G'_{\mathcal{R}}$, and the other containing $C_{\mathcal{J}}$, say $G'_{\mathcal{J}}$. Moreover, consider the induced subgraph $G'' = G'[V(G') \setminus (C_{\mathcal{R}} \cup C_{\mathcal{J}})]$. Observe that G'' contains exactly $2k+2$ connected components, say $G_1, \ldots, G_{2k+2}$, and let G_i be the connected component containing S_i (and hence, B_i and L_i).

The following observation follows directly from the construction of G

▶ **Observation 9.** *The following statements follow from the construction.*

1. *For distinct $i, j \in [2k+2]$, for a vertex $x \in V(G_i)$ and $y \in S_j$, $d(x,y) > \alpha+1$.*
2. *Let $x \in S_i$ for $i \in [k+1]$. For each vertex $y \in C_{\mathcal{R}}$, there is a x,y-path of length $\alpha+1$, and for each vertex $z \in C_{\mathcal{J}}$, there is a x,z-path of length $\alpha+3$.*
3. *Let $x \in S_i$ for $k+2 \leq i \leq 2k+2$. For each vertex $y \in C_{\mathcal{R}}$, there is a x,y-path of length $\alpha+3$, and for each vertex $z \in C_{\mathcal{J}}$, there is a x,z-path of length $\alpha+1$.*

The following lemma establishes that if $\mathcal{R}$ and $\mathcal{J}$ are in $R_{\mathcal{R}}$ and $R_{\mathcal{J}}$, respectively, then both $G'_{\mathcal{R}}$ and $G'_{\mathcal{J}}$ must host at least k dividers, else $\mathcal{R}$ and $\mathcal{J}$ meet in at most $\alpha+2$ rounds.

▶ **Lemma 10.** *Consider a game state $\mathcal{S}_i = (s_i, t_i, d_i^1, \ldots, d_i^{2k})$ for $i \geq 0$. If $s_i \in R_{\mathcal{R}}$ and $t_i \in R_{\mathcal{J}}$, and at least one of $G'_{\mathcal{R}}$ or $G'_{\mathcal{J}}$ hosts less than k dividers, then $s_{i+\alpha+3} = t_{i+\alpha+3}$.*

Proof. Without loss of generality, let us assume that $G'_{\mathcal{R}}$ hosts at most $k-1$ dividers at the end of round i (i.e., $\mathcal{D}_i \cap V(G'_{\mathcal{R}}) < k$). Therefore, at least one of $G_1, \ldots, G_{k+1}$ does not host any divider (as $G_1, \ldots, G_{k+1}$ are disjoint subgraphs of G'_R). Let G_p, $p \in [k+1]$, be such a component, i.e., $V(G_p) \cap \mathcal{D}_i = \emptyset$. To ease the presentation, let $x = s_i$ and $y = t_i$. Moreover, let P_x be the unique isometric x, a_p-path of length $\alpha+1$, and let P_y be the unique isometric

y, b_p-path of length $\alpha + 2$. Furthermore, let the vertices of S_p be marked as $v_1, \dots, v_k$ and let, for $j \in [k]$, the (degree-2) vertex connecting v_j and b_p be u_j. We have the following crucial claim that proves our lemma.

▷ Claim 11. $\mathcal{R}$ and $\mathcal{J}$ can move along the paths P_x and P_y such that $s_{i+\alpha+2} \in S_p$, $t_{i+\alpha+2} = b_p$. Moreover, if $s_{i+\alpha+2} = v_j$, where $j \in [k]$, then $u_j \notin \mathcal{D}_{i+\alpha+2}$.

Proof of Claim. First, we establish that $\mathcal{R}$ can move to the vertex a_p in $\alpha + 1$ rounds. Due to Observation 6, to show that $\mathcal{R}$ can ensure that $s_{i+\alpha+1} = a_p$, it is sufficient to show that for each vertex $v \in V(P_x)$, $d(x, v) \leq \min_{u \in \mathcal{D}_i}(d(u, v))$. We distinguish the following cases depending on where u is in G.

1. $u \in C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup W$: Observe that for each vertex $u \in C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup W$ and each vertex $v \in V(P_x)$, $d(u, v) \geq \alpha + 1$ and $d(x, v) \leq \alpha + 1$. Hence, $d(x, v) \leq \min_{u \in \mathcal{D}_i}(d(u, v))$.
2. $u \in R_{\mathcal{R}} \cup R_{\mathcal{J}}$: Observe that for each vertex $u \in R_{\mathcal{R}} \cup R_{\mathcal{J}}$ and each vertex $v \in V(P_x)$, any u, v-path contains either x or a_p. In the former case, trivially $d(x, v) \leq d(u, v)$, and in the latter case, observe that $d(u, v) \geq \alpha + 1 \geq d(x, v)$. Hence, $d(x, v) \leq \min_{u \in \mathcal{D}_i}(d(u, v))$.
3. $u \in G_j$, for some $j \in [2k+2]$: Due to our choice of p, clearly $u \notin V(G_p)$ (since we choose G_p such that $V(G_p) \cap \mathcal{D}_i = \emptyset$). Hence, $j \neq p$. Since G_j and G_p are disjoint components in $G[V(G) \setminus \{C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup R_{\mathcal{R}} \cup R_{\mathcal{J}}\}]$, each u, v-path passes through a vertex of $u' \in C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup R_{\mathcal{R}} \cup R_{\mathcal{J}}$, and hence this case is implied by the previous two cases.

Thus, the facilitator can ensure that $s_{i+\alpha+1} = a_p$. Finally, since $G'_{\mathcal{R}}$ hosts at most $k - 1$ dividers and for each vertex $u \in V(G) \setminus V(G'_{\mathcal{R}})$ and $v \in S_p$, $d(u, v) > \alpha + 1$, at most $k - 1$ dividers can reach the vertices of S_p in $\alpha + 1$ moves of dividers. Therefore, $|\mathcal{D}_{i+\alpha+1} \cap S_p| < k$. Hence, there is at least one $j \in [k]$ such that $v_j \in S_p \setminus \mathcal{D}_{i+\alpha+1}$, and hence $\mathcal{R}$ can move to v_j in this round, i.e., $s_{i+\alpha+2} = v_j \in S_p$.

The proof that of the claim $t_{i+\alpha+2} = b_p$ is symmetric to the proof that $s_{i+\alpha+1} = a_p$.

Next, we establish that $u_j \notin \mathcal{D}_{i+\alpha+2}$. Recall that $v_j \notin \mathcal{D}_{i+\alpha+1}$ and, due to our choice of p, $V(G_p) \cap \mathcal{D}_i = \emptyset$. It follows from our construction that for each vertex $w \in V(G) \setminus (G_p \cup C_{\mathcal{R}})$, $d(w, u_j) > \alpha + 2$ and for $w' \in C_{\mathcal{R}}$, $d(w', u_j) = \alpha + 2$ and each w', u_j-path of length $\alpha + 2$ passes through v_j. Since $\mathcal{D}_i \cap V(G_p) = \emptyset$, if a divider, say D_ℓ, has to ensure that $d^{\ell}_{i+\alpha+2} = u_j$, D_ℓ has to ensure that $d^{\ell}_{i+\alpha+1} = v_j$, which is not possible since $v_j \notin \mathcal{D}_{i+\alpha+1}$. Hence, $u_j \notin \mathcal{D}_{i+\alpha+2}$. This completes the proof of our claim. ◁

Finally, since $s_{i+\alpha+2} = v_j \in S_p$ (for some $j \in [k]$), $t_{i+\alpha+2} = b_p$ and $u_j \notin \mathcal{D}_{i+\alpha+2}$ (Claim 11), the facilitator can ensure that $\mathcal{R}$ and $\mathcal{J}$ meet in the next round at u_j. ◀

Next, we have the following lemma which establishes that as long as $\mathcal{R}$ and $\mathcal{J}$ are in $R_{\mathcal{R}}$ and $R_{\mathcal{J}}$, respectively, both $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$ must host exactly k dividers each.

▶ **Lemma 12.** *Consider a game state $\mathcal{S}_i = (s_i, t_i, d^1_i, \dots, d^{2k}_i)$ for $i \geq 0$ such that $s_i \in R_{\mathcal{R}}$ and $t_i \in R_{\mathcal{J}}$. If $|\mathcal{D}_i \cap C_{\mathcal{R}}| \neq k$ or $|\mathcal{D}_i \cap C_{\mathcal{J}}| \neq k$, then the facilitator can ensure that $s_{i+\alpha+3} = t_{i+\alpha+3}$.*

Proof. The proof is similar to the proof of Lemma 10. Due to Lemma 10, we know that both $G'_{\mathcal{R}}$ and $G'_{\mathcal{J}}$ host exactly k dividers, and thus, $|\mathcal{D}_i \cap C_{\mathcal{R}}| \leq k$ and $|\mathcal{D}_i \cap C_{\mathcal{J}}| \leq k$, else $\mathcal{R}$ and $\mathcal{J}$ meet in $\alpha + 3$ rounds. At this point, let one of $C_{\mathcal{R}}$ or $C_{\mathcal{J}}$ hosts less that k dividers. Without loss of generality, let $|\mathcal{D}_i \cap C_{\mathcal{R}}| < k$. In this case, similarly to the proof of Lemma 10, there is at least one G_p, $p \in [k+1]$, such that G_p does not contain any divider . Moreover, similarly to the proof of Lemma 10, only the dividers on $C_{\mathcal{R}}$ can reach S_p in $\alpha + 2$ rounds and hence at most $k - 1$ dividers can reach the vertices of S_p in $\alpha + 2$ rounds. Therefore, $\mathcal{R}$ and $\mathcal{J}$ have a strategy to ensure that $s_{i+\alpha+2} = z \in S_p$ and $t_{i+\alpha+2} = b_p$ such that the unique vertex on the isometric $s_{i+\alpha+2}, t_{i+\alpha+2}$-path does not any divider in this round. Hence, $s_{i+\alpha+3} = t_{i+\alpha+3}$. ◀

Next, we prove the following lemma which implies one side of our reduction.

▶ **Lemma 13.** *If the robber has a winning strategy in H against k cops, then $\mathcal{R}$ and $\mathcal{J}$ have a winning strategy in G against $2k$ dividers.*

Proof. The game starts with $\mathcal{R}$ and $\mathcal{J}$ placed on $r_{\mathcal{R}}$ and $r_{\mathcal{J}}$, respectively. Due to Lemma 12, we can assume that as long as $\mathcal{R}$ and $\mathcal{J}$ are in $R_{\mathcal{R}}$ and $R_{\mathcal{J}}$, respectively, both $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$ must host exactly k dividers each, else $\mathcal{R}$ and $\mathcal{J}$ win. First, we will show that after a finite number of rounds $\mathcal{R}$ can enter $C_{\mathcal{R}}$. Observe that any move of dividers in $C_{\mathcal{R}}$ (in G) corresponds to a valid move of cops in C (in H) and $\mathcal{R}$ can make any move in $G[C_{\mathcal{R}} \cup R_{\mathcal{R}}]$ against dividers in G that is accessible to the robber in H against the corresponding position of the cops. Hence, using the winning strategy of the robber, $\mathcal{R}$ can enter $C_{\mathcal{R}}$ after a finite number of rounds (since $C_{\mathcal{R}}$ hosts exactly k dividers). Similarly, $\mathcal{J}$ can move to $C_{\mathcal{J}}$ in a finite number of rounds.

But, observe that since the dividers in $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$ may be using different strategies, $\mathcal{R}$ and $\mathcal{J}$ may not be able to simultaneously move to $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$, respectively. Hence, we distinguish the following two cases.

1. $\mathcal{R}$ and $\mathcal{J}$ move simultaneously to $x \in C_{\mathcal{R}}$ and $y \in C_{\mathcal{J}}$, respectively. Due to Lemma 12, we may assume that when $\mathcal{R}$ and $\mathcal{J}$ made this move, both $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$ hosted exactly k dividers each, and hence no vertex of W was occupied by any divider. Hence, $\mathcal{R}$ and $\mathcal{J}$ can meet in the next round due to Observation 8.
2. $\mathcal{R}$ moves to $x \in C_{\mathcal{R}}$ while $\mathcal{J}$ is on some vertex $y \in R_{\mathcal{J}}$. Similarly to the previous case, due to Lemma 12, we may assume that when $\mathcal{R}$ and $\mathcal{J}$ made this move, both $C_{\mathcal{R}}$ and $C_{\mathcal{J}}$ hosted exactly k dividers each. In the next move of the dividers, observe that there will be at least one G_p, $p \in [k+1]$, such that G_p does not contain any dividers. In the next $\alpha + 2$ rounds, $\mathcal{J}$ moves towards the vertex b_p and $\mathcal{R}$ moves first towards a vertex in S_p, and then to a neighbour of b_p. We note that $\mathcal{R}$ and $\mathcal{J}$ can make these moves following the same arguments presented in the proof of Lemma 10. Hence, $\mathcal{R}$ and $\mathcal{J}$ meet in the next round since they are they are at adjacent vertices.
3. $\mathcal{J}$ moves to $x \in C_{\mathcal{J}}$ while $\mathcal{R}$ is on some vertex $y \in R_{\mathcal{R}}$. This case is symmetric to Case 2.

This completes our proof. ◀

To complete our reduction, we need to establish that if k cops have a winning strategy in H, then $2k$ dividers have a winning strategy in G. To aid the presentation of the proof of this lemma, we need the following notion of *safe states*.

Safe states. Consider the GUARDUNDIR on graph H and consider some game state $\mathcal{G}_i = (r_i, c_i^1, \ldots, c_i^k)$ such that $r_i \in R$. We say that $\mathcal{G}_i$ is a *safe state* if the cops have a strategy to ensure that robber cannot enter C in any round $i' > i$. Similarly, consider the game ROMEO AND JULIET on graph G and some game state $\mathcal{S}_j = (s_j, t_j, d_j^1, \ldots, d_j^{2k})$. We say that $\mathcal{S}_j$ is *$\mathcal{R}$-safe state* (resp., *$\mathcal{J}$-safe state*) if (1) $s_j \in R_{\mathcal{R}}$ (resp., $t_j \in R_{\mathcal{J}}$), (2) $\{d_1^j, \ldots, d_k^j\} \subseteq C_{\mathcal{R}}$ (resp., $\{d_j^{k+1}, \ldots, d_j^{2k}\} \subseteq C_{\mathcal{J}}$), and (3) the dividers $D_1, \ldots, D_k$ (resp., $D_{k+1}, \ldots, D_{2k}$) can ensure that if $\mathcal{R}$ (resp., $\mathcal{J}$) is restricted to $G_{\mathcal{R}}$ (resp., $G_{\mathcal{J}}$), then $\mathcal{R}$ (resp., $\mathcal{J}$) cannot enter a vertex of $C_{\mathcal{R}}$ (resp., $C_{\mathcal{J}}$) in any round $j' > j$. We say that a game state $\mathcal{S}_j$ is *almost-$\mathcal{R}$-safe state* if $\mathcal{R}$ is in some G_i, $i \in [2k+2]$, and the dividers $D_1, \ldots, D_k$ have a strategy that ensures that if for some $j' > j$, $s_{j'} \in V(G_{\mathcal{R}}) \cup V(G_{\mathcal{J}})$ (i.e., $\mathcal{R}$ enters a vertex in $V(G_{\mathcal{R}}) \cup V(G_{\mathcal{J}})$), then $\mathcal{S}_{j'}$ is a $\mathcal{R}$-safe state. We define *almost-$\mathcal{J}$-safe state* analogously. We have the following easy, but useful, observations.

▶ **Observation 14.** *Consider a game state $\mathcal{S}_j = (s_j, t_j, d_j^1, \dots, d_j^{2k})$ of* Romeo and Juliet *on G such that $s_j \in R_{\mathcal{R}}$ and $\{d_j^1, \dots, d_j^k\} \subseteq C_{\mathcal{R}}$. Now, consider a corresponding game state $\mathcal{G}_i = (r_i, c_i^1, \dots, c_i^k)$ of* GuardUndir *on H such that if $s_j = u_{\mathcal{R}}$, then $r_i = u$, and if, for $\ell \in [k]$, $d_j^\ell = v_{\mathcal{R}}$, then $c_j^\ell = v$. If $\mathcal{G}_i$ is a safe state then $\mathcal{S}_j$ is a $\mathcal{R}$-safe state.*

Proof. To prove our statement, we need to show that if $\mathcal{R}$ is restricted to $G_{\mathcal{R}}$, then the dividers have a strategy to ensure that, for $j' > j$, $\mathcal{R}$ cannot enter a vertex of $C_{\mathcal{R}}$. The dividers $D_1, \dots, D_k$ can do so by mimicking the winning strategy of the cops from H in the following manner. Whenever $\mathcal{R}$ moves from a vertex $u_{\mathcal{R}}$ to $w_{\mathcal{R}}$ in $G_{\mathcal{R}}$, we move the robber in H from u to w, and then each cop C_ℓ, for $\ell \in [k]$, moves as per its winning strategy. Notice that this gives us a safe state $\mathcal{S}_{i+1}$. Then, each divider D_ℓ, for $\ell \in [k]$, copies the move of C_i such that if C_i moved from v to x, then D_i moves from $v_{\mathcal{R}}$ to $x_{\mathcal{R}}$. Using this strategy, the dividers $D_1, \dots, D_k$ can ensure that, as long as $\mathcal{R}$ is restricted to $G_{\mathcal{R}}$, $\mathcal{R}$ can never enter a vertex of $C_{\mathcal{R}}$. Hence, $\mathcal{S}_j$ is a $\mathcal{R}$-safe state. ◀

Analogously, we can have the following observation for $\mathcal{J}$-safe states.

▶ **Observation 15.** *Consider a game state $\mathcal{S}_j = (s_j, t_j, d_j^1, \dots, d_j^{2k})$ of* Romeo and Juliet *on G such that $t_j \in R_{\mathcal{J}}$ and $\{d_j^{k+1}, \dots, d_j^{2k}\} \subseteq C_{\mathcal{J}}$. Now, consider a corresponding game state $\mathcal{G}_i = (r_i, c_i^1, \dots, c_i^k)$ of* GuardUndir *on H such that if $t_j = u_{\mathcal{J}}$, then $r_i = u$, and if, for $k+1 \leq \ell \leq 2k$, $d_j^\ell = v_{\mathcal{J}}$, then $c_j^{\ell - k} = v$. If $\mathcal{G}_i$ is a safe state then $\mathcal{S}_j$ is a $\mathcal{J}$-safe state.*

▶ **Observation 16.** *Let k cops have a winning strategy in H against the robber starting at r. Then, for each vertex $v \in R$ ($\subseteq V(H)$), there exists a set of (not necessarily distinct) vertices $u_1, \dots, u_k \in C$ such that the game state $\mathcal{G} = (v, u_1, \dots, u_k)$ is a safe state.*

Proof. Targeting contradiction, let there be a vertex $v \in C$ such that for every $u_1, \dots, u_k \in C$ (possibly $u_i = u_j$ for distinct i, j), $\mathcal{G} = (v, u_1, \dots, u_k)$ is a not a safe state. Let $d(r, v) = \ell \leq \alpha$. Then, the robber have a strategy to reach a game state $\mathcal{G}_\ell = (r_\ell = v, c_\ell^1, \dots, c_\ell^k)$, which is not a safe state (by our contradiction assumption). Hence, the cops do not have any strategy that can restrict the robber to R for each round $\ell' > \ell$, a contradiction to the fact that k cops have a winning strategy in H against the robber starting at r. ◀

Next, we have the following lemma.

▶ **Lemma 17.** *Let k cops have a winning strategy in H against the robber starting at r. Moreover, let $\mathcal{S}_j = (s_j, t_j, d_j^1, \dots, d_j^{2k})$ be a game state in G. Then, the following are true.*

1. *If $s_j = a_p$ for some $p \in [k+1]$ and each vertex of S_p is occupied by a divider from $D_1, \dots, D_k$, then $\mathcal{S}_j$ is an almost-$\mathcal{R}$-safe state.*
2. *If $s_j = a_p$ for some $k+2 \leq p \leq 2k+2$, and each vertex of B_p is occupied by some divider from $D_1, \dots, D_k$, then $\mathcal{S}_j$ is an almost-$\mathcal{R}$-safe state.*

Proof. First, we will show that if $s_j = a_p$ for some $p \in [k+1]$ and each vertex of S_p is occupied by a divider from $D_1, \dots, D_k$, then $\mathcal{S}_j$ is an almost-$\mathcal{R}$-safe state. Let the vertices of S_p be marked $v_1, \dots, v_k$, and without loss of generality, let us assume that $d_j^i = v_i$, for $i \in [k]$. The dividers will maintain the following invariant for $\ell \geq j$: $d(d_\ell^i, v_i) = d(s_\ell, a_p)$. (We have that $d(d_j^i, v_i) = d(s_j, a_p) = 0$.) This invariant will ensure that whenever $\mathcal{R}$ is at the vertex a_p, each vertex of S_p is occupied by some divider, and hence, $\mathcal{R}$ can never access a vertex of S_p. Let $j' > j$ be the smallest integer such that $s_{j'} \in V(G_{\mathcal{R}}) \cup V(G_{\mathcal{C}})$. If no such j' exists, then $\mathcal{S}_j$ is trivially almost-$\mathcal{R}$-safe state, and hence we assume that j' exists. To establish that $\mathcal{S}_j$ is an almost-$\mathcal{R}$-safe state, we need to show the following:

1. $s_{j'} \in R_{\mathcal{R}}$: This is easy to see since $S_p \cup R_{\mathcal{R}}$ separates a_p from each vertex in $R_{\mathcal{J}} \cup C_{\mathcal{R}} \cup C_{\mathcal{J}}$ and whenever $\mathcal{R}$ is at a_p, all vertices of S_p are occupied by the dividers.
2. $\mathcal{S}_{j'}$ is a $\mathcal{R}$-safe state: To ensure this, the dividers implement the following strategy which maintains the invariant $d(d^i_\ell, v_i) = d(s_\ell, a_p)$ (for $\ell \geq j$ and $i \in [k]$). Let $\gamma \geq j$ be the least integer such that $s_\gamma = a_p$ and $s_{\gamma+1} \neq a_p$. Observe that there is a unique vertex $v_{\mathcal{R}}$ in $R_{\mathcal{R}}$ such that $d(s_{\gamma+1}, v_{\mathcal{R}}) = \alpha$. (For every other vertex $w_{\mathcal{R}} \in R_{\mathcal{R}} \setminus \{v_{\mathcal{R}}\}$, $\alpha + 1 \leq d(s_{\gamma+1}, v_{\mathcal{R}}) \leq \alpha + 2$). Since k cops have a winning strategy in H, due to Observation 16, we know that there exists a safe state $\mathcal{G} = (v, u_1, \dots, u_k)$ in H such that $u_1, \dots, u_k \in C$. Moreover, due to Observation 14, we know that, a game state $\mathcal{S}_{j''} = (s_{j''} = v_{\mathcal{R}}, t_{j''}, d^1_{j''}, \dots, d^{2k}_{j''})$ such that, for $i \in [k]$, $d^i_{j''} = u_{i\mathcal{R}}$ is a $\mathcal{R}$-safe state. Now, each divider D_i chooses chooses a $v_i, u_{i\mathcal{R}}$-path, say P_i, of length $\alpha + 1$ and move on this path to maintain the invariant $d(d^i_\ell, v_i) = d(s_\ell, a_p)$ (for $\ell \geq j$ and $i \in [k]$). Now, we distinguish the following two cases:
   a. For $\gamma < \gamma' < j'$, $s_{\gamma'} \neq a_p$. In this case, observe that $s_{j'} = v_{\mathcal{R}}$ and $d^i_{j'} = u_{i\mathcal{R}}$, for $i \in [k]$, which gives a $\mathcal{R}$-safe state.
   b. There is some $\gamma < \gamma' < j'$, such that $s_{\gamma'} = a_p$. Observe that the game state $\mathcal{S}_{\gamma'}$ is identical to the state $\mathcal{S}_j$ from the perspective of $\mathcal{R}$ and $D_1, \dots, D_k$ (due to our invariant). Hence, the dividers $D_1, \dots, D_k$ can restart their strategy that they had starting from $\mathcal{S}_j$.

Second, we will establish that if $s_j = a_p$ for some $k+2 \leq p \leq 2k+2$ and each vertex of B_p is occupied by a divider from $D_1, \dots, D_k$, then $\mathcal{S}_j$ is an almost-$\mathcal{R}$-safe state. It follows directly from our construction of G that for each vertex $y \in C_{\mathcal{R}}$ and $z \in B_p$, there is a y, z-path of length $\alpha + 2$ (that passes through W and $C_{\mathcal{J}}$). Similarly, for each vertex $z \in R_{\mathcal{R}}$, there is a a_p, z-path of length $\alpha + 2$. The proof is similar to the proof of the first case and hence we will provide the proof in a succinct manner. Let the vertices of B_p be marked $v_1, \dots, v_k$, and without loss of generality let us assume that $d^i_j = v_i$ (for $i \in [k]$). Furthermore, let the vertices of S_p be marked $x_1, \dots, x_k$ and let the unique vertex connected a_p and x_i be y_i. Now, the dividers will use the following strategy for $j' \geq j$ while always maintaining the invariant: $d(s_{j'}, a_p) = d(d^i_{j'}, v_i)$ for $i \in [k]$.

- If $s_{j'} = y_q$ for some $q \in [k]$, then for each $i \in [k]$, the divider D_i moves to the vertex x_i, i.e., $d^i_{j'} = x_i$. Observe that this ensures that $\mathcal{R}$ can never reach a vertex of S_p, and hence, whenever $\mathcal{R}$ will enter a vertex of $C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup R_{\mathcal{R}} \cup R_{\mathcal{J}}$, it will do so at a vertex of $R_{\mathcal{R}}$.
- If $s_{j'-1} = a_p$ and $s_{j'} \notin \{a_p\} \cup \{y_1, \dots, y_p\}$. Then, let v be the unique vertex in $R_{\mathcal{R}}$ such that $d(v, s_{j'}) = \alpha + 1$. Since k cops have a winning strategy in H, due to Observation 16, we know that there exists a safe state $\mathcal{G} = (v, u_1, \dots, u_k)$ in H such that $u_1, \dots, u_k \in C$. Moreover, due to Observation 14, we know that, a game state $\mathcal{S}_{j''} = (s_{j''} = v_{\mathcal{R}}, t_{j''}, d^1_{j''}, \dots, d^{2k}_{j''})$ such that, for $i \in [k]$, $d^i_{j''} = u_{i\mathcal{R}}$ is a $\mathcal{R}$-safe state. Now, each divider D_i chooses chooses a $v_i, u_{i\mathcal{R}}$-path, say P_i, of length $\alpha + 2$ and move on this path to maintain the invariant $d(d^i_\ell, v_i) = d(s_\ell, a_p)$ (for $\ell \geq j$ and $i \in [k]$). If $\mathcal{R}$ reaches the vertex $v_{\mathcal{R}}$, then observe that we have reached a safe state. Otherwise, if $\mathcal{R}$ reaches to a_p in some round $\gamma > j'$, then we restart our strategy.

This completes our proof. ◀

Next, we have the following lemma, whose proof is identical to the proof of Lemma 17.

▶ **Lemma 18.** *Let k cops have a winning strategy in H against the robber starting at r. Moreover, let $\mathcal{S}_j = (s_j, t_j, d^1_j, \dots, d^{2k}_j)$ be a game state in G. Then, the following are true.*

1. *If $t_j = a_p$ for some $p \in [k+1]$ and each vertex of B_p is occupied by a divider from $D_{k+1}, \dots, D_{2k}$, then $\mathcal{S}_j$ is an almost-$\mathcal{J}$-safe state.*
2. *If $s_j = a_p$ for some $k+2 \leq p \leq 2k+2$, and each vertex of S_p is occupied by some divider from $D_{k+1}, \dots, D_{2k}$, then $\mathcal{S}_j$ is an almost-$\mathcal{J}$-safe state.*

The following observation is directly implied by our construction.

▶ **Observation 19.** *Consider the connected components of the graph $G[V(G) \setminus (C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup S_1 \cup \cdots \cup S_{2k})]$. The components containing $R_{\mathcal{R}}$ and the component containing $R_{\mathcal{J}}$ are disjoint, and let them be named $F_{\mathcal{R}}$ and $F_{\mathcal{J}}$, respectively.*

Due to Observation 19, if we can show that k of the dividers, say $D_1, \ldots, D_k$, can restrict $\mathcal{R}$ from entering any vertex of $C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup S_1 \cup \cdots \cup S_{2k}$, and the other k dividers, say $D_{k+1}, \ldots, D_{2k}$ can restrict $\mathcal{J}$ from entering any vertex of $C_{\mathcal{R}} \cup C_{\mathcal{J}} \cup S_1 \cup \cdots \cup S_{2k}$, then $\mathcal{R}$ and $\mathcal{J}$ never be able to meet as they will be restricted to disjoint subgraphs $F_{\mathcal{R}}$ and $F_{\mathcal{J}}$, respectively, of G. We use a similar strategy to prove the following lemma, which proves the other side of our reduction.

▶ **Lemma 20.** *If k cops have a winning strategy in H against the robber starting at r, then $2k$ dividers have a winning strategy in G against $\mathcal{R}$ and $\mathcal{J}$ starting at $r_{\mathcal{R}}$ and $r_{\mathcal{J}}$, respectively.*

Proof. Since k cops have a winning strategy against the robber starting at r in H, there is a safe state $\mathcal{G}_0 = (r_0 = r, c_0^1, \ldots, c_0^k)$. Now, we begin Romeo and Juliet on G with the game state $\mathcal{S}_0 = (s_0 = r_{\mathcal{R}}, t_0 = r_{\mathcal{J}}, d_0^1, \ldots, d_0^{2k})$ such that if $c_0^\ell = v$, for $\ell \in [k]$, then $d_0^\ell = v_{\mathcal{R}}$ and $d_0^{\ell+k} = v_{\mathcal{J}}$. It follows from Observations 14 and 15 that $\mathcal{S}_0$ is $\mathcal{R}$-safe state as well as $\mathcal{J}$-safe state. Therefore, as long as $\mathcal{R}$ (resp., $\mathcal{J}$) is restricted to $G_{\mathcal{R}}$ (resp., $G_{\mathcal{J}}$), they cannot enter a vertex of $C_{\mathcal{R}}$ (resp., $C_{\mathcal{J}}$). Hence, unless at least one of $\mathcal{R}$ or $\mathcal{J}$ leaves $G_{\mathcal{R}}$ or $G_{\mathcal{J}}$, respectively, to move to some G_i, for $i \in [2k+2]$, they cannot meet. First, we prove the following claim.

▷ Claim 21. Let $\mathcal{S}_j$ be a $\mathcal{R}$-safe state such that $s_{j+1} \notin R_{\mathcal{R}}$. Then, for $j' > j$, the dividers $D_1, \ldots, D_k$ have a strategy that can ensure the following:

1. $s_{j'} \in F_{\mathcal{R}}$.
2. If $s_{j'} \in R_{\mathcal{R}}$, then $\mathcal{S}_{j'}$ is a $\mathcal{R}$-safe state.

Proof of Claim. Let P_x be the unique x, a_p-path that contains s_{j+1}. For $\ell \in [k]$, let $d_j^\ell = y_\ell$ (i.e., in round j, the divider D_ℓ occupies the vertex $y_\ell \in C_{\mathcal{R}}$). We distinguish the following two cases depending on whether $p \leq k$ or $p > k$.

Case 1: $p \leq k$. Let the vertices of S_p be marked $v_1, \ldots, v_k$. It follows from the construction that $d(y_\ell, v_\ell) = \alpha + 1$ and $d(x, a_p) = \alpha + 1$. Each D_ℓ chooses a y_ℓ, v_ℓ-path of length $\alpha + 1$, say P_ℓ, and move along it to maintain the following invariant for $j' > j$: $d(s_{j'}, a_p) = d(d_{j'}^\ell, v_\ell)$. Again, we have the following cases depending on the moves of $\mathcal{R}$.

1. $\mathcal{R}$ never reaches the vertex a_p: In this case observe that when $\mathcal{R}$ moves to $R_{\mathcal{R}}$, say in round j', it can only move to the vertex x (i.e., $s_{j'} = x$). In this case, observe that $d_{j'}^\ell = y_\ell$. Since this state is identical to $\mathcal{S}_j$ with respect to the placement of $\mathcal{R}$ and the dividers $D_1, \ldots D_k$, $\mathcal{S}_{j'}$ is a $\mathcal{R}$-safe state. Moreover, it is easy to see that $\mathcal{R}$ was restricted to $F_{\mathcal{R}}$ for each round $j \leq j'' \leq j'$.
2. $\mathcal{R}$ reaches the vertex a_p in some round j'': In this case, observe that each vertex v_ℓ of S_p is occupied by the divider D_ℓ (due to our invariant), and hence, $\mathcal{R}$ cannot move to a vertex of S_p in round $j'' + 1$. Observe that the game state $\mathcal{S}_{j''}$ is an almost-$\mathcal{R}$-safe state due to Lemma 17. Therefore $\mathcal{R}$ is restricted to $F_{\mathcal{R}}$ and whenever $\mathcal{R}$ reaches a vertex of $R_{\mathcal{R}}$ in some round j', then $\mathcal{S}_{j'}$ is a $\mathcal{R}$-safe state due to the definition of almost-$\mathcal{R}$-safe state.

Case 2: $p > k$. The proof of this case is similar to the proof of Case 1. Let the vertices of B_p be marked $v_1, \ldots, v_k$. It follows from the construction that $d(y_\ell, v_\ell) = \alpha + 2$ and $d(x, a_p) = \alpha + 2$. Each D_ℓ chooses a y_ℓ, v_ℓ-path of length $\alpha + 2$, say P_ℓ, and move along it to maintain the following invariant for $j' > j$: $d(s_{j'}, a_p) = d(d^\ell_{j'}, v_\ell)$. If $\mathcal{R}$ never reaches the vertex a_p, then our invariant implies that whenever $\mathcal{R}$ will enter $R_\mathcal{R}$, it will enter at a safe state. If $\mathcal{R}$ reaches the vertex a_p, then each vertex of B_p is occupied by a divider (due to our invariant), which is an almost-$\mathcal{R}$-safe state due to Lemma 17. Hence, if $\mathcal{R}$ enters a vertex $R_\mathcal{R}$ in some round j', then $\mathcal{S}_{j'}$ is a $\mathcal{R}$-safe state. Further, observe that in all these rounds, $\mathcal{R}$ is restricted to $F_\mathcal{R}$. This completes the proof of this case. ◁

Next, we have the following claim whose proof is symmetric to the proof of Claim 21.

▷ Claim 22. Let $\mathcal{S}_j$ be a $\mathcal{J}$-safe state such that $t_{j+1} \notin R_\mathcal{J}$. Then, for $j' > j$, the dividers $D_{k+1}, \ldots, D_{2k}$ have a strategy that can ensure the following:

1. $t_{j'} \in F_\mathcal{J}$.
2. If $t_{j'} \in R_\mathcal{J}$, then $\mathcal{S}_{j'}$ is a $\mathcal{J}$-safe state.

Finally, our proof follows from the following facts. Since we start from a state that is $\mathcal{R}$-safe state as well as $\mathcal{J}$-safe state, as long as $\mathcal{R}$ (resp., $\mathcal{J}$) is restricted to $G_\mathcal{R}$ (resp., $G_\mathcal{J}$), they cannot enter a vertex of $C_\mathcal{R}$ (resp., $C_\mathcal{J}$). Moreover, even if $\mathcal{R}$ (resp., $\mathcal{J}$) leave $G_\mathcal{R}$ (resp., $G_\mathcal{J}$), it is restricted to $F_\mathcal{R}$ (resp. $F_\mathcal{J}$), and whenever $\mathcal{R}$ (resp., $\mathcal{J}$) return to $G_\mathcal{R}$ (resp., $G_\mathcal{J}$), it returns to a $\mathcal{R}$-safe state (resp., $\mathcal{J}$-safe state), due to Claim 21 (resp., Claim 22). Since this strategy restricts $\mathcal{R}$ and $\mathcal{J}$ to two disjoint subgraphs $F_\mathcal{R}$ and $F_\mathcal{J}$, respectively, of G, $\mathcal{R}$ and $\mathcal{J}$ will never be able to meet. Hence, $2k$ dividers have a winning strategy in G against $\mathcal{R}$ and $\mathcal{J}$ starting at $r_\mathcal{R}$ and $r_\mathcal{J}$, respectively. This completes our proof. ◀

Finally, we haven the following theorem due to our construction of G from H, Propositions 5 and 7, and Lemmas 13 and 20.

▶ **Theorem 1.** *Romeo and Juliet is EXPTIME-complete for undirected graphs.*

4 EXPTIME-Completeness for Directed Graphs

In this section, we establish that Romeo and Juliet is EXPTIME-complete on directed graphs. Due to Proposition 5, to complete our proofs, we only need to establish that Romeo and Juliet is EXPTIME-hard. To this end, we will reduce Guard to Romeo and Juliet on directed graphs. This is a rather straightforward and easy construction.

Construction. Let $(\overrightarrow{G}, k, r)$ be an instance of Guard (where r is the starting position of the robber and $V(\overrightarrow{G})$ consists of the $C \cup R$). We construct an instance $(\overrightarrow{H}, k, s, t)$ in the following manner. Let $V(\overrightarrow{H}) = V(\overrightarrow{G}) \cup \{s, t, d_1, \ldots, d_k\}$, and let $D = \{d_1, \ldots, d_k\}$. Next, $E(\overrightarrow{H}) = E(\overrightarrow{G}) \cup \{\overrightarrow{sr}, \overrightarrow{sd_1}, \ldots, \overrightarrow{sd_k}, \overrightarrow{d_1t}, \ldots, \overrightarrow{d_kt}\}$. Moreover, for each d_i, $i \in [k]$, and $v \in C$, we add an arc $\overrightarrow{d_ic}$. Furthermore, for each vertex $u \in C$, we add an arc $\overrightarrow{ut}$. See Figure 3 for an illustration. Finally, the starting position for $\mathcal{R}$ is s and for $\mathcal{J}$ is t. This completes the construction. The following observation follows directly from our construction.

▶ **Observation 23.** *The following statements are true.*

1. *The vertex t is a sink, and hence $\mathcal{J}$ cannot move throughout the game.*
2. *If at any point in the game $\mathcal{R}$ is on a vertex in C, then $\mathcal{R}$ and $\mathcal{J}$ meet in the next step.*
3. *If there are less than k dividers, then $\mathcal{R}$ and $\mathcal{J}$ meet in at most 2 rounds.*

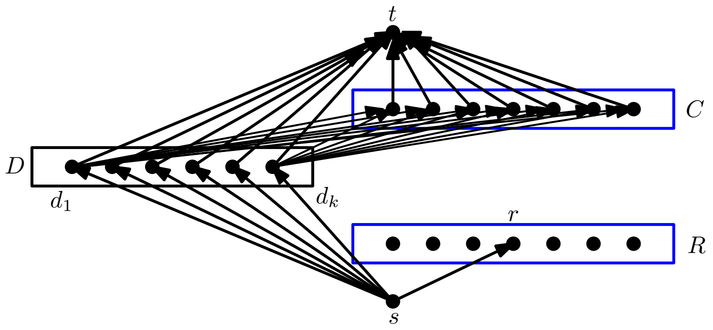


Figure 3 Here, we do not show edges of $\overrightarrow{G}$ to ease the presentation.

Proof. The proof of (1) and (2) follows directly from the construction and the game definition. To see the proof of (3), observe that if there are less than k dividers, there is at least one vertex in D, say x, that is not occupied by any of the dividers (since $|D| = k$). Hence, $\mathcal{R}$ can move to x in the first round, and since $\overrightarrow{xt}$ is an edge, $\mathcal{R}$ and $\mathcal{J}$ meet in the next round. ◀

The following lemma proves the soundness of our reduction.

▶ **Lemma 24.** *In $\overrightarrow{G}$, k cops have winning strategy against the robber starting at r if and only if k dividers have a winning strategy in $\overrightarrow{H}$ against $\mathcal{R}$ and $\mathcal{J}$ starting at s and t, respectively.*

Proof. In one direction, let k cops have a strategy to prevent the robber, who starts at r, from entering C. Then, we prove that k dividers have a winning strategy $\overrightarrow{H}$ against $\mathcal{R}$ and $\mathcal{J}$ starting at s and t, respectively. The k dividers begin with occupying each vertex of D. This restricts $\mathcal{R}$ from entering a vertex of D. Recall that $\mathcal{J}$ cannot move throughout the game (due to Observation 23). Now, the only move possible for $\mathcal{R}$ is to move to vertex r from s. Whenever $\mathcal{R}$ moves to r, the dividers move to the vertices in C where the cops begin in their winning strategy in $\overrightarrow{G}$. Now, observe that $\mathcal{R}$ cannot access s or any vertex in D. Hence, the dividers can restrict $\mathcal{R}$ and $\mathcal{J}$ from meeting by simply restricting $\mathcal{R}$ to ever enter C. Note that the the dividers can do so following the winning strategy for the cops as the rules of movement are the same for both of the games. Hence, k dividers have a winning strategy in $\overrightarrow{H}$ against $\mathcal{R}$ and $\mathcal{J}$ starting at s and t, respectively.

In the other direction, let the robber has a strategy to enter C in G starting from r. In this case, observe that in the beginning, k dividers have to occupy the vertices of D, else $\mathcal{R}$ and $\mathcal{J}$ meet in at most two steps ($\mathcal{R}$ moves to a vertex in D and then meet $\mathcal{J}$ in the next round). Now, $\mathcal{R}$ moves to r from s. Now, at most k dividers move to C. Since the dividers can make the same moves as cops and the robber have a winning strategy against any strategy of k cops to enter C starting from r, $\mathcal{R}$ can use the same strategy to enter C. Hence, $\mathcal{R}$ can enter C after a finite number of rounds, and then meet $\mathcal{J}$ in the next round at t (due to Observation 23). ◀

Hence, we have the following theorem as a consequence of our reduction, Propositions 5 and 7, and Lemma 24.

▶ **Theorem 2.** *Romeo and Juliet is EXPTIME-complete for oriented graphs.*

5 Conclusion

In this work, we considered Romeo and Juliet on directed as well as undirected graphs and established that the game is EXPTIME-complete in both cases, and that the game remains PSPACE-complete even for directed acyclic graphs. Moreover, we defined a game Relaxed Romeo and Juliet that provides a framework for extending the hardness results of Romeo and Juliet in Time on undirected graphs to Romeo and Juliet on DAGs.

It may be an interesting question to figure out if Romeo and Juliet in Time is also EXPTIME-complete as conjectured by Fomin, Golovach, and Thilikos [12]. Moreover, it is known that, assuming ETH, Romeo and Juliet cannot be solved in $n^{o(k)}$ (i.e., $2^{o(k \log n)}$) time [12]. It might be interesting to see if this result can be extended to a lower bound of the form $2^{o(n)}$. Note that this result will be incomparable to the current known bound as k can be $\mathcal{O}(n)$. Aigner and Fromme [1] established that the cop number for planar graphs is at most 3 by the use of a guarding lemma that states that 1 cop can guard the vertices of an isometric path. It is easy to see that the dynamic separation number of planar graphs is unbounded, for eg., consider $K_{2,n}$ and let s, t be the vertices of the partition with two vertices. It might be interesting to figure out the computational complexity of Romeo and Juliet on planar graphs, and more generally, graphs on surfaces.

References

1 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.

2 S. Alpern. The rendezvous search problem. *SIAM Journal on Control and Optimization*, 33(3):673–683, 1995.

3 S. Alpern and S. Gal. *The theory of search games and rendezvous*, volume 55. Springer Science & Business Media, 2006.

4 J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

5 D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdržálek. The dag-width of directed graphs. *Journal of Combinatorial Theory Series B*, 104(4):900–923, 2012.

6 A. Bonato. *An invitation to pursuit-evasion games and graph theory*, volume 97. American Mathematical Society, 2022.

7 A. Bonato and P. Pralat. *Graph searching games and probabilistic methods*. Chapman and Hall/CRC, 2017.

8 P. Bradshaw, S. A. Hosseini, and J. Turcotte. Cops and robbers on directed and undirected abelian Cayley graphs. *European Journal of Combinatorics*, 97, 2021. doi:10.1016/j.ejc.2021.103383.

9 S. Das, H. Gahlawat, U. k. Sahoo, and S. Sen. Cops and robber on some families of oriented graphs. *Theoretical Computer Science*, 888:31–40, 2021.

10 A. Dessmark, P. Fraigniaud, D. R. Kowalski, and A. Pelc. Deterministic rendezvous in graphs. *Algorithmica*, 46:69–96, 2006.

11 F. Fomin, P. Golovach, A. Hall, M. Mihalák, E. Vicari, and P. Widmayer. How to guard a graph? *Algorithmica*, 61(4):839–856, 2011.

12 F. V. Fomin, P. A. Golovach, and D. M. Thilikos. Can romeo and juliet meet? or rendezvous games with adversaries on graphs. *Information and Computation*, 293:105049, 2023.

13 F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

14 F.V. Fomin, P.A. Golovach, and D. Lokshtanov. Guard games on graphs: Keep the intruder out! *Theoretical Computer Science*, 412(46):6484–6497, 2011.

15 T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory Series B*, 82(1):138–154, 2001.

16 W. B. Kinnersley. Cops and robbers is exptime-complete. *Journal of Combinatorial Theory Series B*, 111:201–220, 2015.

17 P. Loh and S. Oh. Cops and robbers on planar directed graphs. *Journal of Graph Theory*, 86(3):329–340, 2017.

18 N. Misra, M. Mulpuri, P. Tale, and G. Viramgami. Romeo and juliet meeting in forest like regions. In *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2022.

19 H. Nagamochi. Cop-robber guarding game with cycle robber-region. *Theoretical Computer Science*, 412:383–390, 2011.

20 R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3):235–239, 1983.

21 A. Quilliot. *Thése d'Etat*. PhD thesis, Université de Paris VI, 1983.

22 R. Šámal, R. Stolař, and T. Valla. Complexity of the cop and robber guarding game. In *Combinatorial Algorithms: 22nd International Workshop, IWOCA 2011, Victoria, BC, Canada, July 20-22, 2011, Revised Selected Papers 22*, pages 361–373. Springer, 2011.

23 R. Šámal and T. Valla. The guarding game is E-complete. *Theoretical Computer Science*, 521:92–106, 2014.

24 A. Ta-Shma and U. Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Transactions on Algorithms (TALG)*, 10(3):1–15, 2014.

25 T. V. Thirumala Reddy, D. Sai Krishna, and C. Pandu Rangan. The guarding problem – complexity and approximation. In *Lecture Notes in Computer Science*, volume 5874, pages 460–470. Springer, 2009.

Minimal Obstructions to C_5-Coloring in Hereditary Graph Classes

Jan Goedgebeur
Department of Computer Science, KU Leuven, Kortrijk, Belgium
Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium

Jorik Jooken
Department of Computer Science, KU Leuven, Kortrijk, Belgium

Karolina Okrasa
Warsaw University of Technology, Poland

Paweł Rzążewski
Warsaw University of Technology, Poland
University of Warsaw, Poland

Oliver Schaudt
University of Cologne, Germany

Abstract

For graphs G and H, an H-coloring of G is an edge-preserving mapping from $V(G)$ to $V(H)$. Note that if H is the triangle, then H-colorings are equivalent to 3-colorings. In this paper we are interested in the case that H is the five-vertex cycle C_5.

A minimal obstruction to C_5-coloring is a graph that does not have a C_5-coloring, but every proper induced subgraph thereof has a C_5-coloring. In this paper we are interested in minimal obstructions to C_5-coloring in F-free graphs, i.e., graphs that exclude some fixed graph F as an induced subgraph. Let P_t denote the path on t vertices, and let $S_{a,b,c}$ denote the graph obtained from paths $P_{a+1}, P_{b+1}, P_{c+1}$ by identifying one of their endvertices.

We show that there is only a finite number of minimal obstructions to C_5-coloring among F-free graphs, where $F \in \{P_8, S_{2,2,1}, S_{3,1,1}\}$ and explicitly determine all such obstructions. This extends the results of Kamiński and Pstrucha [Discr. Appl. Math. 261, 2019] who proved that there is only a finite number of P_7-free minimal obstructions to C_5-coloring, and of Dębski et al. [ISAAC 2022 Proc.] who showed that the triangle is the unique $S_{2,1,1}$-free minimal obstruction to C_5-coloring.

We complement our results with a construction of an infinite family of minimal obstructions to C_5-coloring, which are simultaneously P_{13}-free and $S_{2,2,2}$-free. We also discuss infinite families of F-free minimal obstructions to H-coloring for other graphs H.

2012 ACM Subject Classification Mathematics of computing → Graph enumeration; Mathematics of computing → Graph coloring; Mathematics of computing → Graph algorithms

Keywords and phrases graph homomorphism, critical graphs, hereditary graph classes

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.55

Related Version *Full Version*: `https://arxiv.org/abs/2404.11704` [12]

Supplementary Material
Software: `https://github.com/JorikJooken/ObstructionsForC5Coloring` [22]
archived at `swh:1:dir:3bc6333a056d56efa4e99fd55e2061c149fe8b47`

Funding *Jan Goedgebeur*: Jan Goedgebeur is supported by Internal Funds of KU Leuven and an FWO grant with grant number G0AGX24N.
Jorik Jooken: Jorik Jooken is supported by an FWO grant with grant number 1222524N.
Karolina Okrasa: Karolina Okrasa is supported by a Polish National Science Centre grant no. 2021/41/N/ST6/01507.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 55; pp. 55:1–55:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Paweł Rzążewski: Paweł Rzążewski is supported by a Polish National Science Centre grant no. 2018/31/D/ST6/00062.

Acknowledgements The authors are grateful to Rémi Di Guardia for fruitful and inspiring discussions that led to this project. This project itself was initiated at Dagstuhl Seminar 22481 "Vertex Partitioning in Graphs: From Structure to Algorithms." We are grateful to the organizers and other participants for a productive atmosphere.

1 Introduction

Out of a great number of interesting and elegant graph problems, the notion of graph coloring is, arguably, among the most popular and well-studied ones, not only from combinatorial, but also from algorithmic point of view. For an integer $k \geq 1$, a *k-coloring* of a graph G is a function $c : V(G) \to \{1, \ldots, k\}$ such that for every edge $uv \in E(G)$ it holds that $c(u) \neq c(v)$. For a fixed integer $k \geq 1$, the k-Coloring problem is a computational problem in which an instance is a graph G and we ask whether there exists a k-coloring of G.

The k-Coloring problem is known to be polynomial-time solvable if $k \leq 2$ and NP-complete for larger values of k. Still, even if $k \geq 3$, it is often possible to obtain polynomial-time algorithms that solve k-Coloring if we somehow restrict the class of input instances, for example, to perfect graphs [14, 15], bounded-treewidth graphs [1] or intersection graphs of geometric objects [11].

Observe that these example classes are *hereditary*, i.e., closed under deleting vertices. Such a property is very useful in algorithm design, as it combines well with standard algorithmic techniques, like branching or divide-&-conquer. Therefore, if we want to study some computational problem in a restricted graph class $\mathcal{G}$, choosing $\mathcal{G}$ to be hereditary appears to be reasonable. For a fixed graph F, we say that a graph G is *F-free* if it does not contain F as an induced subgraph. If $\mathcal{F}$ is a family of graphs, we say that a graph G is *$\mathcal{F}$-free* if G is F-free for every $F \in \mathcal{F}$. Each hereditary class of graphs can be characterized by a (possibly infinite) family $\mathcal{F}$ of forbidden induced subgraphs.

Coloring hereditary graph classes. As a first step towards understanding the complexity of k-Coloring in hereditary classes it is natural to consider classes defined by a single induced subgraph F. If F contains a cycle or a vertex of degree at least 3, it follows from the classical results by Emden-Weinert [9], Holyer [20], and Leven and Galil [25] that for every $k \geq 3$, k-Coloring remains NP-complete when restricted to F-free graphs. Thus we are left with the case that F is a *linear forest*, i.e., every component of F is a path.

However, if F is a linear forest, the situation becomes more complicated. For simplicity, let us focus on the case when F is connected, i.e., F is a path on t vertices, denoted by P_t. Then, k-Coloring is polynomial-time-solvable in P_t-free graphs if $t \leq 5$, or if $(k, t) \in \{(3, 6), (3, 7), (4, 6)\}$ [21, 30, 2, 18]. On the other hand, for any $k \geq 4$, the k-Coloring problem is NP-complete in P_t-free graphs for all other values of t [21]. The complexity of the remaining cases, i.e., 3-Coloring of P_t-free graphs where $t \geq 8$, remains unknown: we do not know polynomial-time algorithms nor any hardness proofs. The general belief is that all these cases are in fact tractable, which is supported by the existence of a quasipolynomial-time algorithm for 3-Coloring in P_t-free graphs, for every fixed t [29]. For the summary of the results on the complexity of k-Coloring P_t-free graphs see Figure 1. Let us remark that there are also some results for disconnected forbidden linear forests [24, 6].

k \ t	≤ 4	5	6	7	≥ 8
3	■	■	■	🟩	🟨
4	■	🟩	🟩	🟥	🟥
≥ 5	■	🟩	🟥	🟥	🟥

■ the set of P_t-free minimal obstructions to k-coloring is finite

🟩 the set of P_t-free minimal obstructions to k-coloring is infinite, but k-Coloring P_t-free graphs is polynomial-time solvable

🟨 k-Coloring P_t-free graphs is quasipolynomial-time solvable

🟥 k-Coloring P_t-free graphs is NP-complete

Figure 1 The complexity of k-Coloring P_t-free graphs.

Minimal obstructions to k-coloring. One can look at k-Coloring from another, purely combinatorial perspective. Instead of asking whether a graph G admits a k-coloring, we can ask whether it contains a *dual object*, i.e., some structure that forces the chromatic number to be at least $k + 1$. For example, 2-Coloring can be equivalently expressed as a question whether a graph contains an odd cycle. As another example, k-Coloring restricted to perfect graphs is equivalent to the question of the existence of a $(k + 1)$-clique, i.e., the complete graph on $k + 1$ vertices, denoted by K_{k+1}.

In other words, odd cycles are *minimal non-2-colorable graphs* and $(k + 1)$-cliques are *minimal non-k-colorable perfect graphs* (where minimality is defined with respect to the induced subgraph relation). Formally, if a graph G is not k-colorable, but every proper induced subgraph of it is k-colorable, we say that G is *vertex-$(k + 1)$-critical* or is a *minimal obstruction to k-coloring*. We denote by $\mathsf{Obstructions}(k)$ the set of all minimal obstructions to k-coloring. Note that $\mathsf{Obstructions}(k)$ naturally forms a family of dual objects – a graph is k-colorable if and only if it does not contain any graph from $\mathsf{Obstructions}(k)$ as an induced subgraph.

Suppose that, for some k, there is a polynomial-time algorithm Alg_k that takes as an input a graph G and answers whether it contains any graph from $\mathsf{Obstructions}(k)$ as an induced subgraph (i.e., whether G is *not* $\mathsf{Obstructions}(k)$-free). From the discussion above it follows that the existence of Alg_k yields a polynomial-time algorithm for k-Coloring. Thus it is unlikely that Alg_k exists for any $k \geq 3$. However, it is still possible when we restrict the input graphs to a certain class $\mathcal{G}$ (like perfect graphs in the example above). Recall that we are interested in the case that $\mathcal{G} = F$-free, where F is a path. Let us denote such a restriction Alg_k to F-free graphs by $\mathsf{Alg}_{k,F}$.

Note that the existence of $\mathsf{Alg}_{k,F}$ is trivial if $(\mathsf{Obstructions}(k) \cap F\text{-free})$ is finite; indeed, brute force works in this case. This line of arguments allows us to further refine cases that are polynomial-time solvable: into pairs (k, F), where $(\mathsf{Obstructions}(k) \cap F\text{-free})$ is finite, and the others. Recall that the algorithm for k-Coloring obtained for the former ones is able to produce a *negative certificate*: a small (constant-size) witness that the input graph is *not* k-colorable. We refer the reader to the survey of McConnell et al. [27] for more information about certifying algorithms.

It turns out that we can fully characterize all pairs (k, F) for which $\mathsf{Obstructions}(k) \cap F$-free is finite. It is well-known that P_4-free graphs (also known as *cographs*) are perfect and thus the only minimal obstruction to k-coloring is the $(k + 1)$-clique. Bruce et al. [3] proved that there is a finite number of minimal obstructions to 3-coloring among P_5-free graphs. The result was later extended by Chudnovsky et al. [4] who showed that the family of P_6-free minimal obstructions to 3-coloring is is also finite, and that this is no longer true among P_7-free graphs (and thus for P_t-free graphs for every $t \geq 7$). If the number of colors is larger, things get more difficult faster: Hoàng et al. [19] showed that for each $k \geq 4$ there exists an infinite family of P_5-free minimal obstructions to k-coloring. See also Figure 1.

H-coloring F-free graphs and minimal obstructions to H-colorings. Graph colorings can be seen as a special case of *graph homomorphisms.* For graphs G and H, an *H-coloring* of G is a function $c : V(G) \to V(H)$ such that for every edge $uv \in E(G)$ it holds that $c(u)c(v) \in E(H)$. The graph H is usually called the *target graph.* It is straightforward to verify that homomorphisms from G to the k-clique, are in one-to-one correspondence to k-colorings of G. For this reason one often refers to the vertices of H as *colors.*

For a fixed graph H, by H-Coloring we denote the computational problem that takes as an input a graph G and asks whether G admits a homomorphism to H. The complexity dichotomy for H-Coloring was proven by Hell and Nešetřil [16]: the problem is polynomial-time solvable if H is bipartite, and NP-complete otherwise.

The complexity landscape of H-Coloring in F-free graphs for non-complete target graphs is far from being fully understood. Chudnovsky et al. [5] proved that if H is an odd cycle on at least 5 vertices, then H-Coloring is polynomial-time solvable in P_9-free graphs; they also showed a number of hardness results for more general variants of the homomorphism problem. Feder and Hell [10] and Dębski et al. [8] studied the case when H is an *odd wheel*, i.e., an odd cycle with universal vertex added. The most general algorithmic results were provided by Okrasa and Rzążewski [28] who showed that

(OR1) if H does not contain C_4 as a subgraph, then H-Coloring can be solved in quasi-polynomial time in P_t-free graphs for any fixed t (note that a better running time here would also mean progress for 3-Coloring P_t-free graphs),

(OR2) if H is of girth at least 5, then H-Coloring can be solved in subexponential time in F-free graphs, where F is any fixed *subdivided claw*, i.e., any graph obtained from the three-leaf star by subdividing edges.

While these are not polynomial-time algorithms, no NP-hardness proofs for these cases are known either. To complete the picture, from [28] it also follows that if H is a so-called *projective core* that contains C_4 as a subgraph, then there exists a t such that H-Coloring is NP-complete in P_t-free graphs (and thus also in graphs excluding some fixed subdivided claw). Furthermore, the hardness reductions even exclude any subexponential-time algorithms for these cases, assuming the Exponential Time Hypothesis (ETH). Let us skip the definition of a projective core, as it is quite technical and not really relevant for this paper. However, it is worth pointing out that almost all graphs are projective cores [26, 17].

Since we are interested in a finer classification of polynomial-time-solvable cases, we should be looking at pairs (H, F) of graphs such that the H-Coloring problem is *not* known to be NP-complete in F-free graphs. From the discussion above it follows that there are two natural families of such pairs to consider:

(i) when H does not contain C_4 as a subgraph and F is a path,

(ii) when H is of girth at least 5 and F is a subdivided claw.

It is straightforward to generalize the notion of minimal obstructions to the setting of H-colorings. A graph G is called a *minimal obstruction to H-coloring* if there is no H-coloring of G, but every proper induced subgraph of G can be H-colored.

The area of minimal obstructions to H-coloring is rather unexplored. In the setting of (i), Kamiński and Pstrucha [23] showed that for any $t \geq 5$, there are finitely many minimal obstructions to C_{t-2}-coloring among P_t-free graphs.[1] In particular, the family of P_7-free

[1] While in [23] the authors consider minimality with respect to the *subgraph* relation, it is not hard to observe that bounded number of subgraph-minimal obstructions is equivalent to the bounded number of induced-subgraph-minimal obstructions.

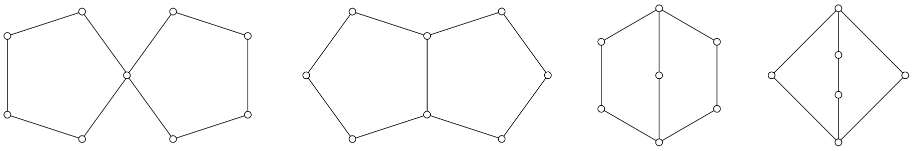


Figure 2 Graphs Q_1, Q_2, Q_3, and Q_4 (left to right).

minimal obstructions to C_5-coloring is finite. In the setting of (ii), Dębski et al. [8] showed that the triangle is the only minimal obstruction to C_5-coloring among graphs that exclude the *fork*, i.e., the graph obtained from the three-leaf star by subdividing one edge once.

Our contribution. As our first result, we show the following strengthening of the result of Kamiński and Pstrucha.

▶ **Theorem 1.** *There are 19 minimal obstructions to C_5-coloring among P_8-free graphs.*

Let us sketch the proof of Theorem 1. Note that K_3 is a minimal obstruction for C_5-coloring, so from now on we focus on graphs that are $\{P_8, K_3\}$-free. For $i \in \{1, 2, 3, 4\}$, by Q_i we denote the graph obtained from two copies of C_5 by identifying an i-vertex subpath of one cycle with an i-vertex subpath of the other one, see Figure 2. In the proof we separately consider minimal obstructions that contain some Q_i as an induced subgraph, and those that are $\{Q_1, Q_2, Q_3, Q_4\}$-free.

The intuition behind this is as follows. Notice that if G contains an induced 5-vertex cycle, the vertices of this cycle must be mapped to the vertices of C_5 bijectively, respecting the ordering along the cycle. Consequently, if G contains some Q_i, the colorings of the vertices in Q_i are somehow restricted. Combining several Q_i's we might impose some contradictory constraints and thus build a graph that is not C_5-colorable. However, as each Q_i already contains quite long induced paths, we might hope that by combining several Q_i we are either forced to create an induced P_8 (if we add only few edges between different Q_i's) or K_3 (if we add too many such edges). Thus the possibilities of building non-C_5-colorable graphs using this approach are somehow limited. It turns out that this intuition is correct: there are 18 graphs that are $\{P_8, K_3\}$-free and contain Q_i, for some $i \in \{1, 2, 3, 4\}$, as an induced subgraph. Together with K_3, they are shown in Figure 3. This part of the proof is computer-aided.

For the second step, we assume that our graph does not contain any Q_i, i.e., we consider graphs that are $\{P_8, K_3, Q_1, Q_2, Q_3, Q_4\}$-free. We show that such graphs are always C_5-colorable. Consequently, each minimal obstruction to C_5-coloring (and, in general, every graph that is not C_5-colorable) was discovered in step 1.

Before we discuss the second result, let us introduce the notation for subdivided claws. For integers $a, b, c \geq 1$, by $S_{a,b,c}$ we denote the graph obtained from the three-leaf star by subdividing each edge, respectively, $a - 1$, $b - 1$, and $c - 1$ times. Equivalently, $S_{a,b,c}$ is obtained from three paths $P_{a+1}, P_{b+1}, P_{c+1}$ by identifying one of their endpoints.

As our second result we show the following extension of the result of Dębski et al. [8].

▶ **Theorem 2.** *There are 3 minimal obstructions to C_5-coloring among $S_{2,2,1}$-free graphs, and 5 minimal obstructions to C_5-coloring among $S_{3,1,1}$-free graphs.*

These graphs are shown in Figure 4. The proof is similar to the proof of Theorem 1. First, we consider minimal obstructions that contain an induced C_5 and, using the computer search, we show that there is only a finite number of them. Then, we show that graphs that

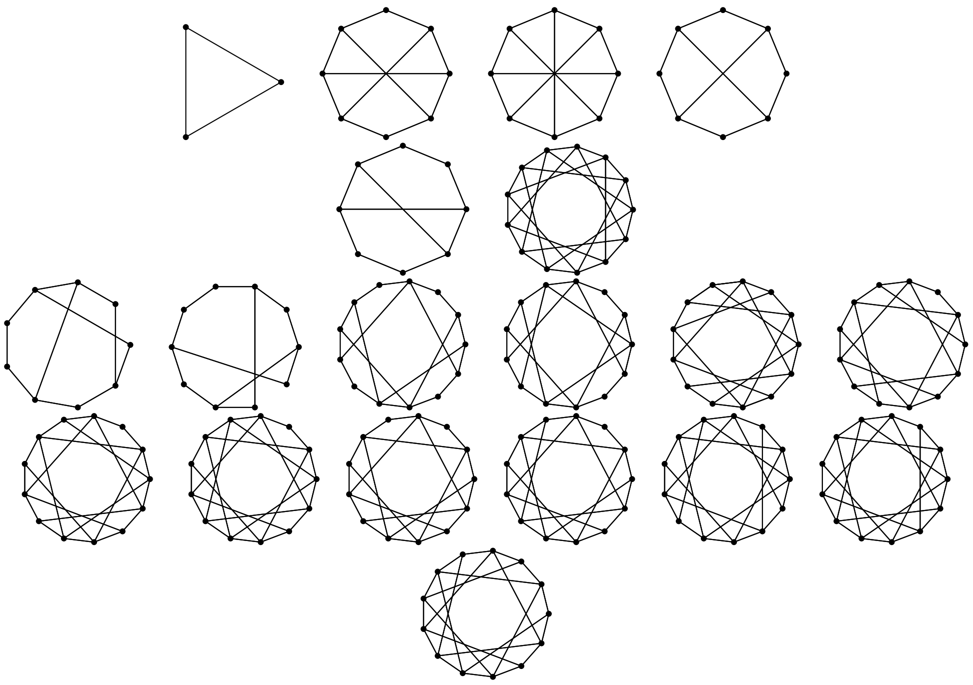


Figure 3 All P_8-free minimal obstructions to C_5-coloring. The graphs in the first row are P_6-free. The graphs in the second row are P_7-free, but not P_6-free. All other graphs are P_8-free, but not P_7-free.

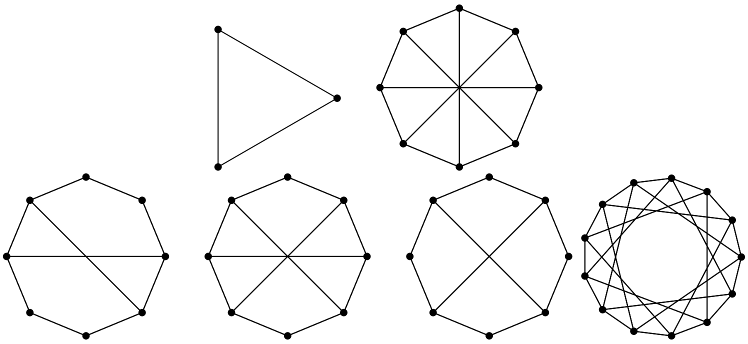


Figure 4 All $S_{2,2,1}$-free and all $S_{3,1,1}$-free minimal obstructions to C_5-coloring. The graphs in the first row are both $S_{2,2,1}$-free and $S_{3,1,1}$-free, whereas the first three graphs in the second row are $S_{3,1,1}$-free, but not $S_{2,2,1}$-free and the last graph in the second row is $S_{2,2,1}$-free, but not $S_{3,1,1}$-free.

exclude K_3 (as it is a minimal obstruction by itself), C_5, and also one of $S_{2,2,1}, S_{3,1,1}$, are either bipartite or are "blown-up cycles" – in both cases C_5-colorability is straightforward to show.

We complement these results with a construction of an infinite family of minimal obstructions to C_5-coloring.

▶ **Theorem 3.** *There is an infinite family of minimal obstructions to C_5-coloring, which simultaneously exclude P_{13}, $S_{2,2,2}$, $S_{5,5,1}$, $S_{11,1,1}$, and $S_{8,2,1}$ as an induced subgraph.*

The construction from Theorem 3 is obtained by generalizing the infinite family of P_7-free minimal obstructions to 3-coloring, provided by Chudnovsky et al. [4]. The idea can be further generalized. Let the *odd girth* of H be the length of a shortest odd cycle in H (and keep it undefined for bipartite graphs).

▶ **Theorem 4.** *Let $q \geq 3$ be odd, and let H be a graph of odd girth q that does not contain C_4 as a subgraph. There is an infinite family of minimal obstructions to H-coloring that are $\{P_{3q-1}, S_{2,2,2}, S_{3(q-1)/2,3(q-1)/2,1}\}$-free.*

Note that Theorem 4 gives a bound for every graph H that was discussed in (OR1) and (OR2). An astute reader might notice that applying Theorem 4 for $H = C_5$, i.e., $q = 5$, yields a family of obstructions that are in particular P_{14}-free and $S_{6,6,1}$-free, which does not match the bounds from Theorem 3. Indeed, obtaining the refined result from Theorem 3 requires some additional work, which again uses a mixture of combinatorial observations and computer search.

Organization of the paper. In Section 2 we introduce some notation and preliminary observations. In Section 3 we explain the algorithm that is later used to generate minimal obstructions. In Sections 4 and 5 we present, respectively, the overview of the proofs of Theorems 1 and 2 In Section 6 we provide constructions of infinite families of graphs that are then used to prove Theorems 3 and 4.

We refer the interested reader to the full version of this paper [12] for the proofs that were omitted due to space limitations (marked by (♠)), and for implementation details of our algorithms and how they were tested for correctness.

2 Preliminaries

For an integer $n \geq 1$ we denote by $[n]$ the set $\{1, \ldots, n\}$, and by $[n]_0$ the set $[n] \cup \{0\}$. For a graph $G = (V, E)$ and a vertex set $U \subseteq V$, the graph $G[U]$ denotes the subgraph of G induced by U. The graph $G - U$ denotes $G[V(G) \setminus U]$. The set $N_G(u)$ denotes the neighborhood of vertex u in the graph G. For $U \subseteq V(G)$ we define $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$. If the graph G is clear from the context, we omit the subscript and write $N(u)$ and $N(U)$.

If there exists an H-coloring of G, we denote this fact by $G \to H$. It is straightforward to verify that if $G \to H$, then $\mathsf{odd\text{-}girth}(G) \geq \mathsf{odd\text{-}girth}(H)$. In particular, K_3 has no C_5-coloring and is actually a minimal obstruction to C_5-coloring. Consequently, every other minimal obstruction to C_5-coloring is K_3-free.

For any two graphs G and H such that G is H-colorable, the graph $\mathsf{hull}(G, H)$ denotes the graph with vertex set $V(G)$ and edge set $\{uv : u, v \in V(G)$ and for every H-coloring c of G, we have $c(u)c(v) \in E(H)\}$. Note that $\mathsf{hull}(G, H)$ is a supergraph of G that is H-colorable and that for every induced subgraph G' of G we have $E(\mathsf{hull}(G', H)) \subseteq E(\mathsf{hull}(G, H))$. Note that $\mathsf{hull}(G, H)$ might contain some induced subgraphs that do not appear in G.

3 Generating F-free minimal obstructions to H-coloring

In this section we describe an algorithm that can be used to generate all F-free minimal obstructions to H-coloring. We emphasize that this approach is robust in the sense that it does not assume that $H = C_5$ and F is a path or a subdivided claw, as needed for Theorems 1 and 2.

Throughout the section graphs H and F are fixed. We will use the term *minimal obstruction* for *minimal obstruction to H-coloring*. The algorithm takes as an input "the current candidate graph" I that it tries to extend to a minimal obstruction by adding a new vertex x and some edges between x and $V(I)$. In particular, the algorithm can be used to generate all F-free minimal obstructions by choosing I as the single-vertex graph.

This algorithm is similar to the algorithm for k-Coloring by [13], but we formulate it in a more general way. In case there are infinitely many minimal obstructions, the generation algorithm will never terminate. If there are only finitely many minimal obstructions, then the algorithm might still not terminate, since the prunning rules might not be strong enough. However, if the algorithm terminates, it is guaranteed that there are only finitely many minimal obstructions and that the algorithm outputs all of them.

Let us explain the algorithm, see also the pseudocode in Algorithm 1.

Algorithm 1 Expand.

```
Constants: target graph $H$, forbidden graph $F$
Input    : current graph $I$
Output   : exhaustive list of $F$-free minimal obstructions to $H$-coloring

if $I$ is $F$-free and not generated before then
    if $I$ is not $H$-colorable then
        if $I$ is a minimal obstruction to $H$-coloring then output $I$
else
    if $I$ contains comparable vertices $(u, v)$ then
        foreach graph $I'$ obtained from $I$ by adding a new vertex $x$ and edges between
          $x$ and vertices in $V(I)$ in all possible ways, such that $ux \in E(I')$, but
          $vx \notin E(\mathsf{hull}(I' - u, H))$ do
            Expand($I'$)
    else if $I$ contains comparable edges $(uv, u'v')$ then
        foreach graph $I'$ obtained from $I$ by adding a new vertex $x$ and edges between
          $x$ and vertices in $V(I)$ in all possible ways, such that $rx \in E(I')$, but
          $r'x \notin E(\mathsf{hull}(I' - \{u, v\}, H)))$ for some $r \in \{u, v\}$ do
            Expand($I'$)
    else
        foreach graph $I'$ obtained from $I$ by adding a new vertex $x$ and edges between
          $x$ and vertices in $V(I)$ in all possible ways do
            Expand($I'$)
```

It starts from a graph I and recursively expands this graph by adding a vertex and edges between this new vertex and existing vertices in each recursive step. The expansion is based on expansion rules that aim at reducing the search space while ensuring that no minimal obstructions are lost in this operation. For example, if an expansion leads to a graph I' that is not F-free, the recursion can be stopped, because all further expansions of I' will not be F-free either (note that we do not add any edges inside $V(I)$ and that the class of F-free graphs is hereditary). Another way to restrict the search space is based on Lemma 5. This lemma and its proof follow Lemma 5 from [4] concerning k-Coloring, but generalizing it to H-Coloring required some adjustments.

▶ **Lemma 5 (♠).** *Let $G = (V, E)$ be a minimal obstruction to H-coloring and let U and W be two non-empty disjoint vertex subsets of G. Let $J := \mathsf{hull}(G - U, H)$. If there exists a homomorphism ϕ from $G[U]$ to $J[W]$, then there exists a vertex $u \in U$ for which $N_G(u) \setminus U \not\subseteq N_J(\phi(u))$.*

As isomorphism is a special type of a homomorphism, Lemma 5 immediately yields the following corollary.

▶ **Corollary 6.** *Let G be an H-colorable graph that is an induced subgraph of a minimal obstruction G'. Let $U, W \subseteq V(G)$ be two non-empty disjoint vertex subsets and let $J := \mathsf{hull}(G - U, H)$. If there exists an isomorphism $\phi : G[U] \to J[W]$ such that $N_G(u) \setminus U \subseteq N_J(\phi(u))$ for all $u \in U$, then there exists a vertex $x \in V(G') \setminus V(G)$ such that x is adjacent to some vertex $u \in U$, but x is not adjacent to $\phi(u)$ in $\mathsf{hull}(G' - U, H)$ (and thus also not adjacent to $\phi(u)$ in $\mathsf{hull}(G'[V(G) \cup \{x\} \setminus U], H)$).*

Actually, we will only use the restricted version of Corollary 6. In what follows we use the notation and assumptions of the Corollary. In case that $|U| = |W| = 1$, say $U = \{u\}$ and $W = \{w\}$, we call the pair (u, w) *comparable vertices*. In case that $G[U]$ and $J[W]$ are both isomorphic to K_2 and, say, $U = \{u, u'\}$ and $W = \{w, w'\}$, we call the pair (uu', ww') *comparable edges*. The algorithm concentrates on finding comparable vertices and edges for computational reasons.

We refer the interested reader to the full version of this paper [12] for additional details about the efficient implementation of this algorithm, independent correctness verifications and sanity checks.

4 Minimal obstructions to C_5-coloring with no long paths

In this section we still only discuss C_5-colorings, thus we will keep writing *minimal obstructions* for *minimal obstructions for C_5-coloring*.

The algorithm from Section 3 was implemented for $H = C_5$ (the source code is made publicly available at [22]). We used the algorithm, combined with some purely combinatorial observations, to generate an exhaustive list of P_t-free minimal obstructions, where $t \leq 8$; see also Figure 3. The minimal obstructions can also be obtained from the database of interesting graphs at the *House of Graphs* [7] by searching for the keywords "minimal obstruction to C5-coloring".

4.1 P_t-free minimal obstructions for $t \in \{6, 7\}$

As a warm-up, let us reprove the result of Kamiński and Pstrucha [23] (in a slightly stronger form, as they did not provide the explicit list of minimal obstructions). It will also serve as a demonstration of the way how the algorithm from 3 is intended to be used.

An exhaustive list for $t \leq 6$ can be obtained by running the algorithm from Section 3 with parameters $(I = K_1, H = C_5, F = P_6)$.[2] This leads to the following observation.

▶ **Observation 7.** *There are four minimal obstructions for C_5-coloring among P_6-free graphs. All of these obstructions, except for the triangle K_3, are P_6-free and not P_5-free.*

Unfortunately, the same simple strategy already fails for $t = 7$. Indeed, the algorithm as presented in Section 3 does not terminate after running for several hours after calling it with parameters $(I = K_1, H = C_5, F = P_7)$. However, with relatively small adjustments, the algorithm is able to produce an exhaustive list of minimal obstructions in a few seconds.

The first adjustment has to do with the order in which the expansion rules are used. Note that the order in which the algorithm checks whether it can find comparable vertices and comparable edges does not affect the correctness of the algorithm, but it might affect

[2] Let us remark that it is a simple exercise to find this list by hand.

whether the algorithm terminates or not. For example, it could happen that by expanding in order to get rid of a pair of comparable vertices, a new pair of comparable vertices is introduced (and this could continue indefinitely). By applying a different expansion rule first, this can be avoided sometimes. For $F = P_7$, the algorithm was run by first looking for comparable vertices and then for comparable edges, except when $|V(I)| = 10$, in which case the algorithm first looks for comparable edges and then for comparable vertices.

The second adjustment is based on the following observation.

▶ **Observation 8 (♠).** *Every P_7-free minimal obstruction to C_5-coloring, except for the graph K_3, contains the cycle C_5 or the cycle C_7 as an induced subgraph.*

By Observation 8, the set of P_7-free minimal obstructions can be partitioned into three subsets:

(i) the triangle K_3,
(ii) minimal obstructions that are P_7-free but contain C_5 as an induced subgraph,
(iii) minimal obstructions that are P_7-free but contain C_7 as an induced subgraph.

Thus, running the algorithm for $(I = C_5, H = C_5, F = P_7)$ and $(I = C_7, H = C_5, F = P_7)$, respectively, we can generate the families (ii) and (iii). This yields the following result; recall that the finiteness of the family of minimal obstructions was already shown by Kamiński and Pstrucha [23].

▶ **Observation 9.** *There are six P_7-free minimal obstructions to C_5-coloring. Two of these obstructions are P_7-free, but not P_6-free.*

4.2 P_8-free minimal obstructions

This section is devoted to the proof of Theorem 1, which we restate below (see also Figure 3).

▶ **Theorem 1.** *There are 19 minimal obstructions to C_5-coloring among P_8-free graphs.*

Similarly to the P_7-free case, the proof uses the algorithm from Section 3, but this time it requires a lot more purely combinatorial insights. For $i \in [4]$, let Q_i be the graph obtained from two disjoint copies of C_5 by identifying i pairs of consecutive corresponding vertices of the cycles (see Figure 2). Let G be a P_8-free minimal obstruction; we aim to understand the structure of G and show that is must be one of 19 graphs in Figure 3. We split the reasoning into two cases: first, we assume that G contains Q_i, for some $i \in [4]$, as an induced subgraph. Then, in the second case, we assume that G is $\{Q_1, Q_2, Q_3, Q_4\}$-free.

Case 1: G contains Q_i for some $i \in [4]$ as an induced subgraph. We deal with this case using the algorithm from Section 3. The algorithm terminates in a few minutes when it is called with the parameters $(I = Q_i, H = C_5, F = P_8)$ for all $i \in [4]$. All minimal obstructions obtained this way are listed in Figure 3.

Case 2: G does not contain Q_i for any $i \in [4]$. As K_3 is a minimal obstruction, from now on we can assume that G is $\{P_8, K_3, Q_1, Q_2, Q_3, Q_4\}$-free. We aim to show that all such graphs are C_5-colorable, i.e., the list obtained in Case 1, plus the triangle, is exhaustive.

▶ **Lemma 10 (♠).** *Let G be a $\{P_8, K_3, Q_1, Q_2, Q_3, Q_4\}$-free graph. Then G is C_5-colorable.*

5 Minimal obstructions to C_5-coloring with no long subdivided claws

This section is devoted to the proof of Theorem 2.

▶ **Theorem 2.** *There are 3 minimal obstructions to C_5-coloring among $S_{2,2,1}$-free graphs, and 5 minimal obstructions to C_5-coloring among $S_{3,1,1}$-free graphs.*

The minimal obstructions are shown in Figure 4 and can also be obtained from the database of interesting graphs at the *House of Graphs* [7] by searching for the keywords “S221-free minimal obstruction to C5-coloring” and “S311-free minimal obstruction to C5-coloring”, respectively. As in this section the target graph is always C_5, we will keep writing *minimal obstructions* for *minimal obstructions to C_5-coloring.* We proceed similarly as in the proof of Theorem 1. Let G be an $S_{2,2,1}$-free (respectively, $S_{3,1,1}$-free) minimal obstruction. We again consider two cases.

Case 1: G contains C_5 as an induced subgraph. This case is solved using the algorithm from Section 3. The algorithm is called with the parameters $(I = C_5, H = C_5, F = S_{2,2,1})$ and then with parameters $(I = C_5, H = C_5, F = S_{3,1,1})$. Both calls terminate, returning a finite list of minimal obstructions.

Case 2: G does not contain C_5 as an induced subgraph. Similarly as in the proof of Theorem 1, note that K_3 is a minimal $\{F, C_5\}$-free obstruction for $F \in \{S_{2,2,1}, S_{3,1,1}\}$. Thus, from now on, we assume that G is K_3-free and prove that there are no more minimal obstructions satisfying this case, i.e., the following result.

▶ **Lemma 11 (♠).** *Let $F \in \{S_{2,2,1}, S_{3,1,1}\}$ and let G be a $\{F, C_5, K_3\}$-free graph. Then G is C_5-colorable.*

Combining the cases, we obtain the statement of Theorem 2.

6 An infinite family of minimal obstructions

In this section we construct infinite families of graphs that will be later used to prove Theorems 3 and 4. The construction is a generalization of the one designed for 3-Coloring [4]; the authors attribute it to Pokrovskiy.

The construction. For every odd $q \geq 3$ and every $p \geq 1$, let $G_{q,p}$ be the graph on vertex set $[qp-3]_0$ (all arithmetic operations on $[qp-3]_0$ here are done modulo $qp-2$), such that for every $i \in [qp-3]_0$ it holds that

$$N(i) = \{i-1, i+1\} \cup \{i + qj - 1 \mid j \in [p-1]\}.$$

To simplify the arguments, we partition $V(G_{q,p})$ into q sets $V_s = \{i \mid i = s \mod q\}$, where $s \in [q-1]_0$. Next observation follows immediately from the definition of $E(G_{q,p})$.

▶ **Observation 12.** *Let $q \geq 3$ be odd, and let $ij \in E(G_{q,p})$ be such that $i \in V_s$ for some $s \in [q-1]_0$. If $i < j$, then $j \in V_{s-1}$, or $j = i+1$, or $i = 0$ and $j = qp-3$. Analogously, if $j > i$, then $j \in V_{s+1}$, or $j = i-1$, or $i = qp-3$ and $j = 0$.*

We now show that graphs $G_{q,p}$ are minimal obstructions to H-coloring for a rich family of graphs H.

▶ **Lemma 13 (♠).** *Let $q \geq 3$ be an odd integer, and let H be graph of odd girth q that does not contain C_4 as a subgraph. For every $p \geq 1$ the graph $G_{q,p}$ is a minimal obstruction to H-coloring.*

Excluded induced subgraphs of $G_{q,p}$. Now let us show an auxiliary lemma that will be helpful in analyzing induced subgraphs that (do not) appear in $G_{q,p}$. In particular, it implies that in order to prove that for each p, the graph $G_{q,p}$ is F-free for some graph F, it is sufficient to show that this statement holds for some small values of p.

▶ **Lemma 14.** *Let q be a fixed constant and F be a graph. Let $p \geq |V(F)| + 2$. If $G_{q,p-1}$ is F-free, then $G_{q,p}$ is F-free.*

Proof. Assume otherwise, and let $U \subseteq V(G_{q,p})$ induce a copy of F in $G_{q,p}$, in particular $|V(F)| = |U|$. Since $|V(F)| < qp-2$, we can assume without loss of generality that the vertex $qp-3$ does not belong to U. Since $|V(G_{q,p})| = qp-2 > q(|V(F)|+1)$, there exist $q+1$ consecutive vertices $\ell, \ldots, \ell+q$ that do not belong to U. Define $R = \{j \in U \mid j > \ell+q\}$, $R' = \{j-q \mid j \in R\}$, and let $L = U \setminus R$.

Now consider $U' = L \cup R'$, and note that $\ell \notin U'$. It is straightforward to verify that $U' \subseteq V(G_{q,p-1})$. We will show that U' induces a copy of F in $V(G_{q,p-1})$. Since this is a contradiction with our assumption, we then conclude that $G_{q,p}$ is F-free.

Let $i, j \in U'$, let $s \in [q-1]_0$ be such that $i \in V_s$. Assume without loss of generality that $i < j$. Note that it is enough to show that

- if $i, j < \ell$, then $ij \in E(G_{q,p-1})$ if and only if $ij \in E(G_{q,p})$,
- if $i, j > \ell$, then $ij \in E(G_{q,p-1})$ if and only if $(i+q)(j+q) \in E(G_{q,p})$,
- if $i < \ell < j$, then $ij \in E(G_{q,p-1})$ if and only if $i(j+q) \in E(G_{q,p})$.

The first item is straightforward.

For the second item, by Observation 12 we have that $ij \in E(G_{q,p-1})$ if and only if $j = i+1$ or $j \in V_{s-1}$. The first is equivalent to $j+q = (i+q)+1$, the latter is equivalent to $j+q \in V_{s-1}$. Hence again using Observation 12 we obtain that $ij \in E(G_{q,p-1})$ if and only if $(i+q)(j+q) \in E(G_{q,p})$.

For the last item, note that $i < \ell < j$ implies $i \in L$ and $j \in R'$. If $ij \in E(G_{q,p-1})$, then by Observation 12, either $j = i+1$ or $v_j \in V_{s-1}$. Since $i < \ell < j$, the first case is not possible. In the second case, if $v_j \in V_{s-1}$, then $j+q \in V_{s-1}$, so $i(j+q) \in E(G_{q,p})$. On the other hand, if $i(j+q) \in E(G_{q,p})$, then, since $i < \ell < j$, it cannot happen that $j+q = i+1$. If $j+q \in V_{s-1}$ then $j \in V_{s-1}$, so we conclude that $ij \in E(G_{q,p-1})$. That concludes the proof. ◀

The power of Lemma 14 is that in order to show that $G_{q,p}$ is F-free for *every* p, it is sufficient to prove it for a finite (and small) set of graphs. This is encapsulated in the following, immediate corollary.

▶ **Corollary 15.** *Let q be a fixed constant and F be a graph. If $G_{q,p}$ is F-free for every $p \leq |V(F)| + 1$, then $G_{q,p}$ is F-free for every p.*

Consequently, for every fixed q and F, Corollary 15 reduces the problem of showing that $G_{q,p}$ is F-free to a constant-size task that can be tackled with a computer.

Proof of Theorem 4 and Theorem 3. Now let us analyze what induced paths and subdivided claws appear in $G_{q,p}$. We start with showing that for every odd $q \geq 3$ and every $p \geq 1$ the graph $G_{q,p}$ is qK_2-free, i.e., they exclude an induced matching on q edges. Here, an induced matching is a set of edges that are not only pairwise disjoint, but also non-adjacent.

▶ **Lemma 16 (♠).** *Let $q \geq 3$ be an odd integer. For every $p \geq 1$ the graph $G_{q,p}$ is qK_2-free.*

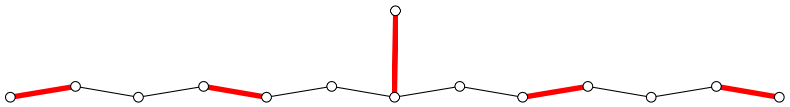


Figure 5 If a graph contains $S_{3(q-1)/2,3(q-1)/2,1}$ as an induced subgraph, then it also contains qK_2.

Let us remark that Lemma 16 is best possible, i.e., if p is large enough, then $G_{q,p}$ contains $(q-1)K_2$ as an induced subgraph. We do not prove it, as later, in Lemma 19, we will show a stronger result. Let us turn our attention to induced paths and subdivided claws that do not appear in $G_{q,p}$. In particular, using Lemma 16 we can exclude the existence of long paths and claws (see Figure 5).

▶ **Lemma 17** (♠). *Let $q \geq 3$ be an odd integer. For every $p \geq 1$ the graph $G_{q,p}$ is $\{P_{3q-1}, S_{2,2,2}, S_{3(q-1)/2,3(q-1)/2,1}\}$-free.*

Now, as an immediate consequence of Lemma 13 and Lemma 17, we obtain Theorem 4, which we restate below.

▶ **Theorem 4.** *Let $q \geq 3$ be odd, and let H be a graph of odd girth q that does not contain C_4 as a subgraph. There is an infinite family of minimal obstructions to H-coloring that are $\{P_{3q-1}, S_{2,2,2}, S_{3(q-1)/2,3(q-1)/2,1}\}$-free.*

Note that for $H = C_5$, i.e., for $q = 5$, Theorem 4 shows that the constructed graphs are in particular P_{14}-free and $S_{6,6,1}$-free. It turns out that they are actually P_{13}-free and $S_{5,5,1}$-free (and also exclude some other subdivided claws). Here we will make use of Corollary 15, combined with computer search.

Let us start with analyzing the length of a longest induced path in $G_{5,p}$. Thus we are interested in applying Corollary 15 to the case $q = 5$ and $F = P_{13}$. Actually, we can even exclude $F = P_{10} + P_2$, i.e., the graph with two components: one isomorphic to P_{10} and the other isomorphic to P_2. Note that $(P_{10} + P_2)$-free graphs are in particular P_{13}-free. Furthermore, the graph $G_{5,p}$ excludes the following subdivided claws: $S_{5,5,1}$, $S_{11,1,1}$, and $S_{8,2,1}$. These results, together with Lemma 13, give us Theorem 3.

▶ **Theorem 3.** *There is an infinite family of minimal obstructions to C_5-coloring, which simultaneously exclude P_{13}, $S_{2,2,2}$, $S_{5,5,1}$, $S_{11,1,1}$, and $S_{8,2,1}$ as an induced subgraph.*

Longest induced paths in $G_{q,p}$. Theorem 3, and the fact that from the result of Chudnovsky et al. [4] it follows that for every p, the graph $G_{3,p}$ is P_7-free (while Lemma 17 only gives P_8-freeness), suggest that the bound on the length of a longest induced path given by Theorem 4 is not optimal also in the other cases. This evidence suggests the following conjecture.

▶ **Conjecture 18.** *Let $q \geq 3$ be odd. For every $p \geq 1$, the graph $G_{q,p}$ is P_{3q-2}-free.*

We conclude this section by showing that the bound from Conjecture 18, if true, is best possible.

▶ **Lemma 19** (♠). *Let $q \geq 3$ be odd. If $p \geq 2q + 1$, then $G_{q,p}$ contains P_{3q-3} as an induced subgraph.*

References

1 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.

2 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. *Comb.*, 38(4):779–801, 2018. doi:10.1007/s00493-017-3553-8.

3 Daniel Bruce, Chính T. Hoàng, and Joe Sawada. A certifying algorithm for 3-colorability of P_5-free graphs. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 594–604. Springer, 2009. doi:10.1007/978-3-642-10631-6_61.

4 Maria Chudnovsky, Jan Goedgebeur, Oliver Schaudt, and Mingxian Zhong. Obstructions for three-coloring graphs without induced paths on six vertices. *J. Comb. Theory, Ser. B*, 140:45–83, 2020. doi:10.1016/j.jctb.2019.04.006.

5 Maria Chudnovsky, Shenwei Huang, Pawel Rzążewski, Sophie Spirkl, and Mingxian Zhong. Complexity of C_k-coloring in hereditary classes of graphs. *Inf. Comput.*, 292:105015, 2023. doi:10.1016/J.IC.2023.105015.

6 Maria Chudnovsky, Shenwei Huang, Sophie Spirkl, and Mingxian Zhong. List 3-Coloring graphs with no induced $P_6 + rP_3$. *Algorithmica*, 83(1):216–251, 2021. doi:10.1007/S00453-020-00754-Y.

7 Kris Coolsaet, Sven D'hondt, and Jan Goedgebeur. House of Graphs 2.0: A database of interesting graphs and more. *Discret. Appl. Math.*, 325:97–107, 2023. doi:10.1016/J.DAM.2022.10.013.

8 Michał Dębski, Zbigniew Lonc, Karolina Okrasa, Marta Piecyk, and Paweł Rzążewski. Computing Homomorphisms in Hereditary Graph Classes: The Peculiar Case of the 5-Wheel and Graphs with No Long Claws. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2022.14.

9 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.

10 Tomás Feder and Pavol Hell. Edge list homomorphisms. Unpublished manuscript, available at http://theory.stanford.edu/~tomas/edgelist.ps.

11 M. R. Garey, David S. Johnson, Gerald L. Miller, and Christos H. Papadimitriou. The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discret. Methods*, 1(2):216–227, 1980. doi:10.1137/0601025.

12 Jan Goedgebeur, Jorik Jooken, Karolina Okrasa, Paweł Rzążewski, and Oliver Schaudt. Minimal obstructions to C_5-coloring in hereditary graph classes. *CoRR*, abs/2404.11704, 2024. arXiv:2404.11704.

13 Jan Goedgebeur and Oliver Schaudt. Exhaustive generation of k-critical H-free graphs. *J. Graph Theory*, 87(2):188–207, 2018. doi:10.1002/JGT.22151.

14 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Comb.*, 1(2):169–197, 1981. doi:10.1007/BF02579273.

15 Martin Grötschel, László Lovász, and Alexander Schrijver. Corrigendum to our paper "The ellipsoid method and its consequences in combinatorial optimization". *Comb.*, 4(4):291–295, 1984. doi:10.1007/BF02579139.

16 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.

17 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discret. Math.*, 109(1-3):117–126, 1992. doi:10.1016/0012-365X(92)90282-K.

18 Chính T. Hoàng, Marcin Kamiński, 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.

19 Chính T. Hoàng, Brian Moore, Daniel Recoskie, Joe Sawada, and Martin Vatshelle. Constructions of k-critical P_5-free graphs. *Discret. Appl. Math.*, 182:91–98, 2015. doi:10.1016/j.dam.2014.06.007.

20 Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10:718–720, 1981.

21 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.

22 Jorik Jooken. Source code of algorithm Expand. https://github.com/JorikJooken/ObstructionsForC5Coloring, 2024.

23 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.

24 Tereza Klimošová, Josef Malík, Tomás Masařík, Jana Novotná, Daniël Paulusma, and Veronika Slívová. Colouring $(P_r + P_s)$-free graphs. *Algorithmica*, 82(7):1833–1858, 2020. doi:10.1007/S00453-020-00675-W.

25 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.

26 Tomasz Łuczak and Jaroslav Nešetřil. Note on projective graphs. *J. Graph Theory*, 47(2):81–86, 2004. doi:10.1002/JGT.20017.

27 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011. doi:10.1016/J.COSREV.2010.09.009.

28 Karolina Okrasa and Paweł Rzążewski. Complexity of the list homomorphism problem in hereditary graph classes. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 54:1–54:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.54.

29 Marcin Pilipczuk, Michał Pilipczuk, and Paweł Rzążewski. Quasi-polynomial-time algorithm for Independent Set in P_t-free graphs via shrinking the space of induced paths. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 204–209. SIAM, 2021. doi:10.1137/1.9781611976496.23.

30 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.

Specification and Automatic Verification of Computational Reductions

Julien Grange
LACL, Université Paris-Est Créteil, France

Fabian Vehlken
Ruhr University Bochum, Germany

Nils Vortmeier
Ruhr University Bochum, Germany

Thomas Zeume
Ruhr University Bochum, Germany

Abstract

We are interested in the following validation problem for computational reductions: for algorithmic problems P and $P^\star$, is a given candidate reduction indeed a reduction from P to $P^\star$? Unsurprisingly, this problem is undecidable even for very restricted classes of reductions. This leads to the question: Is there a natural, expressive class of reductions for which the validation problem can be attacked algorithmically? We answer this question positively by introducing an easy-to-use graphical specification mechanism for computational reductions, called cookbook reductions. We show that cookbook reductions are sufficiently expressive to cover many classical graph reductions and expressive enough so that SAT remains NP-complete (in the presence of a linear order). Surprisingly, the validation problem is decidable for natural and expressive subclasses of cookbook reductions.

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

Keywords and phrases Computational reductions, automatic verification, decidability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.56

Related Version *Full Version*: `https://arxiv.org/abs/2407.04037`

Funding *Fabian Vehlken*: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.
Thomas Zeume: Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.

1 Introduction

Computational reductions are one of the most powerful concepts in theoretical computer science. They are used, among others, to establish undecidability in computability theory and hardness of algorithmic problems in computational complexity theory. In practical applications, reductions help to harness the power of modern SAT solvers for other problems.

For teaching reductions in introductory courses, instructors often design learning tasks for (i) understanding the computational problems involved, (ii) exploring existing reductions via examples, and (iii) designing reductions between computational problems. Technological teaching support so far is only provided for (i) and (ii), likely because these tasks are typically easy to illustrate and checking student solutions is algorithmically straightforward.

Providing teaching support for (iii) requires to address the foundational question: Is there a language for specifying reductions that can express a variety of reductions, but is also algorithmically accessible? In particular, it should be possible to test whether a candidate for a reduction provided by a student is indeed a valid reduction, preferably also providing a counterexample in case a submitted answer is incorrect.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 56; pp. 56:1–56:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we propose such a specification language for reductions and study variants of the following algorithmic problem, parameterized by a class $\mathcal{R}$ of reductions and complexity classes $\mathcal{C}$ and $\mathcal{C}^*$:

Problem: Reduction?$(\mathcal{C}, \mathcal{C}^\star, \mathcal{R})$
Input: Algorithmic problems $P \in \mathcal{C}$, $P^\star \in \mathcal{C}^\star$, and a reduction $\rho \in \mathcal{R}$.
Question: Is ρ a reduction from P to $P^\star$?

More precisely, our contributions are twofold:

- We propose a graphical and modular specification language for reductions, which we call *cookbook reductions* (Section 3). Its design is inspired by "building blocks" such as local replacement of nodes, edges, ... [6] that are used in the context of many standard reductions. Cookbook reductions allow these building blocks to be combined in a simple, stepwise fashion. We compare the expressive power of cookbook reductions with standard methods of specifying reductions. Specifically, we relate cookbook reductions to quantifier-free first-order interpretations (Section 4.2) and observe that SAT remains NP-hard under cookbook reductions, assuming the presence of a linear order (Corollary 3).
- We study variants of the decision problem Reduction?, obtained by choosing different classes of reduction candidates and by either fixing the algorithmic problems $P, P^\star$ or by fixing complexity classes $\mathcal{C}, \mathcal{C}^\star$ and letting $P \in \mathcal{C}, P^\star \in \mathcal{C}^\star$ be part of the input (Section 5). Not surprisingly, Reduction? is undecidable for many restricted variants (Theorem 4). To our surprise, several interesting variants remain decidable: for example, Reduction? is decidable for an arbitrary fixed problem P and fixed $P^\star$ expressible in monadic second-order logic[1], if reduction candidates are from the subclass of cookbook reductions that allows local replacements of edges by a gadget graph (Theorem 10). Also, for some concrete choices of problems $P, P^\star$, we characterize valid reductions; the characterizations can be used to generate counterexamples for invalid candidates, which is particularly relevant in teaching contexts.

Related work. Restricted specification languages have also been used in [3, 9] in the context of learning reductions algorithmically. Reductions that are similar in spirit to cookbook reductions due to their stepwise fashion are pp-constructions and gadget reductions in the realm of (finite) constraint satisfaction problems [1, 5, 2].

2 Preliminaries

We assume familiarity with basic notions from finite model theory [10].

A *(purely relational) schema* $\sigma = \{R_1, \dots, R_m\}$ is a set of relation symbols R_i with associated *arities* $\mathrm{Ar}(R_i)$. A (finite) σ-structure $\mathcal{S} = (U, R_1^{\mathcal{S}}, \dots, R_m^{\mathcal{S}})$ consists of a finite set U, called the *universe* or the *domain* of $\mathcal{S}$, and relations $R_i^{\mathcal{S}} \subseteq U^{\mathrm{Ar}(R_i)}$. If clear from the context, we sometimes omit the superscript $\mathcal{S}$. We also refer to the domain of $\mathcal{S}$ as $\mathrm{dom}(\mathcal{S})$. We write FO_k for the set of all first-order formulas with quantifier depth at most k. The FO_k*-type* of a σ-structure $\mathcal{S}$ is the set of all FO_k formulas over schema σ that $\mathcal{S}$ satisfies. Two structures $\mathcal{S}_1, \mathcal{S}_2$ are FO*-similar* up to quantifier depth k, written $\mathcal{S}_1 \equiv_k^{\mathsf{FO}} \mathcal{S}_2$, if they have the same FO_k-type.

An *isomorphism type* of σ-structures is an equivalence class of the equivalence relation "is isomorphic to". We represent an isomorphism type by an arbitrarily fixed σ-structure $\mathfrak{t}$ with universe $\{1, \dots, k\}$, for the appropriate number k, from that equivalence class. The

[1] This logic extends first-order logic with quantification over sets and can express for example the NP-complete problem 3-Colorability.

Problem: Clique
Input: Undirected graph $G = (V, E)$, $k \in \mathbb{N}$
Question: Is there a set $U \subseteq V$ of size k with $(u, v) \in E$ for all $u, v \in U$?

Problem: IndependentSet
Input: Undirected graph $G = (V, E)$, $k \in \mathbb{N}$
Question: Is there a set $U \subseteq V$ of size k with $(u, v) \notin E$ for all $u, v \in U$?

Problem: VertexCover
Input: Undirected graph $G = (V, E)$, $k \in \mathbb{N}$
Question: Is there a set $U \subseteq V$ of size at most k such that $u \in U$ or $v \in U$ for all $(u, v) \in E$?

Problem: FeedbackVertexSet
Input: Undirected graph $G = (V, E)$, $k \in \mathbb{N}$
Question: Is there a set $U \subseteq V$ of size at most k such that removing U from G yields a cycle-free graph?

Problem: $\textsc{HamCycle}_{\textsc{u}}$
Input: Undirected graph $G = (V, E)$
Question: Is there an undirected cycle in G that passes each node exactly once?

Problem: $\textsc{HamCycle}_{\textsc{d}}$
Input: Directed graph $G = (V, E)$
Question: Is there a directed cycle in G that passes each node exactly once?

Figure 1 Collection of algorithmic problems considered in the paper.

arity of an isomorphism type is the universe size of its representative. Often, we identify an isomorphism type with its representative $\mathfrak{t}$. Given a structure $\mathcal{S}$ and a subset A of its universe, we write $\mathfrak{tp}_{\mathcal{S}}(A)$ for the isomorphism type of $\mathcal{S}[A]$, so, the isomorphism type of the substructure of $\mathcal{S}$ that is induced by A. We write $\mathfrak{tp}(A)$ if $\mathcal{S}$ is clear from the context and call $\mathfrak{tp}(A)$ the isomorphism type of A.

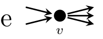
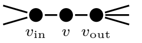
An *embedding* π of a structure $\mathcal{S}$ into a structure $\mathcal{S}^\star$ is an injective mapping from the domain of $\mathcal{S}$ into the domain of $\mathcal{S}^\star$ that is an isomorphism between $\mathcal{S}$ and the substructure of $\mathcal{S}^\star$ that is induced by the image of π. So, an embedding π witnesses that $\mathcal{S}^\star$ contains an isomorphic copy of $\mathcal{S}$ as an induced substructure.

An (*algorithmic*) *problem* P is an isomorphism-closed set of σ-structures, for some schema σ. A *reduction* ρ from a problem P over schema σ to a problem $P^\star$ over schema $\sigma^\star$ is a mapping from σ-structures to $\sigma^\star$-structures such that $\mathcal{S} \in P \Leftrightarrow \rho(\mathcal{S}) \in P^\star$, for every σ-structure $\mathcal{S}$. A *d-dimensional first-order interpretation* from σ-structures to $\sigma^\star$-structures is a tuple $\Psi = (\varphi_U(\bar{x}), \varphi_\sim(\bar{x}_1, \bar{x}_2), (\varphi_R(\bar{x}_1, \ldots, \bar{x}_{\mathrm{Ar}(R)}))_{R \in \sigma^\star})$ of first-order formulas over schema σ, where each tuple $\bar{x} = (x_1, \ldots, x_d), \bar{x}_i = (x_{i,1}, \ldots, x_{i,d})$ consists of d variables. For a given σ-structure $\mathcal{S}$ with universe U, let $\hat{\Psi}(\mathcal{S})$ be the $\sigma^\star$-structure with universe $\hat{U} = \{\bar{a} \in U^d \mid \mathcal{S} \models \varphi_U(\bar{a})\}$ and relations $R^{\hat{\Psi}(\mathcal{S})} = \{(\bar{a}_1, \ldots, \bar{a}_{\mathrm{Ar}(R)}) \in \hat{U}^{\mathrm{Ar}(R)} \mid \mathcal{S} \models \varphi_R(\bar{a}_1, \ldots, \bar{a}_{\mathrm{Ar}(R)})\}$ for each $R \in \sigma^\star$. We demand that for every σ-structure $\mathcal{S}$, the binary relation $\sim^{\hat{\Psi}(\mathcal{S})} = \{(\bar{a}_1, \bar{a}_2) \in \hat{U}^2 \mid \mathcal{S} \models \varphi_\sim(\bar{a}_1, \bar{a}_2)\}$ is a congruence relation on $\hat{\Psi}(\mathcal{S})$, that is, an equivalence relation on the universe that is compatible with the relations of the structure. For a given σ-structure $\mathcal{S}$, the interpretation Ψ defines the $\sigma^\star$-structure $\Psi(\mathcal{S})$ that is the quotient structure of $\hat{\Psi}(\mathcal{S})$ with respect to $\sim^{\hat{\Psi}(\mathcal{S})}$, that is, the structure that results from $\hat{\Psi}(\mathcal{S})$ by restricting the universe to only one element for every equivalence class of $\sim^{\hat{\Psi}(\mathcal{S})}$.

Most of our examples will be drawn from the algorithmic problems from Figure 1. We also consider variants of some of these problems where k is a fixed parameter, e.g. k-Clique asks, given a graph G, whether there is a k-clique in G.

For a natural number n, we sometimes write $[n]$ for the set $\{1, \ldots, n\}$.

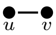
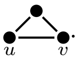
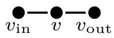
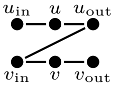
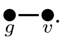
3 Cookbook reductions: A specification language for reductions

When looking for a reduction, one approach by typical experts is to subsequently try building blocks that they have encountered in the context of other reductions before. For example, Garey and Johnson [6, Section 3.2] discuss common proof techniques like local replacements that occur in many standard reductions. An example is the standard reduction from the problem of finding a directed Hamiltonian cycle to finding an undirected Hamiltonian cycle that transforms a directed graph into an undirected graph by mapping each node  to a small gadget . Constructing such node gadgets is one of the typical building blocks when designing reductions.

Our approach towards constructing a specification language for reductions is to (1) identify common building blocks used in computational reductions between graph problems, and to (2) abstract these building blocks into a more general specification language. The resulting language is reasonably broad and, due to its modular and graphical nature, easy to use.

3.1 Building blocks and recipes

Many computational reductions can be crafted from a small set of common building blocks. For reductions between graph problems, some such building blocks are the following:

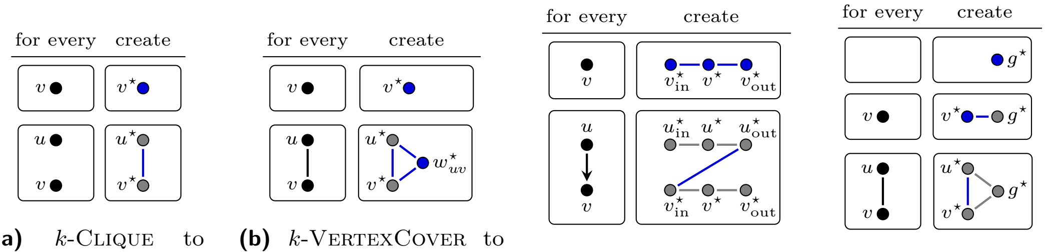
- *Edge gadgets* replace each edge (u, v) of the source instance uniformly by a graph. For example, in the standard reduction from VertexCover to FeedbackVertexSet, every edge  in the source instance is replaced by a triangle .
- *Node gadgets* replace each node of the source instance uniformly by a graph and specify how these graphs are connected. For example, in the standard reduction from HamCycle$_d$ to HamCycle$_u$, every node in the source instance is replaced by a path  and if there is an edge (u, v) in the source instance, then the paths for u and v are connected via .
- *Global gadgets* introduce a (global) graph and specify how each node of this graph is connected to the nodes of the source instance. For example, in the simple reduction from 3-Clique to 4-Clique, a single node is introduced as global graph and each node v of the source instance is connected to g via an edge .

These building blocks have in common that target instances of reductions are obtained from source instances by following simple, recipe-like steps of the form "for every occurrence of a substructure $\mathfrak{t}$ in the source instance, create a copy of the substructure $\mathfrak{t}^\star$ in the target structure". For example, the recipes for the above reductions are as follows:

- Reducing k-VertexCover to k-FeedbackVertexSet: For every node v in the source instance, create a node $v^\star$ in the target instance. For every edge (u, v) in the source instance, create a node $w^\star_{uv}$ and edges $(u^\star, v^\star), (v^\star, w^\star_{uv}), (w^\star_{uv}, u^\star)$ in the target instance.
- Reducing HamCycle$_d$ to HamCycle$_u$: For every node v in the source instance, create nodes $v^\star_{\text{in}}, v^\star, v^\star_{\text{out}}$ in the target instance and connect them as a path. For every directed edge (u, v) in the source instance, create the undirected edge $(u^\star_{\text{out}}, v^\star_{\text{in}})$ in the target instance.
- Reducing 3-Clique to 4-Clique: Create a node $g^\star$ in the target instance. For every node v of the source instance, create a node $v^\star$ in the target instance and add the edge $(v^\star, g^\star)$. Copy all edges (u, v) of the source instance as edges $(u^\star, v^\star)$ to the target instance.

Other reductions can also be phrased in this form, for instance:

- Reducing k-Clique to k-IndependentSet: First, for every node v of the source instance, create a node $v^\star$ in the target instance. Then, for every pair u, v of nodes that are not connected by an edge in the source instance, create an edge $(u^\star, v^\star)$ in the target instance.



(a) k-Clique to k-IndependentSet.

(b) k-VertexCover to k-FeedbackVertexSet.

(c) HamCycle$_d$ to HamCycle$_u$.

(d) 3-Clique to 4-Clique.

Figure 2 Graphical representations of four reductions. The reductions are applied stepwise, from the top-most step to the bottom-most step. Nodes and edges coloured blue are created in this step, grey nodes and edges were created in a previous step.

Reductions specified this way capture building blocks such as the ones from [6] and are usually easy to understand, often much more than their presentation as algorithms or as logical interpretations. Such reductions can also easily be specified graphically, see Figure 2.

3.2 Cookbook reductions: Formalization

We now formalize cookbook reductions as such recipe-style descriptions of computational reductions. In general, graphical representations as in Figure 2 can be used to specify a cookbook reduction. In this section, we discuss the formal syntax and semantics.

Intuitively, a reduction specified in our formalism builds, based on a source structure, the target structure in a sequence of stages, starting from an empty structure. At first, independent of the source structure, some global elements and tuples over these elements may be introduced to the target structure. Then, for every element of the source structure, a set of elements may be added, together with tuples that may also incorporate the elements that were introduced in the step before. The added elements and tuples depend on the (atomic) type of the respective element of the source structure. In further stages, elements are analogously introduced for every set of two, three, . . . , elements of the source structure, depending on the type of these sets.

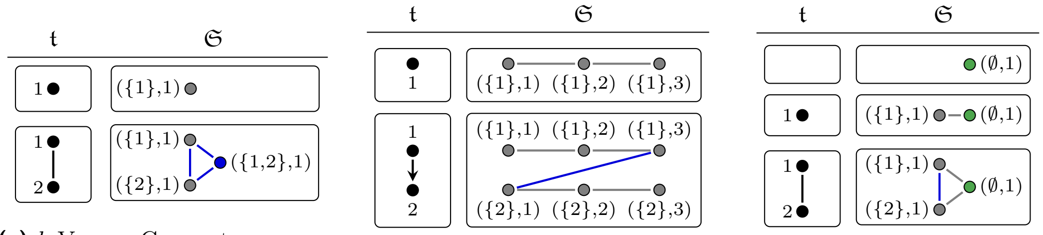
Syntactically, a *cookbook reduction* ρ from σ-structures to $\sigma^\star$-structures is a finite set $\rho = \{(\mathfrak{t}_1, \mathfrak{S}_1), \ldots, (\mathfrak{t}_m, \mathfrak{S}_m)\}$ of pairs which we call *instructions*. The structures $\mathfrak{t}_i$ are σ-structures with universe $\{1, \ldots, k_i\}$, for some natural number $k_i \geq 0$, that represent pairwise distinct isomorphism types of σ-structures. The set $\{\mathfrak{t}_1, \ldots, \mathfrak{t}_m\}$ is the *support* of ρ. The *arity* of ρ is the maximal arity of an isomorphism type in the support of ρ. The structures $\mathfrak{S}_i$ are over the schema $\sigma^\star$. For $(\mathfrak{t}_i, \mathfrak{S}_i) \in \rho$, we also refer to $\mathfrak{S}_i$ as $\mathfrak{S}(\mathfrak{t}_i)$. Each instruction $(\mathfrak{t}, \mathfrak{S})$, where $\mathfrak{t}$ has the universe $[k] = \{1, \ldots, k\}$, satisfies the following properties:

(P1) The universe $\text{dom}(\mathfrak{S})$ of $\mathfrak{S}$ consists of elements (A, j), where $A \subseteq [k]$ and $j \geq 1$. If $(A, j) \in \text{dom}(\mathfrak{S})$ with $j > 1$, then also $(A, 1), \ldots, (A, j-1)$ are in $\text{dom}(\mathfrak{S})$.

(P2) For any $(A, j) \in \text{dom}(\mathfrak{S})$ with $A \subsetneq [k]$, the isomorphism type $\mathfrak{t}' = \mathfrak{tp}_{\mathfrak{t}}(A)$ is in the support of ρ and $(\{1, \ldots, |A|\}, j)$ is in $\text{dom}(\mathfrak{S}(\mathfrak{t}'))$.

(P3) For any tuple $((A_1, j_1), \ldots, (A_\ell, j_\ell))$ in any relation of $\mathfrak{S}$ with $\bigcup_{i \leq \ell} A_i \subsetneq [k]$, the isomorphism type $\mathfrak{tp}_{\mathfrak{t}}(\bigcup_{i \leq \ell} A_i)$ is in the support of ρ.

(P4) For any $(\mathfrak{t}', \mathfrak{S}') \in \rho$ and any $A \subsetneq [k]$ with $\mathfrak{tp}_{\mathfrak{t}}(A) = \mathfrak{t}'$, there is an isomorphism π from $\mathfrak{t}'$ to $\mathfrak{t}[A]$ such that the injective mapping $\hat{\pi}$ with $\hat{\pi}((A', j')) = (\pi(A'), j')$, for all (A', j') in $\text{dom}(\mathfrak{S}')$, is an embedding from $\mathfrak{S}'$ into $\mathfrak{S}$.



(a) k-VertexCover to k-FeedbackVertexSet. **(b)** HamCycle$_d$ to HamCycle$_u$. **(c)** 3-Clique to 4-Clique.

Figure 3 Three reductions formalized as cookbook reductions. Nodes introduced for type $\mathfrak{t}_\emptyset$ are coloured green, nodes and edges introduced for type $\mathfrak{t}_\bullet$ are coloured grey, and nodes and edges introduced for types $\mathfrak{t}_{\bullet-\bullet}$ and $\mathfrak{t}_{\bullet\to\bullet}$ are coloured blue. Compare to Figure 2(b), (c), and (d).

A cookbook reduction has to satisfy a further, semantic property, which we state after defining the semantics.

See Figure 3 for examples of cookbook reductions.

We give some more explanations for the conditions (P1)–(P4). Intuitively, an instruction $(\mathfrak{t}, \mathfrak{S}) \in \rho$ means that for every occurrence of the type $\mathfrak{t}$ in the source structure, a copy of the structure $\mathfrak{S}$ is included in the target structure. The conditions (P1) and (P2) are concerned with the universe $\mathrm{dom}(\mathfrak{S})$ of $\mathfrak{S}$. If $\mathfrak{t}$ is an isomorphism type of k elements, the universe of $\mathfrak{S}$ partly consists of elements $([k], 1), \dots, ([k], m)$, for some number m. These elements are added to the target structure for every occurrence of the type $\mathfrak{t}$. We also call these m elements *fresh* and write $\#_{\text{fresh}}(\mathfrak{t}) = m$ (and $\#_{\text{fresh}}(\mathfrak{t}) = 0$ if no such element exists). The universe of $\mathfrak{S}$ also contains further elements of the form (A, j) with $A \subsetneq [k]$. These represent elements that are added for sets of elements with size $k' < k$ (in the intuitive explanation: in previous stages). If such an element (A, j) occurs in the universe of $\mathfrak{S}$, there has to be a corresponding instruction to add this element, that is, the type $\mathfrak{t}'$ of the set A in $\mathfrak{t}$ has to be in the support of ρ and the element $([k'], j)$ has to be a fresh element in $\mathfrak{S}(\mathfrak{t}')$.

The conditions (P3) and (P4) concern the relations of $\mathfrak{S}$. A tuple $((A_1, j_1), \dots, (A_\ell, j_\ell))$ with $\bigcup_{i \le \ell} A_i = [k]$ in a relation of $\mathfrak{S}$ says that this tuple is to be added to the target structure for every set of elements of type $\mathfrak{t}$. No further conditions on these tuples are imposed by (P3) and (P4). If $A' \overset{\text{def}}{=} \bigcup_{i \le \ell} A_i$ is a proper subset of $[k]$, this tuple is added for the subset A' of elements (intuitively: in a previous stage). Again, there needs to be another instruction that adds this tuple, that is, the isomorphism type $\mathfrak{t}'$ of A' needs to be in the support of ρ.

If a subtype $\mathfrak{t}'$ of $\mathfrak{t}$ is in the support of ρ then the corresponding instruction $(\mathfrak{t}', \mathfrak{S}')$ needs to be respected: for every occurrence of $\mathfrak{t}'$ in $\mathfrak{t}$, a copy of the structure $\mathfrak{S}'$ needs to be present in $\mathfrak{S}$. Formally, if a set $A \subsetneq [k]$ with $|A| = k'$ has type $\mathfrak{t}'$ in $\mathfrak{t}$, as witnessed by some isomorphism π from $\mathfrak{t}'$ to $\mathfrak{t}[A]$, the substructure of $\mathfrak{S}$ that is induced by the set $\{(A_i, j_i) \mid A_i \subseteq \pi([k'])\}$ is isomorphic to $\mathfrak{S}'$.

We now define the semantics of cookbook reductions. A cookbook reduction $\rho = \{(\mathfrak{t}_1, \mathfrak{S}_1), \dots, (\mathfrak{t}_m, \mathfrak{S}_m)\}$ maps a σ-structure $\mathcal{S}$ to a set $\rho(\mathcal{S})$ of $\sigma^\star$-structures, where σ is the schema of the isomorphism types $\mathfrak{t}_i$ and $\sigma^\star$ is the schema of the structures $\mathfrak{S}_i$. For some σ-structure $\mathcal{S}$, the $\sigma^\star$-structure $\mathcal{S}^\star$ is in $\rho(\mathcal{S})$ if the following conditions hold:

(S1) The universe $\mathrm{dom}(\mathcal{S}^\star)$ of $\mathcal{S}^\star$ consists of exactly those elements (A, j) with $A \subseteq \mathrm{dom}(\mathcal{S})$ such that

- the isomorphism type $\mathfrak{t} = \mathfrak{tp}_{\mathcal{S}}(A)$ is in the support of ρ, and
- the structure $\mathfrak{S}$ with $(\mathfrak{t}, \mathfrak{S}) \in \rho$ has the element $(\{1, \dots, |A|\}, j)$ in its universe.

(S2) If a tuple $((A_1, j_1), \dots, (A_\ell, j_\ell))$ is in some relation $R^{\mathcal{S}^\star}$ of $\mathcal{S}^\star$, for any $R \in \sigma^\star$, then the isomorphism type $\mathfrak{tp}_{\mathcal{S}}(\bigcup_{i \le \ell} A_i)$ is in the support of ρ.

(S3) For any $(\mathfrak{t}, \mathfrak{S}) \in \rho$ and any $A \subseteq \text{dom}(\mathcal{S})$ with $\mathfrak{tp}_{\mathcal{S}}(A) = \mathfrak{t}$, there is an isomorphism π from $\mathfrak{t}$ to $\mathcal{S}[A]$ such that the injective mapping $\hat{\pi}$ with $\hat{\pi}((A', j')) = (\pi(A'), j')$, for all (A', j') in the universe of $\mathfrak{S}$, is an embedding from $\mathfrak{S}$ into $\mathcal{S}^\star$.

Intuitively, these conditions state that the elements (S1) and tuples (S3) of $\mathcal{S}^\star$ can be obtained by transforming occurrences of an isomorphism type $\mathfrak{t}$ in $\mathcal{S}$ into $\mathfrak{S}$, for any $(\mathfrak{t}, \mathfrak{S}) \in \rho$, and that no other tuples are present (S2).

A cookbook reduction ρ needs to satisfy the following semantic property[2].

(P5) For every σ-structure $\mathcal{S}$, the set $\rho(\mathcal{S})$ is a non-empty set of isomorphic structures.

Abusing notation, we usually write $\rho(\mathcal{S})$ to denote some arbitrary structure $\mathcal{S}^\star \in \rho(\mathcal{S})$.

4 The expressive power of cookbook reductions

In this section we study the expressive power of cookbook reductions. First, we explain how the building blocks from Section 3 are captured by restricted cookbook reductions. Afterwards, we discuss the expressive power of general cookbook reductions and relate them to quantifier-free first-order interpretations.

4.1 From building blocks to cookbook reductions

Cookbook reductions are a versatile reduction concept and as we have seen in the examples depicted in Figure 2 and Figure 3, many reductions have a small and easily understandable representation as cookbook reductions that have only few isomorphism types in their support.

In fact, the building blocks for graph problems that we discussed as motivation for cookbook reductions can be recovered as restricted variants of cookbook reductions. For undirected graphs with only the binary edge relation E and no self-loops, only four isomorphism types of arity at most 2 are relevant: the type $\mathfrak{t}_\emptyset$ of the graph with 0 nodes, the type $\mathfrak{t}_\bullet$ of a single node, the type $\mathfrak{t}_{\bullet\!-\!\bullet}$ of an undirected edge, and the type $\mathfrak{t}_{\bullet\,\bullet}$ of non-edges.

We obtain the following characterization:

- For a *global gadget reduction*, the inserted global graph $\mathfrak{S}(\mathfrak{t}_\emptyset)$ is arbitrary. Nodes of the source instance are copied, so we fix $\#_{\text{fresh}}(\mathfrak{t}_\bullet) = 1$, but allow $\mathfrak{S}(\mathfrak{t}_\bullet)$ to arbitrarily select nodes from the global graph that are connected to every source node. Edges of the source are copied, so $\#_{\text{fresh}}(\mathfrak{t}_{\bullet\!-\!\bullet}) = 0$ and $\mathfrak{S}(\mathfrak{t}_{\bullet\!-\!\bullet})$ just adds the edge.
- A *node gadget reduction* replaces every node by some gadget, so $\mathfrak{S}(\mathfrak{t}_\bullet)$ is arbitrary. The reduction can define how these gadgets are connected in case there is an edge between the corresponding nodes in the source instance, resulting in $\#_{\text{fresh}}(\mathfrak{t}_{\bullet\!-\!\bullet}) = 0$ and $\mathfrak{S}(\mathfrak{t}_{\bullet\!-\!\bullet})$ being arbitrary apart from that.
- An *edge gadget reduction* replaces edges by some gadget. As every node from the source is copied to the target, $\mathfrak{S}(\mathfrak{t}_\bullet)$ is a single node. We allow any symmetric $\mathfrak{S}(\mathfrak{t}_{\bullet\!-\!\bullet})$.

Only the mentioned isomorphism types are in the support of the cookbook reduction.

A similar characterization holds if the source graph is directed.

Global, node or edge gadget reductions constitute expressive subclasses of cookbook reductions that are relatively easy to comprehend. More fragments can be defined by, e.g., setting an upper bound for $\#_{\text{fresh}}(\mathfrak{t}_\bullet)$ in a node gadget reduction, or selecting a different set

[2] For global and node gadget reductions as introduced in Section 3.1, this property is trivially satisfied, for edge gadget reductions it is satisfied if the gadget graph is symmetric. In general, the following syntactic restriction is necessary: For every $(\mathfrak{t}, \mathfrak{S}) \in \rho$ and any automorphism π of $\mathfrak{t}$ there is an automorphism $\hat{\pi}$ of $\mathfrak{S}$ with $\hat{\pi}((A, j)) = (\pi(A), j')$, for any (A, j) in the universe of $\mathfrak{S}$.

of isomorphism types $\mathfrak{t}$ for which $\mathfrak{S}(\mathfrak{t})$ needs to be provided. This modularity of cookbook reductions helps finding decidable cases of the Reduction? problem. In a teaching context, instructors can select the degree of freedom students have.

4.2 Relating cookbook reductions to quantifier-free interpretations

Quantifier-free first-order (FO) interpretations constitute a widely-used class of reductions with very low complexity, see, e.g., [7]. They are still expressive enough to show hardness of problems: SAT, the satisfiability problem for propositional formulas, is NP-hard even under quantifier-free FO interpretations [4].

In this section, we show that cookbook reductions can be expressed as quantifier-free FO interpretations. If we assume a linear order on the input structures, mildly restricted quantifier-free FO interpretations can be expressed as cookbook reductions. It follows that if input structures are linearly ordered, SAT is NP-hard under cookbook reductions.

We say that two reductions ρ_1 and ρ_2 are *equivalent* for a source structure $\mathcal{S}$ over the appropriate schema, if the target structures $\rho_1(\mathcal{S})$ and $\rho_2(\mathcal{S})$ are isomorphic.

▶ **Theorem 1.** *For every cookbook reduction ρ there is a d-dimensional quantifier-free first-order interpretation Ψ, for some number d, such that ρ and Ψ are equivalent for every structure with at least* 2 *elements.*

Proof idea. Suppose that for a cookbook reduction $\rho = \{(\mathfrak{t}_1, \mathfrak{S}_1), \dots, (\mathfrak{t}_m, \mathfrak{S}_m)\}$ the maximal arity of an isomorphism type $\mathfrak{t}_i$ is k and ℓ is the maximal size of the universe of a structure $\mathfrak{S}_i$. The interpretation Ψ intuitively creates for each set of elements of type $\mathfrak{t}_i$ a copy of the structure $\mathfrak{S}_i$, so, defines a universe of elements of the form (A, i), where $|A| \leq k$ and $i \leq \ell$. Such elements can be encoded by tuples of length $d \overset{\text{def}}{=} k + \ell + 1$. Quantifier-free formulas can determine the isomorphism type of a set of elements and, by the properties of a cookbook reduction, whether a tuple $((A, i_1), \dots, (A, i_r))$ exists in the interpreted structure only depends on the isomorphism type of A. ◀

We call a first-order interpretation *set-respecting* if, for the equivalence relation defined by the formula $\varphi_\sim(\bar{x}_1, \bar{x}_2)$, two tuples $\bar{a}_1, \bar{a}_2$ are only in the same equivalence class if $\bar{a}_1$ and $\bar{a}_2$ contain the same set of elements.

▶ **Theorem 2.** *For every set-respecting quantifier-free first-order interpretation Ψ there is a cookbook reduction ρ such that ρ and Ψ are equivalent for every structure with a linearly ordered universe.*

Proof idea. Let d be the dimension of Ψ. For every isomorphism type $\mathfrak{t}$ of $k \leq d$ elements, the number ℓ of elements $([k], 1), \dots, ([k], \ell)$ in the universe of $\mathfrak{S}(\mathfrak{t})$, so, the number of elements added to the target structure because of a set of elements with isomorphism type $\mathfrak{t}$, is equal to the number of equivalence classes of the congruence defined by $\varphi_\sim$ on the set of d-tuples that contain exactly the k elements of $\mathfrak{t}$ and satisfy the formula φ_U of Ψ. We identify each of the ℓ elements with a particular d-tuple over the set $[k]$, which is possible as $[k]$ is linearly ordered. The structure $\mathfrak{S}(\mathfrak{t})$ is then defined as dictated by Ψ. ◀

As SAT is NP-hard under set-respecting quantifier-free FO interpretations [4], we obtain:

▶ **Corollary 3.** *Assuming that input structures are linearly ordered,* SAT *is* NP*-hard under cookbook reductions.*

Note that in descriptive complexity theory one often studies relational input structures that are not linearly ordered (although Immerman usually assumes a linear order to be present [7, Proviso 1.14]). However, when considering Turing machines as models of computation in complexity theory, inputs are binary string encodings and therefore linearly ordered.

5 Towards automated correctness tests and feedback

We now turn to the problem of checking whether a given reduction candidate is a valid reduction between two computational problems P and $P^\star$. In a first variation of this problem, a corresponding algorithm gets as input the reduction candidate $\rho \in \mathcal{R}$ as well as the two problems $P \in \mathcal{C}$ and $P^\star \in \mathcal{C}^\star$, for a fixed class $\mathcal{R}$ of reductions and fixed complexity classes $\mathcal{C}$ and $\mathcal{C}^\star$. Formally, this corresponds to solving the following algorithmic problem Reduction?$(\mathcal{C}, \mathcal{C}^\star, \mathcal{R})$, parameterized by $\mathcal{C}$, $\mathcal{C}^\star$, and $\mathcal{R}$. Also fixing the problems P and $P^\star$ yields the special case Reduction?$(P, P^\star, \mathcal{R})$.

Problem: Reduction?$(\mathcal{C}, \mathcal{C}^\star, \mathcal{R})$
Input: Algorithmic problems $P \in \mathcal{C}$, $P^\star \in \mathcal{C}^\star$, and a reduction $\rho \in \mathcal{R}$.
Question: Is ρ a reduction from P to $P^\star$?

Problem: Reduction?$(P, P^\star, \mathcal{R})$
Input: A reduction $\rho \in \mathcal{R}$.
Question: Is ρ a reduction from P to $P^\star$?

We are slightly vague here, as for the moment we leave open how algorithmic problems and reductions are represented. It will be clear how these are represented for all classes $\mathcal{C}$, $\mathcal{C}^\star$ and $\mathcal{R}$ we will consider. For standard classes of reductions, – including reductions computable in polynomial time or logarithmic space, as well as first-order definable reductions – already the second, more restricted problem is clearly undecidable for all non-trivial P and $P^\star$. Already testing whether a quantifier-free interpretation or even an edge gadget reduction reduces from some problem P to another problem $P^\star$ is undecidable, for simple P and $P^\star$. As soon as P or $P^\star$ are part of the input, the Reduction? problem is undecidable in most cases in which one of the classes $\mathcal{C}$ or $\mathcal{C}^\star$ is defined by an undecidable fragment of second-order logic, even for very simple classes of reductions.

▶ **Theorem 4.**

1. *Reduction?$(P, P^\star, \mathcal{R})$ is undecidable for the following parameters:*
   - **a.** *The class $\mathcal{R}$ of first-order interpretations, $P = \emptyset$ and arbitrary $P^\star$ (or vice versa, i.e. arbitrary P and $P^\star = \emptyset$).*
   - **b.** *The class $\mathcal{R}$ of edge gadget reductions, $P = \emptyset$ and some graph problem $P^\star$ definable in first-order logic with arithmetic.*
   - **c.** *The class $\mathcal{R}$ of quantifier-free interpretations, $P = \emptyset$ and the graph problem $P^\star$ defined by the first-order formula $\varphi^\star \overset{def}{=} \forall x \exists y E(x, y)$.*
2. *Reduction?$(\mathcal{C}, \mathcal{C}^\star, \mathcal{R})$ is undecidable for the following parameters:*
   - **a.** *A class $\mathcal{R}$ containing the identity mapping, a class $\mathcal{C}$ containing the empty problem, and a class $\mathcal{C}^\star$ defined by a fragment of second-order logic with undecidable finite satisfiability problem.*
   - **b.** *A class $\mathcal{R}$ containing the identity mapping, a class $\mathcal{C}$ defined by a fragment of second-order logic with undecidable finite satisfiability problem, and a class $\mathcal{C}^\star$ containing the empty problem.*

In the rest of this section, we explore how to overcome the undecidability barriers. That is, we explore for which parameters one can obtain algorithms for solving Reduction?$(P, P^\star, \mathcal{R})$ and Reduction?$(\mathcal{C}, \mathcal{C}^\star, \mathcal{R})$. Our focus is on (restrictions of) cookbook reductions.

We start by exhibiting toy examples for algorithms for Reduction?$(P, P^\star, \mathcal{R})$ for concrete algorithmic problems P and $P^\star$ in Section 5.1. For these examples, counterexamples can be provided if the input is not a correct reduction. A generalized view is taken in Section 5.2, where we exhibit algorithm templates for Reduction?$(P, P^\star, \mathcal{R})$ for algorithmic problems P and $P^\star$ selected from classes of problems. Then, in Section 5.3, we consider algorithmic problems as part of the input by studying Reduction?$(\mathcal{C}, \mathcal{C}^\star, \mathcal{R})$.

5.1 Warm-up: Reductions between explicit algorithmic problems

In this section we provide toy examples of how Reduction?$(P, P^\star, \mathcal{R})$ can be decided for very restricted classes $\mathcal{R}$: (1) for reducing k-Clique to ℓ-Clique via global gadgets, for $k < \ell$, (2) for reducing k-VertexCover to k-FeedbackVertexSet via edge gadgets, and (3) for reducing HamCycle$_d$ to HamCycle$_u$ via restricted node gadgets. In all cases, the decision procedures are obtained by characterizing the class of correct gadgets.

While not deep, these characterizations and the algorithms resulting from them are a first step towards more general results.

We start by characterizing those global gadgets that reduce k-Clique to ℓ-Clique. For simplicity, we represent global gadget reductions ρ by a global gadget $\mathfrak{g}_\rho$ and a distinguished subset A of its nodes. When applying ρ to a graph $G = (V, E)$, the gadget $\mathfrak{g}_\rho$ is disjointly added to G and edges (u, v) are introduced for all $u \in A$ and all $v \in V$.

▶ **Proposition 5.** *Let ρ be a global gadget reduction with global gadget $\mathfrak{g}_\rho$ and a distinguished subset A of its nodes. Let $k, \ell \in \mathbb{N}$ with $k < \ell$. Then the following are equivalent:*

1. *ρ is a reduction from k-Clique to ℓ-Clique*
2. *$\mathfrak{g}_\rho$ and A satisfy the following conditions:*
   a. *$\mathfrak{g}_\rho$ has no ℓ-clique*
   b. *$\mathfrak{g}_\rho$ has an $(\ell - k)$-clique contained in A*
   c. *$\mathfrak{g}_\rho$ has no $(\ell - k + 1)$-clique contained in A*

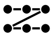
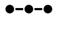
Furthermore, if ρ is not a reduction from k-Clique to ℓ-Clique, then a counterexample can be computed efficiently.

We next characterize those edge gadgets that constitute a reduction from k-VertexCover to k-FeedbackVertexSet. We represent edge gadget reductions ρ by an edge gadget $\mathfrak{g}_\rho$ with two distinguished nodes c and d. When applying ρ to a graph $G = (V, E)$, all edges $(u, v) \in E$ are replaced by disjoint copies of $\mathfrak{g}_\rho$, where u, v are identified with c, d, respectively.

▶ **Proposition 6.** *Let ρ be an edge gadget reduction based on the edge gadget $\mathfrak{g}_\rho$ with distinguished nodes c and d. Then the following are equivalent:*

1. *ρ is a reduction from k-VertexCover to k-FeedbackVertexSet*
2. *$\mathfrak{g}_\rho$ satisfies the following conditions:*
   a. *$\{c\}$ and $\{d\}$ are feedback vertex sets of $\mathfrak{g}_\rho$*
   b. *$\emptyset$ is not a feedback vertex set of $\mathfrak{g}_\rho$.*

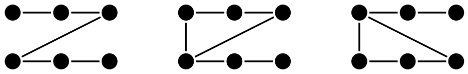
Furthermore, if ρ is not a reduction from k-VertexCover to k-FeedbackVertexSet, then a counterexample can be computed efficiently.

Lastly, we characterize restricted node gadget reductions from the directed Hamiltonian cycle problem HamCycle$_d$ to the undirected variant HamCycle$_u$. For simplicity, we represent node gadget reductions ρ by node gadgets $\mathfrak{g}_\rho$. A node gadget $\mathfrak{g}_\rho$ consists of two copies of a *node graph* $\mathfrak{S}(\mathfrak{t}_\bullet)$ and a set of additional edges between these copies. As an example, the standard reduction from HamCycle$_d$ to HamCycle$_u$ is represented by the node gadget  consisting of two copies of the node graph  with one additional edge between them (cf. Figures 2(c) and 3(b)). When applying $\mathfrak{g}_\rho$ to a graph $G = (V, E)$, all nodes in V are replaced by a copy of the node graph and two such copies for nodes u, v are connected accordingly by the additional set of edges, if $(u, v) \in E$.

As a first step towards characterizing node gadget reductions between HamCycle$_d$ and HamCycle$_u$, we characterize all correct node gadget reductions whose node graph has at most three nodes.

▶ **Proposition 7.** *Let ρ be a node gadget reduction with node gadget $\mathfrak{g}_\rho$ whose node graph has at most three nodes. Then the following are equivalent:*

1. *ρ is a reduction from* HamCycle$_d$ *to* HamCycle$_u$
2. *$\mathfrak{g}_\rho$ is either of the following node gadgets (with the two copies of the node graphs depicted at top and bottom), up to symmetries:*



Furthermore, if ρ is not a reduction from HamCycle$_d$ *to* HamCycle$_u$*, a counterexample can be computed efficiently.*

5.2 Decidable cases for classes of (fixed) algorithmic problems

In this section, we study the question whether there are classes $\mathcal{C}$ and $\mathcal{C}^\star$ of algorithmic problems as well as classes $\mathcal{R}$ of reductions, such that after fixing $P \in \mathcal{C}$ and $P^\star \in \mathcal{C}^\star$ there is an algorithm that tests correctness of inputs $\rho \in \mathcal{R}$.

We first give an example that decidability results are possible for non-trivial classes of reductions and problems. Afterwards, we sketch how the technique employed in the proof can be generalized.

▶ **Theorem 8.** *Reduction?($P, P^\star, \mathcal{R}$) is decidable for the class $\mathcal{R}$ of cookbook reductions with arity bounded by some $r > 0$, arbitrary P, and $P^\star$ definable in first-order logic.*

The proof idea is to represent cookbook reductions ρ by "recipe structures" recipe(ρ) such that $\rho(\mathcal{A})$ can be constructed from the disjoint union $\mathcal{A} \uplus \text{recipe}(\rho)$ of $\mathcal{A}$ and recipe(ρ) via an FO-interpretation which depends on the arity and schema of ρ, but is independent of ρ itself. Then we prove that correctness of reductions in the setting of Theorem 8 only depends on the FO-similarity type of their recipe.

Intuitively, the recipe of a cookbook reduction ρ is the disjoint union of the structures $\mathfrak{S}(\mathfrak{t})$ for all relevant isomorphism types $\mathfrak{t}$, where additional unary relations indicate the source structure and an additional binary relation identifies inherited elements (those (A, j) where A is a strict subset of the domain of $\mathfrak{t}$) with their origin. Formally, fix two schemas σ and $\sigma^\star$, an arity $r \in \mathbb{N}$, and define $\mathfrak{T}_{\leq r}$ to be the finite set of all isomorphism types $\mathfrak{t}$ over the schema σ of arity at most r. The *recipe* recipe(ρ) of a cookbook reduction ρ of arity at most r from σ to $\sigma^\star$ is a structure over the schema $\sigma^\star \cup \{\approx\} \cup \{C_\mathfrak{t} \mid \mathfrak{t} \in \mathfrak{T}_{\leq r}\}$, where $\approx$ is binary and all $C_\mathfrak{t}$ are unary. The restriction of recipe(ρ) to the schema $\sigma^\star \cup \{C_\mathfrak{t} \mid \mathfrak{t} \in \mathfrak{T}_{\leq r}\}$ is the disjoint union $\biguplus_{\mathfrak{t} \in \mathfrak{T}_{\leq r}} \mathfrak{S}(\mathfrak{t})$, where we set $\mathfrak{S}(\mathfrak{t}) = \rho(\mathfrak{t})$ if $\mathfrak{t}$ is not in the support of ρ, and each $C_\mathfrak{t}$ is interpreted as the universe of $\mathfrak{S}(\mathfrak{t})$. The relation $\approx$ "identifies" inherited elements and their original version: for every $\mathfrak{t}, \mathfrak{t}' \in \mathfrak{T}_{\leq r}$ such that $\mathfrak{t}$ is the type of a strict subset of the elements of $\mathfrak{t}'$, if a' is an element of $\mathfrak{S}(\mathfrak{t}')$ inherited from $\mathfrak{S}(\mathfrak{t})$'s element a, then $a' \approx a$ holds in recipe(ρ).

The structure recipe(ρ) representing the cookbook reduction ρ from 3-Clique to 4-Clique given in Figure 3 can be found in Figure 4.

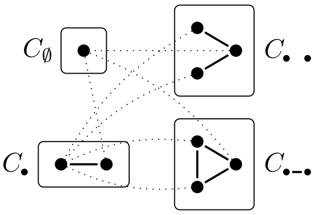


Figure 4 The recipe recipe(ρ) for the cookbook reduction of arity 2 from 3-Clique to 4-Clique from Figure 3. There are four unary relations for the types $\mathfrak{t}_{\emptyset}$, $\mathfrak{t}_{\bullet}$, $\mathfrak{t}_{\bullet\!-\!\bullet}$, and $\mathfrak{t}_{\bullet\;\bullet}$ of loopless undirected graphs. The dotted edges represent the binary inheritance relation $\approx$.

There is an FO-interpretation that applies a recipe recipe(ρ) to a structure $\mathcal{A}$ by interpreting $\mathcal{A} \uplus \text{recipe}(\rho)$.

▶ **Lemma 9.** *Fix $r > 0$ and two schemas $\sigma, \sigma^\star$. There is an* FO*-interpretation $\mathcal{I}^r_{\sigma,\sigma^\star}$ such that $\rho(\mathcal{A})$ and $\mathcal{I}^r_{\sigma,\sigma^\star}(\mathcal{A} \uplus \textit{recipe}(\rho))$ are isomorphic, for every cookbook reduction ρ from σ to $\sigma^\star$ of arity at most r and for every σ-structure $\mathcal{A}$.*

As FO-interpretations preserve FO-similarity, there is a function $f^r_{\sigma,\sigma^\star} : \mathbb{N} \to \mathbb{N}$ such that for every $k \in \mathbb{N}$, $\mathcal{A} \equiv^{\mathsf{FO}}_{f^r_{\sigma,\sigma^\star}(k)} \mathcal{A}'$ entails $\mathcal{I}^r_{\sigma,\sigma^\star}(\mathcal{A}) \equiv^{\mathsf{FO}}_k \mathcal{I}^r_{\sigma,\sigma^\star}(\mathcal{A}')$ (see, e.g., [8, Section 3.2]).

We now prove Theorem 8.

Proof of Theorem 8. We show that whether a cookbook reduction ρ is a reduction from P to $P^\star$ solely depends on the FO_m-type of recipe(ρ), for some large enough m that depends only on r, P, and $P^\star$. As there are only finitely many such FO_m-types and because the type of recipe(ρ) can be determined, the statement follows.

Let k be the quantifier rank of a formula $\varphi^\star \in \mathsf{FO}$ defining $P^\star$. If the recipes of two reductions ρ and ρ' of arity at most r are $f^r_{\sigma,\sigma^\star}$-similar, then so are $\mathcal{A} \uplus \text{recipe}(\rho)$ and $\mathcal{A} \uplus \text{recipe}(\rho')$ for all σ-structures $\mathcal{A}$ (due to a simple Ehrenfeucht-Fraïsse argument). But then $\mathcal{I}^r_{\sigma,\sigma^\star}(\mathcal{A} \uplus \text{recipe}(\rho))$ and $\mathcal{I}^r_{\sigma,\sigma^\star}(\mathcal{A} \uplus \text{recipe}(\rho'))$ – and therefore also $\rho(\mathcal{A})$ and $\rho'(\mathcal{A})$ –, are k-similar. In particular, the reductions ρ and ρ' behave in the same way for all σ-structures $\mathcal{A}$, that is $\rho(\mathcal{A}) \models \varphi^\star$ if and only if $\rho'(\mathcal{A}) \models \varphi^\star$.

We conclude that whether $\rho(\mathcal{A})$ satisfies $\varphi^\star$ only depends on the $\mathsf{FO}_{f^r_{\sigma,\sigma^\star}(k)}$-type of recipe($\rho$) for all $\mathcal{A}$. Hence, the recipe of positive instances of Reduction?($P, P^\star, \mathcal{R}$) is a union of equivalence classes for $\equiv^{\mathsf{FO}}_{f^r_{\sigma,\sigma^\star}(k)}$. For a reduction ρ it can now be evaluated whether its recipe satisfies the type of one of these equivalence classes. ◀

In the rest of this section, we explore how the technique used in the proof above can be generalized to logics beyond FO. Our focus is on monadic-second order logic (MSO), which extends FO by quantifiers for sets of elements. One of the key ingredients, that FO-interpretations preserve FO-similarity, does not translate to MSO for interpretations of dimension greater than one (not even for quantifier-free interpretations). Yet, decidability is retained for problems $P^\star \in \mathsf{MSO}$ if we restrict ourselves to *edge gadget reductions* (on graphs), instead of general cookbook reductions. This generalizes Proposition 6.

▶ **Theorem 10.** *Reduction?($P, P^\star, \mathcal{R}$) is decidable for the class $\mathcal{R}$ of edge gadget reductions, arbitrary P, and $P^\star$ definable in monadic second-order logic.*

The proof exploits compositionality of MSO and can be generalized to other subclasses of cookbook reductions. A discussion of such subclasses is postponed to the long version of this paper.

Proof sketch. An edge gadget reduction ρ is specified as a graph $\mathfrak{g}_\rho$, with two distinguished nodes. As in the proof of Theorem 8, the idea is to show that there is an integer m such that whether ρ is a reduction from P to $P^\star$ only depends on the MSO_m-type of $\mathfrak{g}_\rho$. More precisely, for all gadget graphs $\mathfrak{g}_\rho$ and $\mathfrak{g}_{\rho'}$ with $\mathfrak{g}_\rho \equiv_m^{\mathsf{MSO}} \mathfrak{g}_{\rho'}$, one proves that $\rho(G) \equiv_k^{\mathsf{MSO}} \rho'(G)$ for all graphs G, where k is the quantifier rank of an MSO-sentence describing $P^\star$.

For proving MSO_k-similarity of $\rho(G)$ and $\rho'(G)$, one can use Ehrenfeucht-Fraïssé games for MSO (see, e.g., [10, Section 7.2]). The graphs $\rho(G)$ and $\rho'(G)$ are a composition of G with the edge gadgets $\mathfrak{g}_\rho$ and $\mathfrak{g}_{\rho'}$, respectively. Duplicator has a winning strategy for the MSO-game played on (G, G) as well as for the MSO-game played on $(\mathfrak{g}_\rho, \mathfrak{g}_{\rho'})$. Her strategy for the game on $\rho(G)$ and $\rho'(G)$ is to combine these two winning strategies. For instance, if Spoiler moves on $\rho(G)$ and part of his move is on the edge gadget inserted for an edge (u, v) of G, then Duplicator's response for this part of the move is derived from her strategy for the game on $(\mathfrak{g}_\rho, \mathfrak{g}_{\rho'})$. The partial answers for individual edges are then combined. ◀

For both FO and MSO, the proof uses that the respective classes of reductions can be finitely partitioned into similarity classes and that all reductions in one class are either correct or not correct. This provides a basis for characterizations akin to the ones in Section 5.1 for concrete, arbitrary problems P and concrete $P^\star$ definable in FO or MSO.

5.3 Algorithmic problems as input: decidable cases

We now explore decidability when source and/or target problems are part of the input. We consider classes $\mathcal{C}$ and $\mathcal{C}^\star$ captured by logics $\mathcal{L}$ and $\mathcal{L}^\star$, respectively, and write, e.g., Reduction?$(\mathcal{L}, \mathcal{L}^\star, \mathcal{R})$ for the algorithmic problem where we ask, given $\varphi \in \mathcal{L}$, $\varphi^\star \in \mathcal{L}^\star$ and $\rho \in \mathcal{R}$, whether ρ is a reduction from the problem defined by φ to the one defined by $\varphi^\star$.

One approach for obtaining decidability for the problem Reduction?$(\mathcal{L}, \mathcal{L}^\star, \mathcal{R})$ is by restating it as a satisfiability question for a decidable logic. For a quantifier-free interpretation $\mathcal{I}$ from σ-structures to $\sigma^\star$-structures, denote by $\mathcal{I}^{-1}(\varphi^\star)$ the σ-formula obtained from a $\sigma^\star$-formula $\varphi^\star$ by replacing atoms in $\varphi^\star$ according to their definition in $\mathcal{I}$. Whether a quantifier-free interpretation $\mathcal{I}$ is a reduction from the algorithmic problem defined by $\varphi \in \mathcal{L}$ to the one defined by $\varphi^\star \in \mathcal{L}^\star$ is equivalent to whether $\mathcal{A} \models \varphi$ if and only if $\mathcal{I}(\mathcal{A}) \models \varphi^\star$, for all structures $\mathcal{A}$. This in turn is equivalent to checking whether $\varphi \leftrightarrow \mathcal{I}^{-1}(\varphi^\star)$ is a tautology.

These observations yield, for instance, the following decidable variants, some involving the class QF of quantifier-free first-order interpretations, a class that includes all cookbook reductions, see Theorem 1. See the full version for the proof.

▶ **Theorem 11.**

1. *Reduction?*$(\exists^*\mathsf{FO}, \exists^*\mathsf{FO}, QF)$ *is decidable.*
2. *Reduction?*$(P, \exists^*\mathsf{FO}, QF)$ *is decidable for every fixed algorithmic problem* P*.*
3. *Reduction?*$(\exists^*\mathsf{FO}, P^\star, \mathcal{R})$ *is decidable for every fixed algorithmic problem* $P^\star$ *definable in* MSO *and the class* $\mathcal{R}$ *of edge gadget reductions.*

6 Summary and discussion

We studied variants of the algorithmic problem Reduction? which asks whether a given mapping is a computational reduction between two algorithmic problems. In addition to studying this problem for standard classes of reductions, we also proposed a graphical and compositional language for computational reductions, called cookbook reductions, and compared their expressive power to quantifier-free first-order interpretations. While Reduction? is undecidable in many restricted settings, we identified multiple decidable cases involving (restricted) cookbook reductions and quantifier-free first-order interpretations. Due to its graphical and compositional nature, cookbook reductions are well-suited to be used in teaching support systems for learning tasks tackling the design of computational reductions.

A prototype[3] of our formal framework has been integrated into the teaching support system *Iltis* [11]. Recently it has been used in introductory courses *Theoretical Computer Science* with > 300 students at Ruhr University Bochum and TU Dortmund in workflows covering (i) understanding computational problems, (ii) exploring reductions via examples, and (iii) designing reductions.

References

1 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223:363–398, 2018.

2 Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. Lecture Notes in Logic. Cambridge University Press, 2021.

3 Michael S. Crouch, Neil Immerman, and J. Eliot B. Moss. Finding reductions automatically. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2010. doi:10.1007/978-3-642-15025-8_10.

4 Elias Dahlhaus. Reduction to NP-complete problems by interpretations. In Egon Börger, Gisbert Hasenjaeger, and Dieter Rödding, editors, *Logic and Machines: Decision Problems and Complexity, Proceedings of the Symposium "Rekursive Kombinatorik" held from May 23-28, 1983 at the Institut für Mathematische Logik und Grundlagenforschung der Universität Münster/Westfalen*, volume 171 of *Lecture Notes in Computer Science*, pages 357–365. Springer, 1983. doi:10.1007/3-540-13331-3_51.

5 Victor Dalmau and Jakub Oprsal. Local consistency as a reduction between constraint satisfaction problems. *CoRR*, abs/2301.05084, 2023. doi:10.48550/arXiv.2301.05084.

6 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

7 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987. doi:10.1137/0216051.

8 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

9 Charles Jordan and Lukasz Kaiser. Experiments with reduction finding. 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 192–207. Springer, 2013. doi:10.1007/978-3-642-39071-5_15.

10 Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004.

11 Marko Schmellenkamp, Fabian Vehlken, and Thomas Zeume. Teaching formal foundations of computer science with Iltis. *Educational Column of the Bulletin of EATCS*, 2024. URL: http://bulletin.eatcs.org/index.php/beatcs/article/download/797/842.

[3] See https://iltis.cs.tu-dortmund.de/computational-reductions

Higher-Order Constrained Dependency Pairs for (Universal) Computability

Liye Guo
Radboud University, Nijmegen, The Netherlands

Kasper Hagens
Radboud University, Nijmegen, The Netherlands

Cynthia Kop
Radboud University, Nijmegen, The Netherlands

Deivid Vale
Radboud University, Nijmegen, The Netherlands

Abstract

Dependency pairs constitute a series of very effective techniques for the termination analysis of term rewriting systems. In this paper, we adapt the static dependency pair framework to logically constrained simply-typed term rewriting systems (LCSTRSs), a higher-order formalism with logical constraints built in. We also propose the concept of universal computability, which enables a form of open-world termination analysis through the use of static dependency pairs.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Higher-order term rewriting, constrained rewriting, dependency pairs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.57

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2406.19379 [11]

Supplementary Material *Software (Source Code)*: https://github.com/hezzel/cora [20]
Software (Source Code): https://zenodo.org/records/12551027 [21]

Funding The authors are supported by NWO VI.Vidi.193.075, project "CHORPE".

1 Introduction

Logically constrained simply-typed term rewriting systems (LCSTRSs) [12] are a formalism of higher-order term rewriting with logical constraints (built on its first-order counterpart [19]). Proposed for program analysis, LCSTRSs offer a flexible representation of programs owing to – in contrast with traditional TRSs – their support for primitive data types such as (arbitrary-precision or fixed-width) integers and floating-point numbers. Without compromising the ability to directly reason about these widely used data types, LCSTRSs bridge the gap between the abundant techniques based on term rewriting and automatic program analysis.

We consider *termination* analysis in this paper. The termination of LCSTRSs was first discussed in [12] through a variant of the higher-order recursive path ordering (HORPO) [14]. This paper furthers that discussion by introducing dependency pairs [1] to LCSTRSs. As a broad framework for termination, this method was initially proposed for unconstrained first-order term rewriting, and was later generalized in a variety of higher-order settings (see, e.g., [31, 23, 30, 2]). Modern termination analyzers rely heavily on dependency pairs.

In higher-order termination analysis, dependency pairs take two forms: the dynamic [31, 23] and the static [30, 2, 24, 7]. This paper concentrates on *static* dependency pairs, and is based on the definitions in [7, 24]. First-order dependency pairs with logical constraints have been informally defined by the third author [15], from which we also take inspiration.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 57; pp. 57:1–57:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For program analysis, the traditional notion of termination can be inefficient, and arguably insufficient. It assumes that the whole program is known and analyzed, i.e., *closed-world* analysis. This way even small programs that happen to import a large standard library need sophisticated analysis; local changes in a multipart, previously verified program also require the entire analysis to be redone. As O'Hearn [26] argues (in a different context), studying *open-world* analysis opens up many applications. In particular, it is practically desirable to analyze the termination of standard libraries – or modules of a larger program in general – without prior knowledge of how the functions they define may be used.

It is tricky to characterize such a property, especially in the presence of higher-order arguments. For example, map and fold are usually considered "terminating", even though passing a non-terminating function to them can surely result in non-termination. Hence, we need to narrow our focus to certain "reasonable" calls. On the other hand, the function $\mathsf{app}\ (\mathsf{lam}\ f) \to f$ where $\mathsf{app} : \mathsf{o} \to \mathsf{o} \to \mathsf{o}$ and $\mathsf{lam} : (\mathsf{o} \to \mathsf{o}) \to \mathsf{o}$ would generally be considered "non-terminating", because if we define $\mathsf{w}\ x \to \mathsf{app}\ x\ x$, an infinite rewrite sequence starts from $\mathsf{app}\ (\mathsf{lam}\ \mathsf{w})\ (\mathsf{lam}\ \mathsf{w})$ – this encodes the famous Ω in the untyped λ-calculus. The property we are looking for must distinguish map and fold from app.

To capture this property, we propose a new concept, called *universal computability.* In light of information hiding, this concept can be further generalized to *public computability.* We will see that static dependency pairs are a natural vehicle for analyzing these properties.

Various modular aspects of term rewriting have been studied by the community. Our scenario roughly corresponds to hierarchical combinations [27, 28, 29, 6], where different parts of a program are analyzed separately. We follow this terminology so that it will be easier to compare our work with the literature. However, our setup – higher-order constrained rewriting – is separate from the first-order and unconstrained setting in which hierarchical combinations were initially proposed. Furthermore, our approach has a different focus – namely, the use of static dependency pairs.

Contributions. We recall the formalism of LCSTRSs and the predicate of computability in Section 2. Then the contributions of this paper follow:

- We propose in Section 3 the first definition of *dependency pairs* for higher-order logically constrained TRSs. This is also the first DP approach for constrained rewriting as the prior work on first-order constrained dependency pairs [15] has never been formally published.
- We define in Section 4 the *constrained DP framework* for termination analysis with five classes of *DP processors*, which can be used to simplify termination problems.
- We extend the notion of a *hierarchical combination* [27, 28, 29, 6] to LCSTRSs and define *universal* and *public computability* in Section 5. We also fine-tune the DP framework to support these properties, and provide extra DP processors for public computability. This allows the DP framework to be used for open-world analysis. We base Section 5 on LCSTRSs to emphasize those notions in real-world programming, but they are new and of theoretical interest in higher-order term rewriting even without logical constraints.
- We have implemented the DP framework for both termination and public computability in our open-source analyzer Cora. We describe the experimental evaluation in Section 6.

2 Preliminaries

In this section, we collect the preliminary definitions and results we need from the literature. First, we recall the definition of an LCSTRS [12]. In this paper, we put a restriction on rewrite rules: ℓ is always a pattern in $\ell \to r\ [\varphi]$. Next, we recall the definition of computability (with accessibility) from [7]. This version is particularly tailored for static dependency pairs.

2.1 Logically Constrained STRSs

Terms Modulo Theories. Given a non-empty set $\mathcal{S}$ of *sorts* (or *base types*), the set $\mathcal{T}$ of simple types over $\mathcal{S}$ is generated by the grammar $\mathcal{T} ::= \mathcal{S} \mid (\mathcal{T} \to \mathcal{T})$. Right-associativity is assigned to → so we can omit some parentheses. Given disjoint sets $\mathcal{F}$ and $\mathcal{V}$, whose elements we call *function symbols* and *variables*, respectively, the set $\mathfrak{T}$ of *pre-terms* over $\mathcal{F}$ and $\mathcal{V}$ is generated by the grammar $\mathfrak{T} ::= \mathcal{F} \mid \mathcal{V} \mid (\mathfrak{T}\ \mathfrak{T})$. Left-associativity is assigned to the juxtaposition operation, called *application*, so $t_0\ t_1\ t_2$ stands for $((t_0\ t_1)\ t_2)$, for example.

We assume every function symbol and variable is assigned a unique type. Typing works as expected: if pre-terms t_0 and t_1 have types $A \to B$ and A, respectively, $t_0\ t_1$ has type B. The set $T(\mathcal{F}, \mathcal{V})$ of *terms* over $\mathcal{F}$ and $\mathcal{V}$ consists of pre-terms that have a type. We write $t : A$ if a term t has type A. We assume there are infinitely many variables of each type.

The set $\mathrm{Var}(t)$ of variables in $t \in T(\mathcal{F}, \mathcal{V})$ is defined by $\mathrm{Var}(f) = \emptyset$ for $f \in \mathcal{F}$, $\mathrm{Var}(x) = \{\, x \,\}$ for $x \in \mathcal{V}$ and $\mathrm{Var}(t_0\ t_1) = \mathrm{Var}(t_0) \cup \mathrm{Var}(t_1)$. A term t is called *ground* if $\mathrm{Var}(t) = \emptyset$.

For constrained rewriting, we make further assumptions. First, we assume that there is a distinguished subset $\mathcal{S}_\vartheta$ of $\mathcal{S}$, called the set of *theory sorts*. The grammar $\mathcal{T}_\vartheta ::= \mathcal{S}_\vartheta \mid (\mathcal{S}_\vartheta \to \mathcal{T}_\vartheta)$ generates the set $\mathcal{T}_\vartheta$ of *theory types* over $\mathcal{S}_\vartheta$. Note that a theory type is essentially a non-empty list of theory sorts. Next, we assume that there is a distinguished subset $\mathcal{F}_\vartheta$ of $\mathcal{F}$, called the set of *theory symbols*, and that the type of every theory symbol is in $\mathcal{T}_\vartheta$, which means that the type of any argument passed to a theory symbol is a theory sort. Theory symbols whose type is a theory sort are called *values*. Elements of $T(\mathcal{F}_\vartheta, \mathcal{V})$ are called *theory terms*.

Theory symbols are interpreted in an underlying theory: given an $\mathcal{S}_\vartheta$-indexed family of sets $(\mathfrak{X}_A)_{A \in \mathcal{S}_\vartheta}$, we extend it to a $\mathcal{T}_\vartheta$-indexed family by letting $\mathfrak{X}_{A \to B}$ be the set of mappings from $\mathfrak{X}_A$ to $\mathfrak{X}_B$; an *interpretation* of theory symbols is a $\mathcal{T}_\vartheta$-indexed family of mappings $(\llbracket\cdot\rrbracket_A)_{A \in \mathcal{T}_\vartheta}$ where $\llbracket\cdot\rrbracket_A$ assigns to each theory symbol of type A an element of $\mathfrak{X}_A$ and is bijective if $A \in \mathcal{S}_\vartheta$. Given an interpretation of theory symbols $(\llbracket\cdot\rrbracket_A)_{A \in \mathcal{T}_\vartheta}$, we extend each indexed mapping $\llbracket\cdot\rrbracket_B$ to one that assigns to each *ground theory term* of type B an element of $\mathfrak{X}_B$ by letting $\llbracket t_0\ t_1\rrbracket_B$ be $\llbracket t_0\rrbracket_{A \to B}(\llbracket t_1\rrbracket_A)$. We write just $\llbracket\cdot\rrbracket$ when the type can be deduced.

▶ **Example 1.** Let $\mathcal{S}_\vartheta$ be $\{\, \mathsf{int} \,\}$. Then $\mathsf{int} \to \mathsf{int} \to \mathsf{int}$ is a theory type over $\mathcal{S}_\vartheta$ while $(\mathsf{int} \to \mathsf{int}) \to \mathsf{int}$ is not. Let $\mathcal{F}_\vartheta$ be $\{\, - \,\} \cup \mathbb{Z}$ where $- : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$ and $n : \mathsf{int}$ for all $n \in \mathbb{Z}$. The values are the elements of $\mathbb{Z}$. Let $\mathfrak{X}_{\mathsf{int}}$ be $\mathbb{Z}$, $\llbracket\cdot\rrbracket_{\mathsf{int}}$ be the identity mapping and $\llbracket - \rrbracket$ be the mapping $\lambda m.\, \lambda n.\, m - n$. The interpretation of $(-)\ 1$ is the mapping $\lambda n.\, 1 - n$.

Substitutions, Contexts and Subterms. Type-preserving mappings from $\mathcal{V}$ to $T(\mathcal{F}, \mathcal{V})$ are called *substitutions*. Every substitution σ extends to a type-preserving mapping $\bar{\sigma}$ from $T(\mathcal{F}, \mathcal{V})$ to $T(\mathcal{F}, \mathcal{V})$. We write $t\sigma$ for $\bar{\sigma}(t)$ and define it as follows: $f\sigma = f$ for $f \in \mathcal{F}$, $x\sigma = \sigma(x)$ for $x \in \mathcal{V}$ and $(t_0\ t_1)\sigma = (t_0\sigma)\ (t_1\sigma)$. Let $[x_1 := t_1, \ldots, x_n := t_n]$ denote the substitution σ such that $\sigma(x_i) = t_i$ for all i, and $\sigma(y) = y$ for all $y \in \mathcal{V} \setminus \{\, x_1, \ldots, x_n \,\}$.

A context is a term containing a hole. Let $\Box$ be a special terminal symbol and assign to it a type A; a *context* $C[]$ is an element of $T(\mathcal{F}, \mathcal{V} \cup \{\, \Box \,\})$ such that $\Box$ occurs in $C[]$ exactly once. Given a term $t : A$, let $C[t]$ denote the term produced by replacing $\Box$ in $C[]$ with t.

A term t is called a (maximally applied) *subterm* of a term s, written as $s \trianglerighteq t$, if either $s = t$, $s = s_0\ s_1$ where $s_1 \trianglerighteq t$, or $s = s_0\ s_1$ where $s_0 \trianglerighteq t$ and $s_0 \neq t$; i.e., $s = C[t]$ for $C[]$ that is not of form $C'[\Box\ t_1]$. We write $s \triangleright t$ and call t a *proper subterm* of s if $s \trianglerighteq t$ and $s \neq t$.

Constrained Rewriting. Constrained rewriting requires the theory sort bool: we henceforth assume that $\mathsf{bool} \in \mathcal{S}_\vartheta$, $\{\, \mathfrak{f}, \mathfrak{t} \,\} \subseteq \mathcal{F}_\vartheta$, $\mathfrak{X}_{\mathsf{bool}} = \{\, 0, 1 \,\}$, $\llbracket \mathfrak{f} \rrbracket_{\mathsf{bool}} = 0$ and $\llbracket \mathfrak{t} \rrbracket_{\mathsf{bool}} = 1$. A *logical constraint* is a theory term φ such that φ has type bool and the type of each variable in $\mathrm{Var}(\varphi)$

is a theory sort. A (constrained) *rewrite rule* is a triple $\ell \to r\ [\varphi]$ where ℓ and r are terms which have the same type, φ is a logical constraint, the type of each variable in $\mathrm{Var}(r) \setminus \mathrm{Var}(\ell)$ is a theory sort and ℓ is a pattern that takes the form $f\ t_1 \cdots t_n$ for some function symbol f and contains at least one function symbol in $\mathcal{F} \setminus \mathcal{F}_\vartheta$. Here a *pattern* is a term whose subterms are either $f\ t_1 \cdots t_n$ for some function symbol f or a variable. A substitution σ is said to *respect* $\ell \to r\ [\varphi]$ if $\sigma(x)$ is a value for all $x \in \mathrm{Var}(\varphi) \cup (\mathrm{Var}(r) \setminus \mathrm{Var}(\ell))$ and $[\![\varphi\sigma]\!] = 1$.

A *logically constrained simply-typed term rewriting system* (LCSTRS) collects the above data – $\mathcal{S}$, $\mathcal{S}_\vartheta$, $\mathcal{F}$, $\mathcal{F}_\vartheta$, $\mathcal{V}$, $(\mathfrak{X}_A)$ and $[\![\cdot]\!]$ – along with a set $\mathcal{R}$ of rewrite rules. We usually let $\mathcal{R}$ alone stand for the system. The set $\mathcal{R}$ induces the *rewrite relation* $\to_\mathcal{R}$ over terms: $t \to_\mathcal{R} t'$ if and only if there exists a context $C[]$ such that either (1) $t = C[\ell\sigma]$ and $t' = C[r\sigma]$ for some rewrite rule $\ell \to r\ [\varphi] \in \mathcal{R}$ and some substitution σ which respects $\ell \to r\ [\varphi]$, or (2) $t = C[f\ v_1 \cdots v_n]$ and $t' = C[v']$ for some theory symbol f and some values $v_1, \ldots, v_n, v'$ with $n > 0$ and $[\![f\ v_1 \cdots v_n]\!] = [\![v']\!]$. When no ambiguity arises, we may write $\to$ for $\to_\mathcal{R}$.

If $t \to_\mathcal{R} t'$ due to the second condition above, we also write $t \to_\kappa t'$ and call it a *calculation step*. Theory symbols that are not a value are called *calculation symbols*. Let $t{\downarrow_\kappa}$ denote the (unique) κ-normal form of t, i.e., the term t' such that $t \to_\kappa^* t'$ and $t' \not\to_\kappa t''$ for any t''. For example, $(f\ (7 * (3 * 2))){\downarrow_\kappa} = f\ 42$ if f is not a calculation symbol, or if $f : \mathsf{int} \to A \to B$.

A rewrite rule $\ell \to r\ [\varphi]$ is said to *define* a function symbol f if $\ell = f\ t_1 \cdots t_n$. Given an LCSTRS $\mathcal{R}$, f is called a *defined symbol* if some rewrite rule in $\mathcal{R}$ defines f. Let $\mathcal{D}$ denote the set of defined symbols. Values and function symbols in $\mathcal{F} \setminus (\mathcal{F}_\vartheta \cup \mathcal{D})$ are called *constructors*.

▶ **Example 2.** Below is the factorial function in continuation-passing style as an LCSTRS:

$$\begin{array}{lll} \mathsf{fact}\ n\ k \to k\ 1 & [n \leq 0] & \mathsf{comp}\ g\ f\ x \to g\ (f\ x) \\ \mathsf{fact}\ n\ k \to \mathsf{fact}\ (n - 1)\ (\mathsf{comp}\ k\ ((*)\ n)) & [n > 0] & \mathsf{id}\ x \to x \end{array}$$

We use infix notation for some binary operators, and omit the logical constraint of a rewrite rule when it is $\mathfrak{t}$. An example rewrite sequence is $\mathsf{fact}\ 1\ \mathsf{id} \to \mathsf{fact}\ (1 - 1)\ (\mathsf{comp}\ \mathsf{id}\ ((*)\ 1)) \to_\kappa \mathsf{fact}\ 0\ (\mathsf{comp}\ \mathsf{id}\ ((*)\ 1)) \to \mathsf{comp}\ \mathsf{id}\ ((*)\ 1)\ 1 \to \mathsf{id}\ ((*)\ 1\ 1) \to_\kappa \mathsf{id}\ 1 \to 1$.

2.2 Accessibility and Computability

We recall the notion of computability with accessibility – which originates from [3] and is reformulated in [7] to couple with static dependency pairs – and adapt the notion of accessible function passing [7] to LCSTRSs.

Accessibility. Assume given a *sort ordering* – a quasi-ordering $\succsim$ over $\mathcal{S}$ whose strict part $\succ\, = \,\succsim \setminus \precsim$ is well-founded. We inductively define two relations $\succsim_+$ and $\succ_-$ over $\mathcal{S}$ and $\mathcal{T}$: given a sort A and a type $B = B_1 \to \cdots \to B_n \to C$ where C is a sort and $n \geq 0$, $A \succsim_+ B$ if and only if $A \succsim C$ and $\forall i.\, A \succ_- B_i$, and $A \succ_- B$ if and only if $A \succ C$ and $\forall i.\, A \succsim_+ B_i$.

Given a function symbol $f : A_1 \to \cdots \to A_n \to B$ where B is a sort, the set $\mathrm{Acc}(f)$ of the *accessible argument positions* of f is defined as $\{\, 1 \leq i \leq n \mid B \succsim_+ A_i \,\}$. A term t is called an *accessible subterm* of a term s, written as $s \trianglerighteq_{\mathrm{acc}} t$, if either $s = t$, or $s = f\ s_1 \cdots s_m$ for some $f \in \mathcal{F}$ and there exists $k \in \mathrm{Acc}(f)$ such that $s_k \trianglerighteq_{\mathrm{acc}} t$. An LCSTRS $\mathcal{R}$ is called *accessible function passing* (AFP) if there exists a sort ordering such that for all $f\ s_1 \cdots s_m \to r\ [\varphi] \in \mathcal{R}$ and $x \in \mathrm{Var}(f\ s_1 \cdots s_m) \cap \mathrm{Var}(r) \setminus \mathrm{Var}(\varphi)$, there exists k such that $s_k \trianglerighteq_{\mathrm{acc}} x$.

▶ **Example 3.** An LCSTRS $\mathcal{R}$ is AFP (with $\succsim$ equating all the sorts) if for all $f\ s_1 \cdots s_m \to r\ [\varphi] \in \mathcal{R}$ and $i \in \{\, 1, \ldots, m \,\}$, the type of each proper subterm of s_i is a sort. Rewrite rules for common higher-order functions, e.g., map and fold, usually fit this criterion.

Consider $\{\,\mathsf{complst}\ \mathsf{fnil}\ x \to x, \mathsf{complst}\ (\mathsf{fcons}\ f\ l)\ x \to \mathsf{complst}\ l\ (f\ x)\,\}$, where $\mathsf{complst} : \mathsf{funlist} \to \mathsf{int} \to \mathsf{int}$ composes a list of *functions*. This system is AFP with $\mathsf{funlist} \succ \mathsf{int}$.

The system $\{\,\mathsf{app}\ (\mathsf{lam}\ f) \to f\,\}$ in Section 1 is not AFP since $\mathsf{o} \succ \mathsf{o}$ cannot be true.

Computability. A term is called *neutral* if it takes the form $x\ t_1 \cdots t_n$ for some variable x. A set of *reducibility candidates*, or an *RC-set*, for the rewrite relation $\to_{\mathcal{R}}$ of an LCSTRS $\mathcal{R}$ is an $\mathcal{S}$-indexed family of sets $(I_A)_{A \in \mathcal{S}}$ (let I denote $\bigcup_A I_A$) satisfying the following conditions:

(1) Each element of I_A is a terminating (with respect to $\to_{\mathcal{R}}$) term of type A.

(2) Given terms s and t such that $s \to_{\mathcal{R}} t$, if s is in I_A, so is t.

(3) Given a neutral term s, if t is in I_A for all t such that $s \to_{\mathcal{R}} t$, so is s.

Given an RC-set I for $\to_{\mathcal{R}}$, a term t_0 is called *I-computable* if either the type of t_0 is a sort and $t_0 \in I$, or the type of t_0 is $A \to B$ and $t_0\ t_1$ is I-computable for all I-computable $t_1 : A$.

We are interested in a specific RC-set $\mathbb{C}$, whose existence is guaranteed by Theorem 4.

▶ **Theorem 4** (see [7]). *Given a sort ordering and an RC-set I for $\to_{\mathcal{R}}$, let $\Rrightarrow_I$ be the relation over terms such that $s \Rrightarrow_I t$ if and only if both s and t have a base type, $s = f\ s_1 \cdots s_m$ for some function symbol f, $t = s_k\ t_1 \cdots t_n$ for some $k \in \mathrm{Acc}(f)$ and t_i is I-computable for all i.*

Given an LCSTRS $\mathcal{R}$ with a sort ordering, there exists an RC-set $\mathbb{C}$ for $\to_{\mathcal{R}}$ such that $t \in \mathbb{C}_A$ if and only if $t : A$ is terminating with respect to $\to_{\mathcal{R}} \cup \Rrightarrow_{\mathbb{C}}$, and for all t' such that $t \to_{\mathcal{R}}^* t'$, if $t' = f\ t_1 \cdots t_n$ for some function symbol f, t_i is $\mathbb{C}$-computable for all $i \in \mathrm{Acc}(f)$.

Thus, given a $\mathbb{C}$-computable term $f\ t_1 \cdots t_n$, all its reducts and the accessible arguments – t_i for $i \in \mathrm{Acc}(f)$ – are also $\mathbb{C}$-computable. We consider $\mathbb{C}$-computability throughout this paper.

3 Static Dependency Pairs for LCSTRSs

Originally proposed for unconstrained first-order term rewriting, the dependency pair approach [1] – a methodology that analyzes the recursive structure of function calls – is at the heart of most modern automatic termination analyzers for various styles of term rewriting. There follow multiple higher-order generalizations, among which we adopt here the *static* branch [24, 7]. As we shall see in Section 5, this approach extends well to open-world analysis.

In this section, we adapt static dependency pairs to LCSTRSs. We start with a notation:

▶ **Definition 5.** *Given an LCSTRS $\mathcal{R}$, let $\mathcal{F}^\sharp$ be $\mathcal{F} \cup \{\, f^\sharp \mid f \in \mathcal{D}\,\}$ where $\mathcal{D}$ is the set of defined symbols in $\mathcal{R}$ and $f^\sharp$ is a fresh function symbol for all f. Let dp be a fresh sort, and for each defined symbol $f : A_1 \to \cdots \to A_n \to B$ where $B \in \mathcal{S}$, we assign $f^\sharp : A_1 \to \cdots \to A_n \to \mathsf{dp}$. Given a term $t = f\ t_1 \cdots t_n \in T(\mathcal{F}, \mathcal{V})$ where $f \in \mathcal{D}$, let $t^\sharp$ denote $f^\sharp\ t_1 \cdots t_n \in T(\mathcal{F}^\sharp, \mathcal{V})$.*

In the presence of logical constraints, a dependency pair should be more than a pair. Two extra components – a logical constraint and a set of variables – keep track of what substitutions are expected by the dependency pair.

▶ **Definition 6.** *A* static dependency pair *(SDP) is a quadruple $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L]$ where $s^\sharp$ and $t^\sharp$ are terms of type dp, φ is a logical constraint and $L \supseteq \mathrm{Var}(\varphi)$ is a set of variables whose types are theory sorts. Given a rewrite rule $\ell \to r\ [\varphi]$, let $\mathrm{SDP}(\ell \to r\ [\varphi])$ denote the set of SDPs of form $\ell^\sharp\ x_1 \cdots x_m \Rightarrow g^\sharp\ t_1 \cdots t_q\ y_{q+1} \cdots y_n\ [\varphi \mid \mathrm{Var}(\varphi) \cup (\mathrm{Var}(r) \setminus \mathrm{Var}(\ell))]$ such that*

(1) *$\ell^\sharp : A_1 \to \cdots \to A_m \to \mathsf{dp}$ while $x_i : A_i$ is a fresh variable for all i,*

(2) *$r\ x_1 \cdots x_m \trianglerighteq g\ t_1 \cdots t_q$ for $g \in \mathcal{D}$, and*

(3) *$g^\sharp : B_1 \to \cdots \to B_n \to \mathsf{dp}$ while $y_i : B_i$ is a fresh variable for all $i > q$.*

Let $\mathrm{SDP}(\mathcal{R})$ be $\bigcup_{\ell \to r\ [\varphi] \in \mathcal{R}} \mathrm{SDP}(\ell \to r\ [\varphi])$. A substitution σ is said to respect *an SDP $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L]$ if $\sigma(x)$ is a ground theory term for all $x \in L$ and $[\![\varphi\sigma]\!] = 1$.*

The component L is new compared to [15]. We shall see its usefulness in Section 4.4, as it gives us more freedom to manipulate dependency pairs. We introduce two shorthand notations for SDPs: $s^\sharp \Rightarrow t^\sharp\ [\varphi]$ for $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid \mathrm{Var}(\varphi)]$, and $s^\sharp \Rightarrow t^\sharp$ for $s^\sharp \Rightarrow t^\sharp\ [\mathsf{t} \mid \emptyset]$.

▶ **Example 7.** Consider the system $\mathcal{R}$ consisting of the following rewrite rules, in which $\mathsf{gcdlist} : \mathsf{intlist} \to \mathsf{int}$, $\mathsf{fold} : (\mathsf{int} \to \mathsf{int} \to \mathsf{int}) \to \mathsf{int} \to \mathsf{intlist} \to \mathsf{int}$ and $\mathsf{gcd} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$.

$$\begin{array}{lllll}
\mathsf{gcdlist} \to \mathsf{fold}\ \mathsf{gcd}\ 0 & & \mathsf{fold}\ f\ y\ \mathsf{nil} \to y & \mathsf{fold}\ f\ y\ (\mathsf{cons}\ x\ l) \to f\ x\ (\mathsf{fold}\ f\ y\ l) & \\
\mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n & [m < 0] & \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n) & & [n < 0] \\
\mathsf{gcd}\ m\ 0 \to m & [m \geq 0] & \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n) & & [m \geq 0 \wedge n > 0]
\end{array}$$

The set $\mathrm{SDP}(\mathcal{R})$ consists of (1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$, (2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$, (3) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l$, (4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0]$, (5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0]$, and (6) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m \geq 0 \wedge n > 0]$. Note that in (1), m' and n' occur on the right-hand side of $\Rightarrow$ but not on the left while they are *not* required to be instantiated to ground theory terms ($L = \emptyset$). This is normal for SDPs [7, 24].

Termination analysis via SDPs is based on the notion of a chain:

▶ **Definition 8.** *Given a set $\mathcal{P}$ of SDPs and a set $\mathcal{R}$ of rewrite rules, a* $(\mathcal{P}, \mathcal{R})$-chain *is a (finite or infinite) sequence $({s_0}^\sharp \Rightarrow {t_0}^\sharp\ [\varphi_0 \mid L_0], \sigma_0), ({s_1}^\sharp \Rightarrow {t_1}^\sharp\ [\varphi_1 \mid L_1], \sigma_1), \ldots$ such that for all i, ${s_i}^\sharp \Rightarrow {t_i}^\sharp\ [\varphi_i \mid L_i] \in \mathcal{P}$, σ_i is a substitution which respects ${s_i}^\sharp \Rightarrow {t_i}^\sharp\ [\varphi_i \mid L_i]$, and ${t_{i-1}}^\sharp \sigma_{i-1} \to_{\mathcal{R}}^* {s_i}^\sharp \sigma_i$ if $i > 0$. The above $(\mathcal{P}, \mathcal{R})$-chain is called* computable *if $u\sigma_i$ is $\mathbb{C}$-computable for all i and u such that $t_i \rhd u$.*

▶ **Example 9.** Following Example 7, $(1, [l := \mathsf{nil}, m := 42, n := 24]), (6, [m := 42, n := 24]), (6, [m := 24, n := 18]), (6, [m := 18, n := 6])$ is a computable $(\mathrm{SDP}(\mathcal{R}), \mathcal{R})$-chain.

The key to establishing termination is the following result:

▶ **Theorem 10.** *An AFP system $\mathcal{R}$ is terminating if there exists no infinite computable* $(\mathrm{SDP}(\mathcal{R}), \mathcal{R})$*-chain.*

The proof (see [11, Appendix A.1]) is very similar to that for unconstrained SDPs [24, 7].

4 The Constrained DP Framework

In this section, we present several techniques based on SDPs, each as a class of *DP processors*; formally, we call this collection of DP processors the *constrained (static) DP framework*. In general, a DP framework [9, 7] constitutes a broad method for termination and non-termination. The presentation here is not complete – for example, we do not consider non-termination – and a complete one is beyond the scope of this paper. We rather focus on the most essential DP processors and those newly designed to handle logical constraints.

For presentation, we fix an LCSTRS $\mathcal{R}$.

▶ **Definition 11.** *A* DP problem *is a set $\mathcal{P}$ of SDPs. A DP problem $\mathcal{P}$ is called* finite *if there exists no infinite computable $(\mathcal{P}, \mathcal{R})$-chain. A* DP processor *is a partial mapping which possibly assigns to a DP problem a set of DP problems. A DP processor ρ is called* sound *if a DP problem $\mathcal{P}$ is finite whenever $\rho(\mathcal{P})$ consists only of finite DP problems.*

Following Theorem 10, in order to establish the termination of an AFP system $\mathcal{R}$, it suffices to show that $\mathrm{SDP}(\mathcal{R})$ is a finite DP problem. Given a collection of sound DP processors, we have the following procedure: (1) $Q := \{\, \mathrm{SDP}(\mathcal{R}) \,\}$; (2) while Q contains a DP problem $\mathcal{P}$ to which some sound DP processor ρ is applicable, $Q := (Q \setminus \{\, \mathcal{P} \,\}) \cup \rho(\mathcal{P})$. If this procedure ends with $Q = \emptyset$, we can conclude that $\mathcal{R}$ is terminating.

4.1 The DP Graph and Its Approximations

The interconnection of SDPs via chains gives rise to a graph, namely, the DP graph [1], which models the reachability between dependency pairs. Since this graph is not computable in general, we follow the usual convention and consider its (over-)approximations:

▶ **Definition 12.** *Given a set $\mathcal{P}$ of SDPs, a* graph approximation (G_θ, θ) *for $\mathcal{P}$ consists of a finite directed graph G_θ and a mapping θ which assigns to each SDP in $\mathcal{P}$ a vertex of G_θ so that there is an edge from $\theta(p_0)$ to $\theta(p_1)$ whenever $(p_0, \sigma_0), (p_1, \sigma_1)$ is a $(\mathcal{P}, \mathcal{R})$-chain for some substitutions σ_0 and σ_1.*

Here (G_θ, θ) approximates the true DP graph by allowing θ to assign a single vertex to multiple (possibly, infinitely many) SDPs, and by allowing G_θ to contain an edge from $\theta(p_0)$ to $\theta(p_1)$ even if p_0 and p_1 are not connected by any $(\mathcal{P}, \mathcal{R})$-chain. In practice, we typically deal with only a finite set $\mathcal{P}$ of SDPs, in which case we usually take a bijection for θ.

This graph structure is useful because we can leverage it to decompose the DP problem.

▶ **Definition 13.** *Given a DP problem $\mathcal{P}$, a* graph processor *computes a graph approximation (G_θ, θ) for $\mathcal{P}$ and the strongly connected components (SCCs) of G_θ, then returns $\{\, \{\, p \in \mathcal{P} \mid \theta(p)$ belongs to $S \,\} \mid S$ is a non-trivial SCC of $G_\theta \,\}$.*

▶ **Example 14.** Following Example 7, a (tight) graph approximation for SDP($\mathcal{R}$) is in Figure 1. If a graph processor produces this graph as the graph approximation, it will return the set of DP problems $\{\, \{\, 3 \,\}, \{\, 4, 5 \,\}, \{\, 6 \,\} \,\}$.

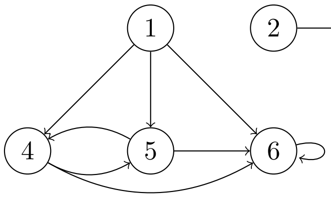


Figure 1 A graph approximation for SDP($\mathcal{R}$) from Example 7.

Implementation. To compute a graph approximation, we adapt the common Cap approach [10, 33] and take theories into account. Considering theories allows us, for example, *not* to have an edge from (6) to (4) in Figure 1.

We assume given a finite set of SDPs and let θ be a bijection. Whether there is an edge from $\theta({s_0}^\sharp \Rightarrow {t_0}^\sharp\ [\varphi_0 \mid L_0])$ to $\theta({s_1}^\sharp \Rightarrow {t_1}^\sharp\ [\varphi_1 \mid L_1])$ – we rename variables if necessary to avoid name collisions between the two SDPs – is determined by the satisfiability (which we check by an SMT solver) of $\varphi_0 \wedge \varphi_1 \wedge \zeta({t_0}^\sharp, {s_1}^\sharp)$ where $\zeta(u, v)$ is defined as follows:

- If $u = f\ u_1 \cdots u_n$ where $f \in \mathcal{F}^\sharp$ and no rewrite rule in $\mathcal{R}$ takes the form $f\ \ell_1 \cdots \ell_k \to r\ [\varphi]$ for $k \leq n$, we define $\zeta(u, v)$ in two cases:
  (1) $\zeta(u, v) = \zeta(u_1, v_1) \wedge \cdots \wedge \zeta(u_n, v_n)$ if $v = f\ v_1 \cdots v_n$.
  (2) $\zeta(u, v) = \mathfrak{f}$ if $v = g\ v_1 \cdots v_m$ for some function symbol g other than f, and either f is not a theory symbol or g is not a value.
- Suppose $\zeta(u, v)$ is not defined above; $\zeta(u, v) = (u \equiv v)$ if $u \in T(\mathcal{F}_\vartheta, L_0)$ has a base type and v is a theory term in which the type of each variable is a theory sort, and $\zeta(u, v) = \mathfrak{t}$ otherwise.

See [11, Appendix A.2] for the proof that this approach produces a graph approximation.

Then strongly connected components can be computed by Tarjan's algorithm [32].

▶ **Example 15.** In Figure 1, since $(m_0 \geq 0 \wedge n_0 > 0) \wedge m_1 < 0 \wedge (n_0 \equiv m_1 \wedge m_0 \bmod n_0 \equiv n_1)$ is unsatisfiable, there is no edge from (6) to (4).

4.2 The Subterm Criterion

The subterm criterion [13, 24] handles structural recursion and allows us to remove decreasing SDPs without considering rewrite rules in $\mathcal{R}$. We start with defining projections:

▶ **Definition 16.** *Let* heads$(\mathcal{P})$ *denote the set of function symbols heading either side of an SDP in $\mathcal{P}$. A* projection *ν for a set $\mathcal{P}$ of SDPs is a mapping from* heads$(\mathcal{P})$ *to integers such that $1 \leq \nu(f^\sharp) \leq n$ if $f^\sharp : A_1 \to \cdots \to A_n \to \mathsf{dp}$. Let $\bar{\nu}(f^\sharp\ t_1 \cdots t_n)$ denote $t_{\nu(f^\sharp)}$.*

A projection chooses an argument position for each relevant function symbol so that arguments at those positions do not increase in a chain.

▶ **Definition 17.** *Given a set $\mathcal{P}$ of SDPs, a projection ν is said to* $\rhd$-orient *a subset $\mathcal{P}'$ of $\mathcal{P}$ if $\bar{\nu}(s^\sharp) \rhd \bar{\nu}(t^\sharp)$ for all $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P}'$ and $\bar{\nu}(s^\sharp) = \bar{\nu}(t^\sharp)$ for all $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P} \setminus \mathcal{P}'$. A* subterm criterion processor *assigns to a DP problem $\mathcal{P}$ the singleton $\{\, \mathcal{P} \setminus \mathcal{P}' \,\}$ for some non-empty subset $\mathcal{P}'$ of $\mathcal{P}$ such that there exists a projection for $\mathcal{P}$ which $\rhd$-orients $\mathcal{P}'$.*

▶ **Example 18.** Following Example 14, a subterm criterion processor is applicable to $\{\, 3 \,\}$. Let $\nu(\mathsf{fold}^\sharp)$ be 3 so that $\bar{\nu}(\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l)) = \mathsf{cons}\ x\ l \rhd l = \bar{\nu}(\mathsf{fold}^\sharp\ f\ y\ l)$. The processor returns $\{\, \emptyset \,\}$, and the empty DP problem can (trivially) be removed by a graph processor.

Implementation. The search for a suitable projection can be done through SMT and is standard: we introduce an integer variable $N_{f^\sharp}$ that represents $\nu(f^\sharp)$ for each $f^\sharp \in \mathrm{heads}(\mathcal{P})$, and a boolean variable strict_p for each $p \in \mathcal{P}$; then we encode the requirement per SDP.

4.3 Integer Mappings

The subterm criterion deals with recursion over the structure of terms, but not recursion over, say, integers, which requires us to utilize the information in logical constraints. In this subsection, we assume that $\mathsf{int} \in \mathcal{S}_\vartheta$ and $\mathcal{F}_\vartheta \supseteq \{\, \geq, >, \wedge \,\}$, where $\geq : \mathsf{int} \to \mathsf{int} \to \mathsf{bool}$, $> : \mathsf{int} \to \mathsf{int} \to \mathsf{bool}$ and $\wedge : \mathsf{bool} \to \mathsf{bool} \to \mathsf{bool}$ are interpreted in the standard way.

▶ **Definition 19.** *Given a set $\mathcal{P}$ of SDPs, for all $f^\sharp \in$* heads$(\mathcal{P})$ *(see Definition 16) where $f^\sharp : A_1 \to \cdots \to A_n \to \mathsf{dp}$, let $\iota(f^\sharp)$ be the subset of $\{\, 1, \ldots, n \,\}$ such that $i \in \iota(f^\sharp)$ if and only if $A_i \in \mathcal{S}_\vartheta$ and the i-th argument of any occurrence of $f^\sharp$ in an SDP $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P}$ is in $T(\mathcal{F}_\vartheta, L)$. Let $\mathcal{X}(f^\sharp)$ be a set of fresh variables $\{\, x_{f^\sharp,i} \mid i \in \iota(f^\sharp) \,\}$ where $x_{f^\sharp,i} : A_i$ for all i. An* integer mapping *$\mathcal{J}$ for $\mathcal{P}$ is a mapping from* heads$(\mathcal{P})$ *to theory terms such that for all $f^\sharp$, $\mathcal{J}(f^\sharp) : \mathsf{int}$ and $\mathrm{Var}(\mathcal{J}(f^\sharp)) \subseteq \mathcal{X}(f^\sharp)$. Let $\bar{\mathcal{J}}(f^\sharp\ t_1 \cdots t_n)$ denote $\mathcal{J}(f^\sharp)[x_{f^\sharp,i} \coloneqq t_i]_{i \in \iota(f^\sharp)}$.*

With integer mappings, we can handle decreasing integer values.

▶ **Definition 20.** *Given a set $\mathcal{P}$ of SDPs, an integer mapping $\mathcal{J}$ is said to* $>$-orient *a subset $\mathcal{P}'$ of $\mathcal{P}$ if $\varphi \models \bar{\mathcal{J}}(s^\sharp) \geq 0 \wedge \bar{\mathcal{J}}(s^\sharp) > \bar{\mathcal{J}}(t^\sharp)$ for all $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P}'$, and $\varphi \models \bar{\mathcal{J}}(s^\sharp) \geq \bar{\mathcal{J}}(t^\sharp)$ for all $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P} \setminus \mathcal{P}'$, where $\varphi \models \varphi'$ denotes that $[\![\varphi\sigma]\!] = 1$ implies $[\![\varphi'\sigma]\!] = 1$ for each substitution σ which maps variables in $\mathrm{Var}(\varphi) \cup \mathrm{Var}(\varphi')$ to values. An* integer mapping processor *assigns to a DP problem $\mathcal{P}$ the singleton $\{\, \mathcal{P} \setminus \mathcal{P}' \,\}$ for some non-empty subset $\mathcal{P}'$ of $\mathcal{P}$ such that there exists an integer mapping for $\mathcal{P}$ which $>$-orients $\mathcal{P}'$.*

▶ **Example 21.** Following Example 14, an integer mapping processor is applicable to $\{\,6\,\}$. Let $\mathcal{J}(\mathsf{gcd}^\sharp)$ be $x_{\mathsf{gcd}^\sharp,2}$ so that $\bar{\mathcal{J}}(\mathsf{gcd}^\sharp\ m\ n) = n$, $\bar{\mathcal{J}}(\mathsf{gcd}^\sharp\ n\ (m \bmod n)) = m \bmod n$ and $m \geq 0 \wedge n > 0 \models n \geq 0 \wedge n > m \bmod n$. The processor returns $\{\,\emptyset\,\}$, and the empty DP problem can (trivially) be removed by a graph processor.

Implementation. There are several ways to implement integer mapping processors. In our implementation, we generate a number of "interpretation candidates" and use an SMT encoding to select for each $f^\sharp \in \mathrm{heads}(\mathcal{P})$ one candidate that satisfies the requirements. Candidates include forms such as $\mathcal{J}(f^\sharp) = x_{f^\sharp,i}$ and those that are generated from the SDPs' logical constraints – e.g., given $\mathsf{f}^\sharp\ x\ y \Rightarrow \mathsf{g}^\sharp\ x\ (y+1)\ [y < x]$, we generate $\mathcal{J}(\mathsf{f}^\sharp) = x_{\mathsf{f}^\sharp,1} - x_{\mathsf{f}^\sharp,2} - 1$ because $y < x$ implies $x - y - 1 \geq 0$.

4.4 Theory Arguments

Integer mapping processors have a clear limitation: what if some key variables do not occur in the set L? This is observed in the remaining DP problem $\{\,4, 5\,\}$ from Example 7. It is clearly finite but no integer mapping processor is applicable since $\iota(\mathsf{gcd}^\sharp) = \emptyset$.

This restriction exists for a reason. Variables that are not guaranteed to be instantiated to theory terms may be instantiated to *non-deterministic* terms – e.g., $\{\,\mathsf{f}^\sharp\ x\ y\ z \Rightarrow \mathsf{f}^\sharp\ x\ (x+1)\ (x-1)\ [y < z]\,\}$ is not a finite DP problem if $\mathcal{R} \supseteq \{\,\mathsf{c}\ x\ y \to x, \mathsf{c}\ x\ y \to y\,\}$.

The problem of $\{\,4, 5\,\}$ arises because each SDP focuses on only one argument: for example, the logical constraint (with the component L) of (5) only concerns n so in principle we cannot assume anything about m. Yet, if (5) follows (4) in a chain, we *can* derive that m must be instantiated to a ground theory term (we call such an argument a *theory argument*). We explore a way of propagating this information.

▶ **Definition 22.** *A* theory argument (position) mapping τ *for a set* $\mathcal{P}$ *of SDPs is a mapping from* $\mathrm{heads}(\mathcal{P})$ *(see Definition 16) to subsets of* $\mathbb{Z}$ *such that* $\tau(f^\sharp) \subseteq \{\,1 \leq i \leq m \mid A_i \in \mathcal{S}_\vartheta\,\}$ *if* $f^\sharp : A_1 \to \cdots \to A_m \to \mathsf{dp}$, s_i *is a theory term and the type of each variable in* $\mathrm{Var}(s_i)$ *is a theory sort for all* $f^\sharp\ s_1 \cdots s_m \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P}$ *and* $i \in \tau(f^\sharp)$*, and* t_j *is a theory term and* $\mathrm{Var}(t_j) \subseteq L \cup \bigcup_{i \in \tau(f^\sharp)} \mathrm{Var}(s_i)$ *for all* $f^\sharp\ s_1 \cdots s_m \Rightarrow g^\sharp\ t_1 \cdots t_n\ [\varphi \mid L] \in \mathcal{P}$ *and* $j \in \tau(g^\sharp)$*. Let* $\bar{\tau}(f^\sharp\ s_1 \cdots s_m \Rightarrow t^\sharp\ [\varphi \mid L])$ *denote* $f^\sharp\ s_1 \cdots s_m \Rightarrow t^\sharp\ [\varphi \mid L \cup \bigcup_{i \in \tau(f^\sharp)} \mathrm{Var}(s_i)]$*.*

By a theory argument mapping, we choose a subset of the given set of SDPs from which the theory argument information is propagated.

▶ **Definition 23.** *Given a set* $\mathcal{P}$ *of SDPs, a theory argument mapping* τ *is said to* fix *a subset* $\mathcal{P}'$ *of* $\mathcal{P}$ *if* $\bigcup_{i \in \tau(f^\sharp)} \mathrm{Var}(t_i) \subseteq L$ *for all* $s^\sharp \Rightarrow f^\sharp\ t_1 \cdots t_n\ [\varphi \mid L] \in \mathcal{P}'$*. A* theory argument processor *assigns to a DP problem* $\mathcal{P}$ *the pair* $\{\,\{\,\bar{\tau}(p) \mid p \in \mathcal{P}\,\}, \mathcal{P} \setminus \mathcal{P}'\,\}$ *for some non-empty subset* $\mathcal{P}'$ *of* $\mathcal{P}$ *such that there exists a theory argument mapping for* $\mathcal{P}$ *which fixes* $\mathcal{P}'$*.*

▶ **Example 24.** Following Example 14, a theory argument processor is applicable to $\{\,4, 5\,\}$. Let $\tau(\mathsf{gcd}^\sharp)$ be $\{\,1\,\}$ so that τ fixes $\{\,4\,\}$. The processor returns the pair $\{\,\{\,4, (7)\ \mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0 \mid \{m, n\}]\,\}, \{\,5\,\}\,\}$. The integer mapping processor with $\mathcal{J}(\mathsf{gcd}^\sharp) = -x_{\mathsf{gcd}^\sharp,1}$ removes (4) from $\{\,4, 7\,\}$. Then $\{\,7\,\}$ and $\{\,5\,\}$ can be removed by graph processors.

Implementation. To find a valid theory argument mapping, we simply start by setting $\tau(f^\sharp) = \{\,1, \ldots, m\,\}$ for all $f^\sharp$, and choose one SDP to fix. Then we iteratively remove arguments that do not satisfy the condition until no such argument is left.

4.5 Reduction Pairs

Although it is not needed by the running example, we present a constrained variant of *reduction pair processors*, which are at the heart of most unconstrained termination analyzers.

▶ **Definition 25.** *A* constrained relation *R is a set of quadruples (s, t, φ, L) where s and t are terms which have the same type, φ is a logical constraint and $L \supseteq \mathrm{Var}(\varphi)$ is a set of variables whose types are theory sorts. We write $s\ R^L_\varphi\ t$ if $(s, t, \varphi, L) \in R$. A binary relation R' over terms is said to* cover *a constrained relation R if $s\ R^L_\varphi\ t$ implies that $(s\sigma)\downarrow_\kappa R'\ (t\sigma)\downarrow_\kappa$ for each substitution σ such that $\sigma(x)$ is a ground theory term for all $x \in L$ and $[\![\varphi\sigma]\!] = 1$.*

A constrained reduction pair *$(\succeq, \succ)$ is a pair of constrained relations such that $\succeq$ is covered by some reflexive relation $\sqsupseteq$ which includes $\rightarrow_\kappa$ and is monotonic (i.e., $s \sqsupseteq t$ implies $C[s] \sqsupseteq C[t]$), $\succ$ is covered by some well-founded relation $\sqsupset$, and $\sqsupseteq ; \sqsupset \subseteq \sqsupset^+$.*

▶ **Definition 26.** *A* reduction pair processor *assigns to a DP problem $\mathcal{P}$ the singleton $\{\,\mathcal{P}\backslash\mathcal{P}'\,\}$ for some non-empty subset $\mathcal{P}'$ of $\mathcal{P}$ such that there exists a constrained reduction pair $(\succeq, \succ)$ where (1) $s^\sharp \succ^L_\varphi t^\sharp$ for all $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P}'$, (2) $s^\sharp \succeq^L_\varphi t^\sharp$ for all $s^\sharp \Rightarrow t^\sharp\ [\varphi \mid L] \in \mathcal{P} \setminus \mathcal{P}'$, and (3) $\ell \succeq^{\mathrm{Var}(\varphi)\cup(\mathrm{Var}(r)\setminus\mathrm{Var}(\ell))}_\varphi r$ for all $\ell \rightarrow r\ [\varphi] \in \mathcal{R}$.*

While a variety of reduction pairs have been proposed for unconstrained rewriting, it is not yet the case in a higher-order and constrained setting: so far the only one is a limited version of HORPO [12], which is adapted into a weakly monotonic reduction pair [18] and then implemented in the DP framework. This is still a prototype definition.

We have included reduction pair processors here because their definition allows us to start designing constrained reduction pairs. In particular, as unconstrained reduction pairs can be used as the covering pair $(\sqsupseteq, \sqsupset)$, it is likely that many of them (such as variants of HORPO and weakly monotonic algebras) can be adapted.

We conclude this section by the following result (see [11, Appendix A.2]):

▶ **Theorem 27.** *All the DP processors defined in Section 4 are sound.*

5 Universal Computability

Termination is not a *modular* property: given terminating systems $\mathcal{R}_0$ and $\mathcal{R}_1$, we cannot generally conclude that $\mathcal{R}_0 \cup \mathcal{R}_1$ is also terminating. As computability is based on termination, it is not modular either. For example, both $\{\,\mathsf{a} \rightarrow \mathsf{b}\,\}$ and $\{\,\mathsf{f}\ \mathsf{b} \rightarrow \mathsf{f}\ \mathsf{a}\,\}$ are terminating, and $\mathsf{f} : \mathsf{o} \rightarrow \mathsf{o}$ is computable in the second system; yet, combining the two yields $\mathsf{f}\ \mathsf{a} \rightarrow \mathsf{f}\ \mathsf{b} \rightarrow \mathsf{f}\ \mathsf{a} \rightarrow \cdots$, which refutes the termination of the combination and the computability of f.

On the other hand, functions like map and fold are prevalently used; the lack of a modular principle for the termination analysis of higher-order systems involving such functions is painful. Moreover, if such a system is non-terminating, this is seldom attributed to those functions, which are generally considered "terminating" regardless of how they may be called.

In this section, we propose *universal computability*, a concept which corresponds to the termination of a function in all "reasonable" uses. First, we rephrase the notion of a hierarchical combination [27, 28, 29, 6] in terms of LCSTRSs:

▶ **Definition 28.** *An LCSTRS $\mathcal{R}_1$ is called an* extension *of a base system $\mathcal{R}_0$ if the two systems' interpretations of theory symbols coincide over all the theory symbols in common, and function symbols in $\mathcal{R}_0$ are not defined by any rewrite rule in $\mathcal{R}_1$. Given a base system $\mathcal{R}_0$ and an extension $\mathcal{R}_1$ of $\mathcal{R}_0$, the system $\mathcal{R}_0 \cup \mathcal{R}_1$ is called a* hierarchical combination.

In a hierarchical combination, function symbols in the base system can occur in the extension, but cannot be (re)defined. This forms the basis of the modular programming scenario we are interested in: think of the base system as a library containing the definitions of, say, map and fold. We further define a class of extensions to take information hiding into account:

▶ **Definition 29.** *Given an LCSTRS $\mathcal{R}_0$ and a set of function symbols – called* hidden symbols *– in $\mathcal{R}_0$, an extension $\mathcal{R}_1$ of $\mathcal{R}_0$ is called a* public extension *if hidden symbols do not occur in any rewrite rule in $\mathcal{R}_1$.*

Now we present the central definitions of this section:

▶ **Definition 30.** *Given an LCSTRS $\mathcal{R}_0$ with a sort ordering $\succsim$, a term t is called* universally computable *if for each extension $\mathcal{R}_1$ of $\mathcal{R}_0$ and each extension $\succsim'$ of $\succsim$ to sorts in $\mathcal{R}_0 \cup \mathcal{R}_1$ (i.e., $\succsim'$ coincides with $\succsim$ over sorts in $\mathcal{R}_0$), t is $\mathbb{C}$-computable in $\mathcal{R}_0 \cup \mathcal{R}_1$ with $\succsim'$; if a set of hidden symbols in $\mathcal{R}_0$ is also given and the above universal quantification of $\mathcal{R}_1$ is restricted to* public *extensions, such a term t is called* publicly computable*.*

The base system $\mathcal{R}_0$ is called universally computable if all its terms are; it is called publicly computable if all its public *terms – terms that contain no hidden symbol – are.*

With an empty set of hidden symbols, the two notions – universal computability and public computability – coincide. Below we state common properties in terms of public computability.

In summary, we consider passing $\mathbb{C}$-computable arguments to a defined symbol in $\mathcal{R}_0$ the "reasonable" way of calling the function. To establish the universal computability of higher-order functions such as map and fold – i.e., to prove that they are $\mathbb{C}$-computable in *all* relevant hierarchical combinations – we will use SDPs, which are about $\mathbb{C}$-computability.

▶ **Example 31.** The system $\{\,\mathsf{app}\ (\mathsf{lam}\ f) \to f\,\}$ in Section 1 is not universally computable due to the extension $\{\,\mathsf{w}\ x \to \mathsf{app}\ x\ x\,\}$.

5.1 The DP Framework Revisited

To use SDPs for universal – or public – computability, we need a more general version of Theorem 10. We start with defining public chains:

▶ **Definition 32.** *An SDP $f^\sharp\ s_1 \cdots s_m \Rightarrow t^\sharp\ [\varphi \mid L]$ is called public if f is not a hidden symbol. A $(\mathcal{P}, \mathcal{R})$-chain is called public if its first SDP is public.*

Now we state the main result of this section:

▶ **Theorem 33.** *An AFP system $\mathcal{R}_0$ with sort ordering $\succsim$ is publicly computable with respect to a set of hidden symbols in $\mathcal{R}_0$ if there exists no infinite computable $(\mathrm{SDP}(\mathcal{R}_0), \mathcal{R}_0 \cup \mathcal{R}_1)$-chain that is* public *for each public extension $\mathcal{R}_1$ of $\mathcal{R}_0$ and each extension $\succsim'$ of $\succsim$ to sorts in $\mathcal{R}_0 \cup \mathcal{R}_1$.*

While this result is not surprising and its proof (see [11, Appendix A.3]) is standard, it is not obvious how it can be used. The key observation which enables us to use the DP framework for public computability is that among the DP processors in Section 4, only reduction pair processors rely on the rewrite rules of the underlying system $\mathcal{R}$ (depending on how it computes a graph approximation, a graph processor does not have to know all the rewrite rules). Henceforth, we fix a base system $\mathcal{R}_0$, a set of hidden symbols in $\mathcal{R}_0$ and an arbitrary, unknown public extension $\mathcal{R}_1$ of $\mathcal{R}_0$. Now $\mathcal{R}$ is the hierarchical combination $\mathcal{R}_0 \cup \mathcal{R}_1$.

First, we generalize the definition of a DP problem:

▶ **Definition 34.** *A* (universal) DP problem $(\mathcal{P}, \mathtt{p})$ *consists of a set $\mathcal{P}$ of SDPs and a flag* $\mathtt{p} \in \{ \mathfrak{an}, \mathfrak{pu} \}$ *(for* any *or* public*). A DP problem $(\mathcal{P}, \mathtt{p})$ is called* finite *if either (1)* $\mathtt{p} = \mathfrak{an}$ *and there exists no infinite computable $(\mathcal{P}, \mathcal{R}_0 \cup \mathcal{R}_1)$-chain, or (2)* $\mathtt{p} = \mathfrak{pu}$ *and there exists no infinite computable $(\mathcal{P}, \mathcal{R}_0 \cup \mathcal{R}_1)$-chain which is* public*.*

DP processors are defined in the same way as before, now for universal DP problems. The goal is to show that $(\mathrm{SDP}(\mathcal{R}_0), \mathfrak{pu})$ is finite, and the procedure for termination in Section 4 also works here if we change the initialization of Q accordingly.

Next, we review the DP processors presented in Section 4. For each ρ of the original graph, subterm criterion and integer mapping processors, the processor ρ' such that $\rho'(\mathcal{P}, \mathtt{p}) = \{ (\mathcal{P}', \mathfrak{an}) \mid \mathcal{P}' \in \rho(\mathcal{P}) \}$ is sound for universal DP problems. For theory argument processors, we can do better when the input flag is $\mathfrak{pu}$ (when it is $\mathfrak{an}$, we just handle $\mathcal{P}$ in the same way as we do in Section 4 and the output flags are obviously $\mathfrak{an}$): if the subset $\mathcal{P}'$ of $\mathcal{P}$ fixed by a theory argument mapping τ contains all the public SDPs in $\mathcal{P}$, the processor should return the singleton $\{ (\{ p \mid p \in \mathcal{P} \text{ is public} \} \cup \{ \bar{\tau}(p) \mid p \in \mathcal{P} \text{ is not public} \}, \mathfrak{pu}) \}$; otherwise, the pair $\{ (\{ \bar{\tau}(p) \mid p \in \mathcal{P} \}, \mathfrak{an}), (\mathcal{P} \setminus \mathcal{P}', \mathfrak{pu}) \}$. Reduction pair processors require knowledge of the extension $\mathcal{R}_1$ so we do not adapt them.

New Processors. Last, we propose two classes of DP processors that are useful for public computability. Processors of the first class do not actually simplify DP problems; they rather alter their input to allow other DP processors to be applied subsequently.

▶ **Definition 35.** *Given sets $\mathcal{P}_1$ and $\mathcal{P}_2$ of SDPs, $\mathcal{P}_2$ is said to* cover *$\mathcal{P}_1$ if for each SDP $s^\sharp \Rightarrow t^\sharp\ [\varphi_1 \mid L_1] \in \mathcal{P}_1$ and each substitution σ_1 which respects $s^\sharp \Rightarrow t^\sharp\ [\varphi_1 \mid L_1]$, there exist an SDP $s^\sharp \Rightarrow t^\sharp\ [\varphi_2 \mid L_2] \in \mathcal{P}_2$ and a substitution σ_2 such that σ_2 respects $s^\sharp \Rightarrow t^\sharp\ [\varphi_2 \mid L_2]$, $s\sigma_1 = s\sigma_2$ and $t\sigma_1 = t\sigma_2$. A* constraint modification processor *assigns to a DP problem $(\mathcal{P}, \mathtt{p})$ the singleton $\{ (\mathcal{P}', \mathtt{p}) \}$ for some $\mathcal{P}'$ which covers $\mathcal{P}$.*

Now combined with the information of hidden symbols, the DP graph allows us to remove SDPs that are unreachable from any public SDP.

▶ **Definition 36.** *A* reachability processor *assigns to a DP problem $(\mathcal{P}, \mathfrak{pu})$ the singleton $\{ (\{ p \in \mathcal{P} \mid \theta(p)$ is reachable from $\theta(p_0)$ for some public SDP $p_0 \}, \mathfrak{pu}) \}$, given a graph approximation (G_θ, θ) for $\mathcal{P}$.*

These two classes of DP processors are often used together: a constraint modification processor can split an SDP into simpler ones, some of which may be removed by a reachability processor. In our implementation, a constraint modification processor is particularly used to break an SDP $s^\sharp \Rightarrow t^\sharp\ [u \neq v \mid L]$ into two SDPs with logical constraints $u < v$ and $u > v$, respectively (see Example 37); similarly for $s^\sharp \Rightarrow t^\sharp\ [u \vee v \mid L]$.

▶ **Example 37.** Consider an alternative implementation of the factorial function from Example 2, which has SDPs (1) $\mathsf{fact}^\sharp\ n\ k \Rightarrow \mathsf{comp}^\sharp\ k\ ((*)\ n)\ x'\ [n \neq 0]$, (2) $\mathsf{fact}^\sharp\ n\ k \Rightarrow \mathsf{fact}^\sharp\ (n-1)\ (\mathsf{comp}\ k\ ((*)\ n))\ [n \neq 0]$, and (3) $\mathsf{init}^\sharp\ k \Rightarrow \mathsf{fact}^\sharp\ 42\ k$. Assume that fact is a hidden symbol. Note that $(\{ 1, 2, 3 \}, \mathfrak{pu})$ is not finite without this assumption. A constraint modification processor can replace (2) with (4) $\mathsf{fact}^\sharp\ n\ k \Rightarrow \mathsf{fact}^\sharp\ (n-1)\ (\mathsf{comp}\ k\ ((*)\ n))\ [n < 0]$, and (5) $\mathsf{fact}^\sharp\ n\ k \Rightarrow \mathsf{fact}^\sharp\ (n-1)\ (\mathsf{comp}\ k\ ((*)\ n))\ [n > 0]$. A reachability processor can then remove (4). The remaining DP problem $(\{ 1, 3, 5 \}, \mathfrak{pu})$ can easily be handled by a graph processor and an integer mapping processor.

We conclude this section by the following result (see [11, Appendix A.4]):

▶ **Theorem 38.** *All the DP processors defined in Section 5 are sound.*

6 Experiments and Future Work

All the results in this paper have been implemented in our open-source analyzer Cora [20]. We have evaluated Cora on three groups of experiments, and the results are in Table 1.

Table 1 Cora experiment results.

	Custom	STRS	ITRS
Termination	20/28	72/140	69/117
Computability	20/28	66/140	68/117
Wanda	–	105/140	–
AProVE	–	–	102/117

The first group contains examples in this paper and several other LC(S)TRS benchmarks we have collected. The second group contains all the λ-free problems from the higher-order category of TPDB [5]. The third group contains problems from the first-order "integer TRS innermost" category. The computability tests analyze public computability; since there are no hidden symbols in TPDB, the main difference from a termination check is that reduction pair processors are disabled. A full evaluation page is available through the link:

`https://www.cs.ru.nl/~cynthiakop/experiments/mfcs2024`

Unsurprisingly, Cora is substantially weaker than Wanda [16] on unconstrained higher-order TRSs, and AProVE [8] on first-order integer TRSs: this work aims to be a starting point for *combining* higher-order term analysis and theory reasoning, and cannot yet compete with dedicated tools that have had years of development. Nevertheless, we believe that these results show a solid foundation with only a handful of simple techniques.

Future Work. Many of the existing techniques used in the analyses of integer TRSs and higher-order TRSs are likely to be extensible to our setting, leaving many encouraging avenues for further development. We highlight the most important few:

- Usable rules with respect to an argument filtering [10, 17]. To effectively use reduction pairs, being able to discard some rewrite rules is essential (especially for universal computability, if we can discard the unknown ones). Closely related is the adaptation of more reduction pairs such as weakly monotonic algebras [36, 34], tuple interpretations [22, 35] and more sophisticated path orderings [4], all of which have higher-order formulations.
- Transformation techniques, such as narrowing, and chaining dependency pairs together (as used for instance for integer transition systems [8, Secion 3.1]). This could also be a step toward using the constrained DP framework for non-termination.
- Handling the innermost or call-by-value strategy. Several functional languages adopt call-by-value evaluation, and applying this restriction may allow for more powerful analyses. In the first-order DP framework, there is ample work on the innermost strategy to build on (see, e.g., [9, 10]).
- Theory-specific processors for popular theories other than integers, e.g., bit vectors [25].

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1–2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

2 F. Blanqui. Higher-order dependency pairs. In A. Geser and H. Søndergaard, editors, *Proc. WST*, pages 22–26, 2006. doi:10.48550/arXiv.1804.08855.

3 F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive-data-type systems. *TCS*, 272(1–2):41–68, 2002. doi:10.1016/S0304-3975(00)00347-9.

4 F. Blanqui, J.-P. Jouannaud, and A. Rubio. The computability path ordering: the end of a quest. In M. Kaminski and S. Martini, editors, *Proc. CSL*, pages 1–14, 2008. doi:10.1007/978-3-540-87531-4_1.

5 Community. The termination problem database (TPDB). URL: https://github.com/TermCOMP/TPDB.

6 N. Dershowitz. Hierarchical termination. In N. Dershowitz and N. Lindenstrauss, editors, *Proc. CTRS*, pages 89–105, 1995. doi:10.1007/3-540-60381-6_6.

7 C. Fuhs and C. Kop. A static higher-order dependency pair framework. In L. Caires, editor, *Proc. ESOP*, pages 752–782, 2019. doi:10.1007/978-3-030-17184-1_27.

8 J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing program termination and complexity automatically with AProVE. *JAR*, 58:3–31, 2017. doi:10.1007/s10817-016-9388-y.

9 J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: combining techniques for automated termination proofs. In F. Baader and A. Voronkov, editors, *Proc. LPAR*, pages 301–331, 2005. doi:10.1007/978-3-540-32275-7_21.

10 J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *JAR*, 37:155–203, 2006. doi:10.1007/s10817-006-9057-7.

11 L. Guo, K. Hagens, C. Kop, and D. Vale. Higher-order constrained dependency pairs for (universal) computability. Technical report, Radboud University, 2024. doi:10.48550/arXiv.2406.19379.

12 L. Guo and C. Kop. Higher-order LCTRSs and their termination. In S. Weirich, editor, *Proc. ESOP*, pages 331–357, 2024. doi:10.1007/978-3-031-57267-8_13.

13 N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In V. van Oostrom, editor, *Proc. RTA*, pages 249–268, 2004. doi:10.1007/978-3-540-25979-4_18.

14 J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In G. Longo, editor, *Proc. LICS*, pages 402–411, 1999. doi:10.1109/LICS.1999.782635.

15 C. Kop. Termination of LCTRSs. In J. Waldmann, editor, *Proc. WST*, pages 59–63, 2013. doi:10.48550/arXiv.1601.03206.

16 C. Kop. WANDA – A higher order termination tool. In Z. M. Ariola, editor, *Proc. FSCD*, pages 36:1–36:19, 2020. doi:10.4230/LIPIcs.FSCD.2020.36.

17 C. Kop. Cutting a proof into bite-sized chunks: incrementally proving termination in higher-order term rewriting. In A. P. Felty, editor, *Proc. FSCD*, pages 1:1–1:17, 2022. doi:10.4230/LIPIcs.FSCD.2022.1.

18 C. Kop. A weakly monotonic, logically constrained, HORPO-variant. Technical report, Radboud University, 2024. doi:10.48550/arXiv.2406.18493.

19 C. Kop and N. Nishida. Term rewriting with logical constraints. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Proc. FroCoS*, pages 343–358, 2013. doi:10.1007/978-3-642-40885-4_24.

20 C. Kop and D. Vale. The Cora analyzer. URL: https://github.com/hezzel/cora.

21 C. Kop and D. Vale. hezzel/cora: artifact for MFCS 2024. doi:10.5281/zenodo.12551027.

22 C. Kop and D. Vale. Tuple interpretations for higher-order complexity. In N. Kobayashi, editor, *Proc. FSCD*, pages 31:1–31:22, 2021. doi:10.4230/LIPIcs.FSCD.2021.31.

23 C. Kop and F. van Raamsdonk. Dynamic dependency pairs for algebraic functional systems. *LMCS*, 8(2):10:1–10:51, 2012. doi:10.2168/lmcs-8(2:10)2012.

24 K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting. *AAECC*, 18(5):407–431, 2007. doi:10.1007/s00200-007-0046-9.

25 A. Matsumi, N. Nishida, M. Kojima, and D. Shin. On singleton self-loop removal for termination of LCTRSs with bit-vector arithmetic. In A. Yamada, editor, *Proc. WST*, 2023. doi:10.48550/arXiv.2307.14094.

26 P. W. O'Hearn. Continuous reasoning: scaling the impact of formal methods. In A. Dawar and E. Grädel, editors, *Proc. LICS*, pages 13–25, 2018. doi:10.1145/3209108.3209109.

27 M. R. K. K. Rao. Completeness of hierarchical combinations of term rewriting systems. In R. K. Shyamasundar, editor, *Proc. FSTTCS*, pages 125–138, 1993. doi:10.1007/3-540-57529-4_48.

28 M. R. K. K. Rao. Simple termination of hierarchical combinations of term rewriting systems. In M. Hagiya and J. C. Mitchell, editors, *Proc. TACS*, pages 203–223, 1994. doi:10.1007/3-540-57887-0_97.

29 M. R. K. K. Rao. Semi-completeness of hierarchical and super-hierarchical combinations of term rewriting systems. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *Proc. CAAP*, pages 379–393, 1995. doi:10.1007/3-540-59293-8_208.

30 M. Sakai and K. Kusakari. On dependency pair method for proving termination of higher-order rewrite systems. *IEICE Trans. Inf. Syst.*, E88-D(3):583–593, 2005. doi:10.1093/ietisy/e88-d.3.583.

31 M. Sakai, Y. Watanabe, and T. Sakabe. An extension of the dependency pair method for proving termination of higher-order rewrite systems. *IEICE Trans. Inf. Syst.*, E84-D(8):1025–1032, 2001. URL: https://search.ieice.org/bin/summary.php?id=e84-d_8_1025.

32 R. Tarjan. Depth-first search and linear graph algorithms. *SICOMP*, 1(2):146–160, 1972. doi:10.1137/0201010.

33 R. Thiemann. *The DP Framework for Proving Termination of Term Rewriting.* PhD thesis, RWTH Aachen University, 2007. URL: http://cl-informatik.uibk.ac.at/users/thiemann/paper/diss.pdf.

34 J. C. van de Pol. *Termination of Higher-Order Rewrite Systems.* PhD thesis, Utrecht University, 1996. URL: https://www.cs.au.dk/~jaco/papers/thesis.pdf.

35 A. Yamada. Tuple interpretations for termination of term rewriting. *JAR*, 66:667–688, 2022. doi:10.1007/s10817-022-09640-4.

36 H. Zantema. Termination of term rewriting: interpretation and type elimination. *JSC*, 17(1):23–50, 1994. doi:10.1006/jsco.1994.1003.

Parameterized Vertex Integrity Revisited

Tesshu Hanaka
Department of Informatics, Kyushu University, Fukuoka, Japan

Michael Lampis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Manolis Vasilakis
Université Paris-Dauphine, PSL University, CNRS UMR7243, LAMSADE, Paris, France

Kanae Yoshiwatari
Department of Mathematical Informatics, Nagoya University, Aichi, Japan

Abstract

Vertex integrity is a graph parameter that measures the connectivity of a graph. Informally, its meaning is that a graph has small vertex integrity if it has a small separator whose removal disconnects the graph into connected components which are themselves also small. Graphs with low vertex integrity are very structured; this renders many hard problems tractable and has recently attracted interest in this notion from the parameterized complexity community. In this paper we revisit the NP-complete problem of computing the vertex integrity of a given graph from the point of view of structural parameterizations. We present a number of new results, which also answer some recently posed open questions from the literature. Specifically, we show that unweighted vertex integrity is W[1]-hard parameterized by treedepth; we show that the problem remains W[1]-hard if we parameterize by feedback edge set size (via a reduction from a Bin Packing variant which may be of independent interest); and complementing this we show that the problem is FPT by max-leaf number. Furthermore, for weighted vertex integrity, we show that the problem admits a single-exponential FPT algorithm parameterized by vertex cover or by modular width, the latter result improving upon a previous algorithm which required weights to be polynomially bounded.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Parameterized Complexity, Treedepth, Vertex Integrity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.58

Related Version *Full Version*: https://arxiv.org/abs/2402.09971

Funding This work is partially supported by ANR projects ANR-21-CE48-0022 (S-EX-AP-PE-AL) and ANR-18-CE40-0025-01 (ASSK), and JSPS KAKENHI Grant Numbers JP21K17707, JP21H05852, JP22H00513, and JP23H04388.

1 Introduction

The *vertex integrity* of a graph is a vulnerability measure indicating how easy it is to break down the graph into small pieces. More precisely, the vertex integrity $\mathrm{vi}(G)$ of a graph G is defined as $\mathrm{vi}(G) = \min_{S \subseteq V(G)}\{|S| + \max_{D \in \mathsf{cc}(G-S)} |D|\}$, that is, to calculate the vertex integrity of a graph we must find a separator that minimizes the size of the separator itself plus the size of the largest remaining connected component. Intuitively, a graph has low vertex integrity not only when it contains a small separator, but more strongly when it contains a small separator such that its removal leaves a collection of small connected components.

Vertex integrity was first introduced more than thirty years ago by Barefoot et al. [1], but has recently received particular attention from the parameterized complexity community since it can be considered as a very natural structural parameter: when a graph has vertex

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 58; pp. 58:1–58:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

integrity k, large classes of NP-hard problems admit FPT[1] algorithms with running times of the form $f(k)n^{\mathcal{O}(1)}$ [27]. Note that vertex integrity has a clear relationship with other, well-known structural parameters (see also Figure 1): it is more restrictive than treedepth, pathwidth, and treewidth (all these parameters are upper-bounded by vertex integrity) but more general than vertex cover (a graph of vertex cover k has vertex integrity at most $k+1$). "Price of generality" questions, where one seeks to discover for a given problem the most general parameter for which an FPT algorithm is possible, are a central topic in structural parameterized complexity, and vertex integrity therefore plays a role as a natural stepping stone in the hierarchy of standard parameters [3, 15, 16, 20, 22, 27].

The investigation of the parameterized complexity aspects of vertex integrity is, therefore, an active field of research, but it is important to remember that a prerequisite for any such parameter to be useful is that it should be tractable to calculate the parameter itself (before we try to use it to solve other problems). Since, unsurprisingly, computing the vertex integrity exactly is NP-complete [6], in this paper we focus on this problem from the point of view of parameterized complexity. We consider both the unweighted, as well as a natural weighted variant of the problem. Formally, we want to solve the following:

Unweighted (Weighted) Vertex Integrity

Instance: A graph G (with binary vertex weights $w : V(G) \to \mathbb{Z}^+$), an integer k.

Goal: Determine whether $\mathrm{vi}(G) \leq k$ ($\mathrm{wvi}(G) \leq k$).

The point of view we adopt is that of structural parameterized complexity, where vertex integrity is the target problem we are trying to solve, and not necessarily the parameter. Instead, we parameterize by standard structural width measures, such as variations of treewidth. The questions we would like to address are of several forms:

1. For which structural parameters is it FPT to compute the vertex integrity?
2. For which such parameters is it possible to obtain an FPT algorithm with single-exponential complexity?
3. For which parameters can the weighted version of the problem be handled as well as the unweighted version?

To put these questions in context, we recall some facts from the state of the art. When the parameter k is the vertex integrity itself, Fellows and Stueckle show an $\mathcal{O}(k^{3k}n)$-time algorithm for Unweighted Vertex Integrity [13], and later Drange et al. proposed an $\mathcal{O}(k^{k+1}n)$-time algorithm even for Weighted Vertex Integrity [9], so this problem is FPT. More recently, Gima et al. [21] took up the study of vertex integrity in the same structurally parameterized spirit as the one we adopt here and presented numerous results which already give some answers to the questions we posed above. In particular, for the first question they showed that Unweighted Vertex Integrity is W[1]-hard by pathwidth (and hence by treewidth); for the second question they showed that the problem admits a single-exponential algorithm for parameter modular-width; and for the third question they showed that the problem is (weakly) NP-hard on sub-divided stars, which rules out FPT algorithms for most structural parameters.

[1] We assume the reader is familiar with the basics of parameterized complexity, as given e.g. in [7]. We give precise definitions of all parameters in the next section.

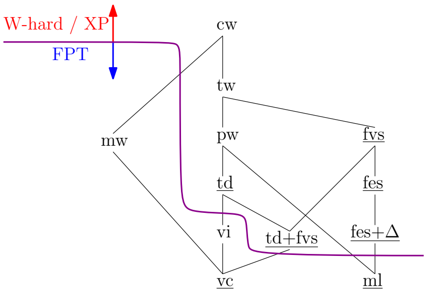


Figure 1 The parameterized complexity of Unweighted Vertex Integrity, with the underlined parameters indicating our results. A connection between two parameters implies that the one above generalizes the one below; that is, the one above is upper-bounded by a function of the one below. Regarding the presented parameters, vc, vi, td, pw, tw, cw, ml, fes, fvs, Δ, and mw stand for vertex cover, vertex integrity, treedepth, pathwidth, treewidth, cliquewidth, max-leaf number, feedback edge set, feedback vertex set, maximum degree, and modular width respectively. All of our FPT algorithms have single-exponential parametric dependence, while the ones for vc and mw extend to the weighted case as well.

Our results. Although the results of [21] are rather comprehensive, they leave open several important questions about the complexity of vertex integrity. In this paper we resolve the questions explicitly left open by [21] and go on to present several other results that further clarify the picture for vertex integrity. In particular, our results are as follows (see also Figure 1):

The first question we tackle is an explicit open problem from [21]: is Unweighted Vertex Integrity FPT parameterized by treedepth? This is a very natural question, because treedepth is the most well-known parameter that sits between pathwidth, where the problem is W[1]-hard by [21], and vertex integrity itself, where the problem is FPT. We resolve this question via a reduction from Bounded Degree Vertex Deletion, showing that Unweighted Vertex Integrity is W[1]-hard for treedepth (Theorem 2).

A second question left open by [21] is the complexity of Unweighted Vertex Integrity for parameter feedback vertex set. Taking a closer look at our reduction from Bounded Degree Vertex Deletion, which is known to be W[1]-hard for this parameter, we observe that it also settles this question, showing that Unweighted Vertex Integrity is also hard. However, in this case we are motivated to dig a little deeper and consider a parameter, feedback edge set, which is a natural restriction of feedback vertex set and typically makes most problems FPT. Our second result is to show that Unweighted Vertex Integrity is in fact W[1]-hard even when parameterized by feedback edge set and the maximum degree of the input graph (Theorem 7). We achieve this via a reduction from Unary Bin Packing parameterized by the number of bins, which is W[1]-hard [23]. An aspect of our reduction which may be of independent interest is that we use a variant of Unary Bin Packing where we are given a choice of only two possible bins per item (we observe that the reduction of [23] applies to this variant).

We complement these mostly negative results with a fixed-parameter tractability result for a more restrictive parameter: we show that Unweighted Vertex Integrity is FPT by max-leaf number (Theorem 10) indeed by a single-exponential FPT algorithm. Note that when a graph has bounded max-leaf number, then it has bounded degree and bounded feedback edge set number, therefore this parameterization is a special case of the one considered in Theorem 7. Hence, this positive result closely complements the problem's hardness in the more general case.

Moving on, we consider the parameterization by modular width, and take a second look at the $2^{\mathcal{O}(\mathrm{mw})}n^{\mathcal{O}(1)}$ algorithm provided by [21], which is able to handle the weighted case of the problem, but only for polynomially-bounded weights. Resolving another open problem posed by [21], we show how to extend their algorithm to handle the general case of weights encoded in binary (Theorem 13).

Finally, we ask the question of whether a single-exponential FPT algorithm is possible for parameters other than max-leaf and modular width. We answer this affirmatively for vertex cover, even in the weighted case (Theorem 16), obtaining a faster and simpler algorithm for the unweighted case (Theorem 14).

Related work. The concept of vertex integrity is natural enough that it has appeared in many slight variations under different names in the literature. We mention in particular, the fracture number [10], which is the minimum k such that it is possible to delete k vertices from a graph so that all remaining components have size at most k, and the starwidth [29], which is the minumum width of a tree decomposition that is a star. Both of these are easily seen to be at most a constant factor away from vertex integrity. Similarly, the safe set number [2, 14] seeks a separator such that every component of the separator is only connected to smaller components. These concepts are so natural that sometimes they are used as parameters without an explicit name, for example [4] uses the parameter "size of a deletion set to a collection of components of bounded size". As observed by [21], despite these similarities, sometimes computing these parameters can have different complexity, especially when weights are allowed. Another closely related computational problem, that we also use, is the Component Order Connectivity [9] problem, where we are given explicit distinct bounds on the size of the separator sought and the allowed size of the remaining components.

2 Preliminaries

Throughout the paper we use standard graph notation [8] and assume familiarity with the basic notions of parameterized complexity [7]. All graphs considered are undirected without loops. Given a graph G, Δ denotes its maximum degree; if we are additionally given $S \subseteq V(G)$, $G[S]$ denotes the subgraph induced by S, while $G - S$ denotes $G[V(G) \setminus S]$. Furthermore, given a weight function $w : V(G) \to \mathbb{Z}^+$, $w(S)$ denotes the sum of the weights of the vertices of S, that is $w(S) = \sum_{s \in S} w(s)$. For $x, y \in \mathbb{Z}$, let $[x, y] = \{z \in \mathbb{Z} : x \leq z \leq y\}$, while $[x] = [1, x]$. For a set of integers $S \subseteq \mathbb{Z}^+$, let $\Sigma(S)$ denote the sum of its elements, i.e. $\Sigma(S) = \sum_{s \in S} s$, while $\binom{S}{c}$ denotes the set of subsets of S of size c, i.e. $\binom{S}{c} = \{S' \subseteq S : |S'| = c\}$. Proofs of statements marked with $(\star)$ are presented in the full version of the paper.

2.1 Vertex Integrity

For a vertex-weighted graph G with $w : V(G) \to \mathbb{Z}^+$, we define its *weighted vertex integrity*, denoted by $\mathrm{wvi}(G)$, as

$$\mathrm{wvi}(G) = \min_{S \subseteq V(G)} \left\{ w(S) + \max_{D \in \mathtt{cc}(G-S)} w(D) \right\},$$

where $\mathtt{cc}(G - S)$ is the set of connected components of $G - S$. A set S such that $w(S) + \max_{D \in \mathtt{cc}(G-S)} w(D) \leq k$ is called a $\mathrm{wvi}(k)$-set. The *vertex integrity* of an unweighted graph G, denoted by $\mathrm{vi}(G)$, is defined in an analogous way, by setting $w(v) = 1$ for all $v \in V(G)$. In that case, $S \subseteq V(G)$ is a $\mathrm{vi}(k)$-set if $|S| + \max_{D \in \mathtt{cc}(G-S)} |D| \leq k$.

A vertex $v \in S$ is called *redundant* if at most one connected component of $G - S$ contains neighbors of v. A set $S \subseteq V(G)$ is *irredundant* if S contains no redundant vertex. Notice that it suffices to only search for irredundant wvi(k)-sets when solving Vertex Integrity, since if $v \in S$ is redundant and S is a wvi(k)-set, that is the case for set $S \setminus \{v\}$ as well.

▶ **Proposition 1** ([9, 21])**.** *A graph with a* wvi(k)*-set has an irredundant* wvi(k)*-set.*

2.2 Graph Parameters

We use several standard graph parameters, so we recall here their definitions and known relations between them. A graph G has *feedback vertex* (respectively *edge*) *set* k if there exists a set of k vertices (respectively edges) such that removing them from G destroys all cycles. We use fvs(G) and fes(G) to denote these parameters. Note that even though computing fvs(G) is NP-complete [24], in all connected graphs with m edges and n vertices fes(G) $= m - n + 1$. The *vertex cover* of a graph G, denoted by vc(G), is the size of the smallest set whose removal destroys all edges. The *treedepth* of a graph G can be defined recursively as follows: $\mathrm{td}(K_1) = 1$; if G is disconnected td(G) is equal to the maximum of the treedepth of its connected components; otherwise $\mathrm{td}(G) = \min_{v \in V(G)} \mathrm{td}(G - v) + 1$. The *max-leaf number* of a graph G, denoted by ml(G), is the maximum number of leaves of any spanning tree of G.

A module of a graph $G = (V, E)$ is a set of vertices $M \subseteq V$ such that for all $x \in V \setminus M$ we have that x is either adjacent to all vertices of M or to none. The *modular width* of a graph $G = (V, E)$ ([17, 18]) is the smallest integer k such that, either $|V| \leq k$, or V can be partitioned into at most $k' \leq k$ sets $V_1, \ldots, V_{k'}$, with the following two properties: (i) for all $i \in [k']$, V_i is a module of G, (ii) for all $i \in [k']$, $G[V_i]$ has modular width at most k.

Let us also briefly explain the relations depicted in Figure 1. Clearly, for all G, we have fvs(G) $\leq$ fes(G), because we can remove from the graph one endpoint of each edge of the feedback edge set. It is known that if a graph has ml(G) $= k$, then G contains at most $\mathcal{O}(k)$ vertices of degree 3 or more (Lemma 8), and clearly such a graph has maximum degree at most k. Since vertices of degree at most 2 are irrelevant for fes, we conclude that the parameterization by ml is more restrictive than that for fes $+ \Delta$. It is also not hard to see that for all G, td(G) $\leq$ vi(G) $\leq$ vc(G) $+ 1$. Note also that even though vc can be seen as a parameter more restrictive than mw, when a graph has vertex cover k, the best we can say is that its modular width is at most $2^k + k$ [26]. As a result, the algorithm of Theorem 13 does not imply a single-exponential FPT algorithm for parameter vc (but does suffice to show that the problem is FPT). We also note that the reductions of Theorem 2 (for td) and Theorem 7 (for fes $+ \Delta$) are complementary and cannot be subsumed by a single reduction. The reason for this is that if in a graph we bound simultaneously the treedepth and the maximum degree, then we actually bound the size of the graph (rendering all problems FPT).

3 Treedepth

Our main result in this section is the following theorem, resolving a question of [21]. We obtain it via a parameter-preserving reduction from Bounded Degree Vertex Deletion, which is known to be W[1]-hard parameterized by treedepth plus feedback vertex set [19].

▶ **Theorem 2.** Unweighted Vertex Integrity *is W[1]-hard parameterized by* td + fvs*. Moreover, it cannot be solved in time* $f(\mathrm{td})n^{o(\mathrm{td})}$ *under the ETH.*

Proof. First we define the closely related Component Order Connectivity problem: given a graph G as well as integers ℓ and p, we want to determine whether there exists $S \subseteq V(G)$ such that $|S| \leq p$ and all components of $G - S$ have size at most ℓ. We will proceed in two steps: we first reduce Bounded Degree Vertex Deletion to Component Order Connectivity, and then employ the reduction of [21] that reduces the latter to Unweighted Vertex Integrity. Notice that [21, Lemma 4.4] creates an equivalent instance of Unweighted Vertex Integrity by solely adding disjoint stars and leaves in the vertices of the initial graph, therefore it suffices to prove the statement for Component Order Connectivity instead.

We give a parameterized reduction from Bounded Degree Vertex Deletion, which is W[1]-hard by treedepth plus feedback vertex set number [19] and cannot be solved in time $f(\text{td})n^{o(\text{td})}$ under the ETH [28]. In Bounded Degree Vertex Deletion we are given a graph $G = (V, E)$ and two integers k and d, and we are asked to determine whether there exists $S \subseteq V$ of size $|S| \leq k$ such that the maximum degree of $G - S$ is at most d. In the following, let $n = |V(G)|$ and $m = |E(G)|$.

Given an instance (G, k, d) of Bounded Degree Vertex Deletion, we construct an equivalent instance (G', ℓ, p) of Component Order Connectivity. We construct G' from G as follows: We subdivide every edge $e = \{u, v\} \in E(G)$ three times, thus replacing it with a path on vertices u, u_v, y_e, v_u, and v, where $T_e = \{u_v, y_e, v_u\}$. Next, we attach $d - 1$ leaves to y_e (see Figure 2). This concludes the construction of G'. Notice that the subdivision of the edges three times and the attachment of pendant vertices does not change the feedback vertex set number, while the treedepth is only increased by an additive constant. Thus, it holds that $\text{fvs}(G') = \text{fvs}(G)$ and $\text{td}(G') = \text{td}(G) + \mathcal{O}(1)$.

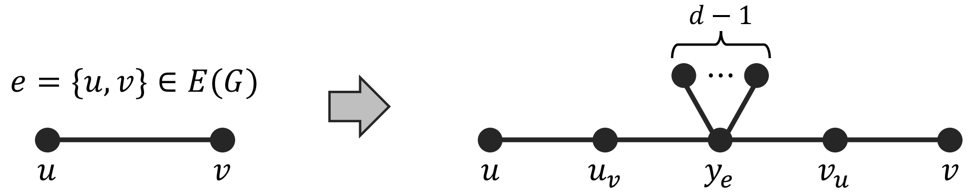


Figure 2 Edge gadget for edge $e = \{u, v\} \in E(G)$.

In the following, we show that (G, k, d) is a yes-instance of Bounded Degree Vertex Deletion if and only if (G', ℓ, p) is a yes-instance of Component Order Connectivity, where $\ell = d + 1$ and $p = k + m$.

For the forward direction, let S be a set of vertices of size at most k such that the maximum degree of $G - S$ is at most d. We will construct a set $S' \subseteq V(G')$ such that $|S'| \leq p$ and every connected component of $G' - S'$ has size at most ℓ. Initially set $S' = S$. Then, add one vertex to S' per edge $e = \{u, v\} \in E(G)$ as follows. If $u, v \in S$ or $u, v \notin S$, we add y_e to S'. Otherwise, if $u \in S$ and $v \notin S$, we add v_u to S'; symmetrically, if $u \notin S$ and $v \in S$, we add u_v instead. Notice that $|S'| = |S| + m \leq k + m = p$, therefore it suffices to show that the size of each connected component of $G' - S'$ is at most $\ell = d + 1$.

Consider a connected component D of $G' - S'$. Assume that D does not contain any vertices of $V \setminus S$. If D is a leaf it holds that $|D| \leq d + 1$. Alternatively, D is a subgraph of the graph induced by u_v (or v_u), y_e, and its attached leaves, for some $e = \{u, v\} \in E(G)$, in which case $|D| \leq d + 1$. Now assume that D contains $u \in V \setminus S$. Notice that u is the only vertex of $V \setminus S$ present in D, since $S' \cap T_e \neq \emptyset$ for all $e \in E(G)$. Moreover, let $N(u) \setminus S = \{u_i : i \in [q]\}$ denote its neighbors in $G - S$, where $q \leq d$ since the maximum degree of $G - S$ is at most d. In that case, it follows that D consists of u, as well as the vertices u_{u_i} for all $i \in [q]$. Consequently, $|D| = q + 1 \leq d + 1$.

For the converse direction, assume there exists $S' \subseteq V(G')$ such that $|S'| \leq p = k + m$ and $|D| \leq \ell = d + 1$, for all connected components $D \in \mathsf{cc}(G' - S')$. Assume that S' does not contain any leaves; if it does, substitute them with their single neighbor. Moreover, $S' \cap T_e \neq \emptyset$ for all $e \in E(G)$, since otherwise $G' - S'$ has a component of size at least $d + 2 > \ell$, which is a contradiction. Assume without loss of generality that $|S' \cap T_e| = 1$, for all $e = \{u, v\} \in E(G)$; if that is not the case, there is always a vertex of $\{u_v, v_u\}$, say u_v, such that $u_v \in S'$ and $S' \cap \{y_e, v_u\} \neq \emptyset$, in which case one may consider the deletion set $(S' \cup \{u\}) \setminus \{u_v\}$ instead (the argument is symmetric in case $v_u \in S'$).

Let $S = S' \cap V$, where $|S| \leq k$. We will prove that $G - S$ has maximum degree at most d. Let D_u denote the connected component of $G' - S'$ that contains $u \in V \setminus S'$; in fact this is the only vertex of $V \setminus S'$ present in D_u, since $S' \cap T_e \neq \emptyset$ for all $e \in E(G)$. Notice that for all $e = \{u, v\} \in E(G)$ where $u, v \notin S'$, it holds that $y_e \in S'$: if that were not the case, then either D_u or D_v contains at least $d + 2 > \ell$ vertices, due to $\{u, u_v, y_e\}$ or $\{v, v_u, y_e\}$ and the leaves of y_e respectively. For $u \in V \setminus S$, let $N(u) \setminus S = \{u_i : i \in [q]\}$, for some integer q, denote its neighbors in $G - S$, where $e_i = \{u, u_i\} \in E(G)$ for $i \in [q]$. It suffices to show that $q \leq d$. Assume that this is not the case, i.e. $q > d$. Then, since $S' \cap T_{e_i} = \{y_{e_i}\}$ for $i \in [q]$, it follows that D_u contains vertices u and u_{u_i}, therefore $|D_u| \geq q + 1 > d + 1 = \ell$, which is a contradiction. Consequently, $|N(u) \setminus S| \leq d$ for all $u \in V \setminus S$, i.e. $G - S$ has maximum degree d. ◀

4 Feedback Edge Set plus Maximum Degree

In this section we prove that Vertex Integrity is W[1]-hard parameterized by fes + Δ. Since our reduction is significantly more involved than the one of Theorem 2, we proceed in several steps. We start from an instance of Unary Bin Packing where the parameter is the number of bins and consider a variant where we are also supplied in the input, for each item, a choice of two possible bins to place it. We first observe that the reduction of [23] shows that this variant is also W[1]-hard. We then reduce this to a *semi-weighted* version of Vertex Integrity, where placing a vertex in the separator always costs 1, but vertices have weights which they contribute to their components if they are not part of the separator, and where we are prescribed the size of the separator to use (this is called the Component Order Connectivity problem). Subsequently, we show how to remove the weights and the prescription on the separator size to obtain hardness for Vertex Integrity.

4.1 Preliminary Tools

Unary Bin Packing. Given a set $S = \{s_1, \dots, s_n\}$ of integers in unary (i.e. $s_i = \mathcal{O}(n^c)$ for some constant c), as well as $k \in \mathbb{Z}^+$, Unary Bin Packing asks whether we can partition S into k subsets $S_1, \dots, S_k$, such that $\Sigma(S_i) = \Sigma(S)/k$, for all $i \in [k]$. This problem is well known to be W[1]-hard parameterized by the number of bins k [23]. We formally define a restricted version where every item is allowed to choose between *exactly two* bins, and by delving deeper into the proof of [23] we observe that an analogous hardness result follows.

Restricted Unary Bin Packing

Instance: A set $S = \{s_1, \dots, s_n\}$ of integers in unary, $k \in \mathbb{Z}^+$, as well as a function $f : S \to \binom{[k]}{2}$.

Goal: Determine whether we can partition S into k subsets $S_1, \dots, S_k$, such that for all $i \in [k]$ it holds that (i) $\Sigma(S_i) = \Sigma(S)/k$, and (ii) $\forall s \in S_i$, $i \in f(s)$.

▶ **Theorem 3** ($\star$)**.** Restricted Unary Bin Packing *is W[1]-hard parameterized by the number of bins.*

Semi-weighted problems. In this section we study semi-weighted versions of Component Order Connectivity and Vertex Integrity, which we first formally define. Then, we prove that the first can be reduced to the latter, while retaining the size of the minimum feedback edge set and the maximum degree.

Semi-Weighted Component Order Connectivity

Instance: A vertex-weighted graph $G = (V, E, w)$, as well as integers $\ell, p \in \mathbb{Z}^+$.
Goal: Determine whether there exists $S \subseteq V$ of size $|S| \leq p$, such that $w(D) \leq \ell$ for all $D \in \mathtt{cc}(G - S)$.

Semi-Weighted Vertex Integrity

Instance: A vertex-weighted graph $G = (V, E, w)$, as well as an integer $\ell \in \mathbb{Z}^+$.
Goal: Determine whether there exists $S \subseteq V$ such that $|S| + w(D) \leq \ell$ for all $D \in \mathtt{cc}(G - S)$.

▶ **Theorem 4** ($\star$)**.** Semi-Weighted Component Order Connectivity *parameterized by* $\text{fes} + \Delta$ *is fpt-reducible to* Semi-Weighted Vertex Integrity *parameterized by* $\text{fes} + \Delta$.

4.2 Hardness Result

Using the results of Section 4.1, we proceed to proving the main theorem of this section. To this end, we present a reduction from Restricted Unary Bin Packing to Semi-Weighted Component Order Connectivity such that for the produced graph G it holds that $\text{fes}(G) + \Delta(G) \leq f(k)$, for some function f and k denoting the number of bins of the Restricted Unary Bin Packing instance.

We first provide a sketch of our reduction. For every bin of the Restricted Unary Bin Packing instance, we introduce a clique of $\mathcal{O}(k)$ heavy vertices, and then connect any pair of such cliques via two paths. The weights are set in such a way that an optimal solution will only delete vertices from said paths. In order to construct a path for a pair of bins, we compute the set of all subset sums of the items that can be placed in these two bins, and introduce a vertex of medium weight per such subset sum. Moreover, every such vertex corresponding to subset sum s is preceded by exactly s vertices of weight 1. An optimal solution will cut the path in such a way that the number of vertices of weight 1 will be partitioned between the two bins, encoding the subset sum of the elements that are placed on each bin. The second path that we introduce has balancing purposes, allowing us to exactly count the number of vertices of medium weight that every connected component will end up with.

▶ **Theorem 5** ($\star$)**.** Semi-Weighted Component Order Connectivity *is W[1]-hard parameterized by* $\text{fes} + \Delta$.

By Theorems 4 and 5, the hardness of Semi-Weighted Vertex Integrity follows.

▶ **Theorem 6.** Semi-Weighted Vertex Integrity *is W[1]-hard parameterized by* $\text{fes} + \Delta$.

Moreover, we can easily reduce an instance (G, w, k) of Semi-Weighted Vertex Integrity to an instance (G', k) of Unweighted Vertex Integrity by attaching a path on $w(v) - 1$ vertices to each vertex v (we assume that $w(v) \leq k$, otherwise v belongs to the deletion set). Thus, $\text{fes}(G') = \text{fes}(G)$ and $\Delta(G') = \Delta(G) + 1$, and due to Theorem 6 the main result of this section follows.

▶ **Theorem 7.** *Unweighted Vertex Integrity is W[1]-hard parameterized by* $\text{fes} + \Delta$.

5 Max-Leaf Number

In this section, we consider Unweighted Vertex Integrity parameterized by the max-leaf number. For a connected graph G we denote by $\text{ml}(G)$ the maximum number of leaves of any spanning tree of G. This is a well-studied but very restricted parameter [11, 12, 26]. In particular, it is known that if a graph G has $\text{ml}(G) \leq k$, then in fact G is a subdivision of a graph on $\mathcal{O}(k)$ vertices [25]. We are motivated to study this parameter because in a sense it lies close to the intractability boundary established in Section 4. Observe that if a graph is a sub-division of a graph on k vertices, then it has maximum degree at most k and feedback edge set at most k^2; however, graphs of small feedback edge set and small degree do not necessarily have small max-leaf number (consider a long path where we attach a leaf to each vertex). Interestingly, the graphs we construct in Section 4.2 *do* have small max-leaf number, if we consider semi-weighted instances. However, adding the necessary simple gadgets in order to simulate weights increases the max-leaf number of the graphs of our reduction. It is thus a natural question whether this is necessary. In this section, we show that indeed this is inevitable, as Vertex Integrity is FPT parameterized by ml.

We start with a high-level overview of our approach. As mentioned, we will rely on the result of Kleitman and West [25] who showed that if a graph $G = (V, E)$ has $\text{ml}(G) \leq k$, then there exists a set X of size $|X| = \mathcal{O}(k)$ such that all vertices of $V \setminus X$ have degree at most 2. Our main tool is a lemma (Lemma 9) which allows us to "rotate" solutions: whenever we have a cycle in our graph, we can, roughly speaking, exchange every vertex of S in the cycle with the next vertex, until we reach a point where our solution removes strictly more vertices of X. We therefore guess the largest intersection of an optimal separator with X, and can now assume that in every remaining cycle, the separator S is not using any vertices. This allows us to simplify the graph in a way that removes all cycles and reduces the case to a tree, which is polynomial-time solvable.

Let us now give more details. We first recall the result of [25].

▶ **Lemma 8** ($\star$). *In any graph G, the set X of vertices of degree at least 3 has size at most* $|X| \leq 12\text{ml}(G) + 32$.

We will solve Component Order Connectivity: for a given ℓ we want to calculate the minimum number of vertices p such that there exists a separator S of size at most p with all components of $G - S$ having size at most ℓ. To obtain an algorithm for Unweighted Vertex Integrity, we will try all possible values of ℓ and select the solution which minimizes $\ell + p$.

Our main lemma is now the following:

▶ **Lemma 9** ($\star$). *Let $G = (V, E)$ be a graph and X be a set of vertices such that all vertices of $V \setminus X$ have degree at most 2 in G. For all positive integers ℓ, p, if there exists a separator S of size at most p such that all components of $G - S$ have size at most ℓ, then there exists such a separator S that also satisfies the following property: for every cycle C of G with $C \cap X \neq \emptyset$ we either have $C \cap S = \emptyset$ or $C \cap X \cap S \neq \emptyset$.*

We are now ready to state the main result of this section.

▶ **Theorem 10** (⋆). Unweighted Vertex Integrity *can be solved in time* $2^{\mathcal{O}(\mathrm{ml})}n^{\mathcal{O}(1)}$.

6 Modular Width

In this section we revisit an algorithm of [21] establishing that Weighted Vertex Integrity can be solved in time $2^{\mathcal{O}(\mathrm{mw})}n^{\mathcal{O}(1)}$, on graphs of modular width mw, but only if weights are polynomially bounded in n (or equivalently, if weights are given in unary). It was left as an explicit open problem in [21] whether this algorithm can be extended to the case where weights are given in binary and can therefore be exponential in n. We resolve this problem positively, by showing how the algorithm of [21] can be modified to work also in this case, without a large increase in its complexity.

The high-level idea of the algorithm of [21] is to perform dynamic programming to solve the related Weighted Component Order Connectivity problem. In this problem we are given a target component weight ℓ and a deletion budget p and are asked if it is possible to delete from the graph a set of vertices with total weight at most p so that the maximum weight of any remaining component is at most ℓ. Using this algorithm as a black box, we can then solve Weighted Vertex Integrity by iterating over all possible values of ℓ, between 1 and the target vertex integrity. If vertex weights are polynomially bounded, this requires a polynomial number of iterations, giving the algorithm of [21]. However, if weights are given in binary, the target vertex integrity could be exponential in n, so in general, it does not appear possible to guess the weight of the heaviest component in an optimal solution.

Our contribution to the algorithm of [21] is to observe that for graphs of modular width mw the weight of the heaviest component may take at most $2^{\mathcal{O}(\mathrm{mw})}n$ distinct possible values. Hence, for this parameter, guessing the weight of the heaviest component in an optimal solution can be done in FPT time. We can therefore plug in this result to the algorithm of [21] to obtain an algorithm for Weighted Vertex Integrity with binary weights running in time $2^{\mathcal{O}(\mathrm{mw})}n^{\mathcal{O}(1)}$.

Our observation is based on the following lemma.

▶ **Lemma 11** (⋆). *Let $G = (V, E)$ be an instance of* Weighted Vertex Integrity. *There exists an optimal solution using a separator S such that for all connected components D of $G - S$ and modules M of G we have one of the following: (i) $M \cap D = \emptyset$, (ii) $M \subseteq D$, or (iii) $D \subseteq M$.*

We also recall the algorithmic result of [21].

▶ **Theorem 12** ([21]). *There exists an algorithm that takes as input a vertex-weighted graph $G = (V, E)$ and an integer ℓ and computes the minimum integer p such that there exists a separator S of G of weight at most p such that each component of $G - S$ has weight at most ℓ. The algorithm runs in time $2^{\mathcal{O}(\mathrm{mw})}n^{\mathcal{O}(1)}$, where n is the size of the input.*

Putting Lemma 11 and Theorem 12 together we obtain the main result of this section.

▶ **Theorem 13** (⋆). *There exists an algorithm that solves* Weighted Vertex Integrity *in time $2^{\mathcal{O}(\mathrm{mw})}n^{\mathcal{O}(1)}$, where* mw *is the modular width of the input graph, n is the size of the input, and weights are allowed to be written in binary.*

7 Vertex Cover Number

In this section, we design single-exponential algorithms for Vertex Integrity parameterized by vertex cover number. We suppose that a minimum vertex cover C of size vc is given since it can be computed in time $\mathcal{O}(1.2738^{\text{vc}} + \text{vc}n)$ [5]. We start by presenting an algorithm for Unweighted Vertex Integrity, before moving on to the weighted version of the problem.

▶ **Theorem 14** ($\star$). Unweighted Vertex Integrity *can be solved in time* $5^{\text{vc}} n^{\mathcal{O}(1)}$.

We now move on to the weighted case of the problem. It is clear that Weighted Vertex Integrity is FPT parameterized by vertex cover, due to Theorem 13 and the relation between modular-width and vertex cover. However, this gives a double-exponential dependence on vc, as $\text{mw} \leq 2^{\text{vc}} + \text{vc}$ and there are some graphs for which this is essentially tight. We would like to obtain an algorithm that is as efficient as that of Theorem 14. The algorithm of Theorem 14, however, cannot be applied to the weighted case because the case of the branching where we place a vertex of the independent set in the separator is not guaranteed to make much progress (the vertex could have very small weight compared to our budget).

Before we proceed, it is worth thinking a bit about how this can be avoided. One way to obtain a faster FPT algorithm would be, rather than guessing only the intersection of the optimal separator S with the vertex cover C, to also guess how the vertices of $C \setminus S$ are partitioned into connected components in the optimal solution. This would immediately imply the decision for all vertices of the independent set: vertices with neighbors in two components must clearly belong to S, while the others cannot belong to S if S is irredundant. This algorithm would give a complexity of $\text{vc}^{\mathcal{O}(\text{vc})} n^{\mathcal{O}(1)}$, however, because the number of partitions of C is slightly super-exponential.

Let us sketch the high level idea of how we handle this. Our first step is, similarly to Theorem 13, to calculate the weight $w_{\max}$ of the most expensive connected component of the optimal solution. For this, there are at most $2^{\text{vc}} + n$ possibilities, because this component is either a single vertex, or it has a non-empty intersection with C. However, if we fix its intersection with C, then this fixes its intersection with the independent set: the component must contain (by irredundancy) exactly those vertices of the independent set all of whose neighbors in C are contained in the component. Having fixed a value of $w_{\max}$ we simply seek the best separator so that all components have weight at most $w_{\max}$. The reason we perform this guessing step is that this version of this problem is easier to decompose: if we have a disconnected graph, we simply calculate the best separator in each part and take the sum (this is not as clear for the initial version of Vertex Integrity).

Suppose then that we have fixed $w_{\max}$, how do we find the best partition of C into connected components? We apply a win/win argument: if the optimal partition has a connected component that contains many (say, more than vc/10) vertices of C, we simply guess the intersection of the component with C and complete it with vertices from the independent set, as previously, while placing vertices with neighbors inside and outside the component in the separator. If the weight of the component is at most $w_{\max}$, we recurse in the remaining instance, which has vertex cover at most 9vc/10. The complexity of this procedure works out as $T(\text{vc}) \leq 2^{\text{vc}} \cdot T(9\text{vc}/10) = 2^{\mathcal{O}(\text{vc})}$.

What if the optimal partition of C only has components with few vertices of C? In that case we observe that we do not need to compute the full partition (which would take time vc^{vc}), but it suffices to guess a good bipartition of C into two sets, of roughly the same size (say, both sets have size at least 2vc/5), such that the two sets are a coarsening of the optimal partition. In other words, we compute two subsets of C, of roughly equal size, such that the intersection of each connected component with C is contained in one of the two sets. This is always possible in this case, because no connected component has a very large intersection with C. Now, all vertices of I which have neighbors on both sides of the bipartition of C must be placed in the separator. But once we do this, we have disconnected the instance into two independent instances, each of vertex cover at most $3\text{vc}/5$. The complexity of this procedure again works out as $T(\text{vc}) \leq 2^{\text{vc}} \cdot 2 \cdot T(3\text{vc}/5) = 2^{\mathcal{O}(\text{vc})}$.

Let us now proceed to the technical details. To solve Weighted Vertex Integrity, we first define the annotated and optimization version of the problem.

Annotated Weighted Vertex Integrity with Vertex Cover

Instance: A vertex-weighted graph $G = (V, E, w)$, a vertex cover C of G, an integer $w_{\max}$.

Goal: Find a minimum weight irredundant wvi-set $S \subseteq V \setminus C$ such that $w(D) \leq w_{\max}$ for all $D \in \texttt{cc}(G - S)$. If there is no such S, report NO.

Then we give an algorithm that solves Annotated Weighted Vertex Integrity with Vertex Cover.

▶ **Theorem 15** ($\star$). Annotated Weighted Vertex Integrity with Vertex Cover *can be solved in time* $2^{\mathcal{O}(|C|)}n^{\mathcal{O}(1)}$.

▶ **Theorem 16** ($\star$). Weighted Vertex Integrity *can be solved in time* $2^{\mathcal{O}(\text{vc})}n^{\mathcal{O}(1)}$.

8 Conclusion

We have presented a number of new results on the parameterized complexity of computing vertex integrity. The main question that remains open is whether the slightly super-exponential $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ algorithm, where k is the vertex integrity itself, can be improved to single-exponential. Although we have given such an algorithm for the more restricted parameter vertex cover, we conjecture that for vertex integrity the answer is negative. Complementing this question, it would be interesting to consider approximation algorithms for vertex integrity, whether trying to obtain FPT approximations in cases where the problem is W[1]-hard, or trying to obtain almost-optimal solutions via algorithms that run with a better parameter dependence. Again, a constant-factor or even $(1+\varepsilon)$-approximation running in time $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ would be the ideal goal. Do such algorithms exist or can they be ruled out under standard complexity assumptions?

References

1 C. A. Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs – A comparative survey. In *Proceedings of the first Carbondale combinatorics conference (Carbondale, Ill., 1986)*, volume 1, pages 13–22, 1987.

2 Rémy Belmonte, Tesshu Hanaka, Ioannis Katsikarelis, Michael Lampis, Hirotaka Ono, and Yota Otachi. Parameterized complexity of safe set. *J. Graph Algorithms Appl.*, 24(3):215–245, 2020. doi:10.7155/JGAA.00528.

3 Matthias Bentert, Klaus Heeger, and Tomohiro Koana. Fully polynomial-time algorithms parameterized by vertex integrity using fast matrix multiplication. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.16.

4 Sriram Bhyravarapu, Lawqueen Kanesh, A. Mohanapriya, Nidhi Purohit, N. Sadagopan, and Saket Saurabh. On the parameterized complexity of minus domination. In *SOFSEM 2024: Theory and Practice of Computer Science – 49th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2024*, volume 14519 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2024. doi:10.1007/978-3-031-52113-3_7.

5 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010. doi:10.1016/J.TCS.2010.06.026.

6 Lane H. Clark, Roger C. Entringer, and Michael R. Fellows. Computational complexity of integrity. *J. Combin. Math. Combin. Comput.*, 2:179–191, 1987.

7 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.

8 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. doi:10.1007/978-3-662-53622-3.

9 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/S00453-016-0127-X.

10 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: programs with few global variables and constraints. *Artif. Intell.*, 300:103561, 2021. doi:10.1016/J.ARTINT.2021.103561.

11 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013. doi:10.1016/J.EJC.2012.04.008.

12 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009. doi:10.1007/S00224-009-9167-9.

13 Michael R. Fellows and Sam Stueckle. The immersion order, forbidden subgraphs and the complexity of network integrity. *J. Combin. Math. Combin. Comput.*, 6:23–32, 1989.

14 Shinya Fujita and Michitaka Furuya. Safe number and integrity of graphs. *Discret. Appl. Math.*, 247:398–406, 2018. doi:10.1016/J.DAM.2018.03.074.

15 Ajinkya Gaikwad and Soumen Maity. Offensive alliances in graphs. *Theor. Comput. Sci.*, 989:114401, 2024. doi:10.1016/J.TCS.2024.114401.

16 Ajinkya Gaikwad and Soumen Maity. On structural parameterizations of the harmless set problem. *Algorithmica*, 86(5):1475–1511, 2024. doi:10.1007/S00453-023-01199-9.

17 Jakub Gajarský, Michael Lampis, Kazuhisa Makino, Valia Mitsou, and Sebastian Ordyniak. Parameterized algorithms for parity games. In *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015*, volume 9235 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2015. doi:10.1007/978-3-662-48054-0_28.

18 Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *Parameterized and Exact Computation – 8th International Symposium, IPEC 2013*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

19 Robert Ganian, Fabian Klute, and Sebastian Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. doi:10.1007/S00453-020-00758-8.

20 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/J.TCS.2022.03.021.

21 Tatsuya Gima, Tesshu Hanaka, Yasuaki Kobayashi, Ryota Murai, Hirotaka Ono, and Yota Otachi. Structural parameterizations of vertex integrity. In *WALCOM: Algorithms and Computation – 18th International Conference and Workshops on Algorithms and Computation, WALCOM 2024*, volume 14549 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2024. doi:10.1007/978-981-97-0566-5_29.

22 Tatsuya Gima and Yota Otachi. Extended MSO model checking via small vertex integrity. *Algorithmica*, 86(1):147–170, 2024. doi:10.1007/S00453-023-01161-9.

23 Klaus Jansen, Stefan Kratsch, Dániel Marx, and Ildikó Schlotter. Bin packing with fixed number of bins revisited. *J. Comput. Syst. Sci.*, 79(1):39–49, 2013. doi:10.1016/J.JCSS.2012.04.004.

24 Richard M. Karp. Reducibility among combinatorial problems. In *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.

25 Daniel J. Kleitman and Douglas B. West. Spanning trees with many leaves. *SIAM J. Discret. Math.*, 4(1):99–106, 1991. doi:10.1137/0404010.

26 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

27 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In *32nd International Symposium on Algorithms and Computation, ISAAC 2021*, volume 212 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ISAAC.2021.34.

28 Michael Lampis and Manolis Vasilakis. Structural parameterizations for two bounded degree problems revisited. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 77:1–77:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.77.

29 Martijn van Ee. Some notes on bounded starwidth graphs. *Inf. Process. Lett.*, 125:9–14, 2017. doi:10.1016/J.IPL.2017.04.011.

The Complexity of (P_3, H)-Arrowing and Beyond

Zohair Raza Hassan
Rochester Institute of Technology, Rochester, NY, USA

Abstract

Often regarded as the study of how order emerges from randomness, Ramsey theory has played an important role in mathematics and computer science, giving rise to applications in numerous domains such as logic, parallel processing, and number theory. The core of graph Ramsey theory is arrowing: For fixed graphs F and H, the (F, H)-Arrowing problem asks whether a given graph, G, has a red/blue coloring of the edges of G such that there are no red copies of F and no blue copies of H. For some cases, the problem has been shown to be coNP-complete, or solvable in polynomial time. However, a more systematic approach is needed to categorize the complexity of all cases.

We focus on (P_3, H)-Arrowing as $F = P_3$ is the simplest meaningful case for which the complexity question remains open, and the hardness for this case likely extends to general (F, H)-Arrowing for nontrivial F. In this pursuit, we also gain insight into the complexity of a class of matching removal problems, since (P_3, H)-Arrowing is equivalent to H-free Matching Removal. We show that (P_3, H)-Arrowing is coNP-complete for all 2-connected H except when $H = K_3$, in which case the problem is in P. We introduce a new graph invariant to help us carefully combine graphs when constructing the gadgets for our reductions. Moreover, we show how (P_3, H)-Arrowing hardness results can be extended to other (F, H)-Arrowing problems. This allows for more intuitive and palatable hardness proofs instead of ad-hoc constructions of SAT gadgets, bringing us closer to categorizing the complexity of all (F, H)-Arrowing problems.

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

Keywords and phrases Graph arrowing, Ramsey theory, Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.59

Acknowledgements I want to thank my advisors Edith Hemaspaandra and Stanisław Radziszowski.

1 Introduction and related work

At what point, if ever, does a system get large enough so that certain patterns become unavoidable? This question lies at the heart of Ramsey theory, which, since its inception in the 1930s, aims to find these thresholds for various combinatorial objects. Ramsey theory has played an important role in mathematics and computer science, finding applications in fields such as cryptography, algorithms, game theory, and more [11]. A key operator within Ramsey theory is the arrowing operator, which is defined for graphs like so: given graphs F, G, and H, we say that $G \to (F, H)$ (read, G *arrows* F, H) if every red/blue coloring of G's edges contains a red F or a blue H. In this work, we analyze the complexity of computing this operator when F and H are fixed graphs. The general problem is defined as follows.

▶ **Problem 1** ((F, H)-Arrowing). *For fixed F and H, given a graph G, does $G \to (F, H)$?*

The problem is clearly in coNP; a red/blue coloring of G's edges with no red F and no blue H forms a certificate that can be verified in polynomial time since F and H are fixed graphs. Such a coloring is referred to as an (F, H)-good coloring. The computational complexity of (F, H)-Arrowing has been categorized for several – but not all – pairs (F, H). For instance, (P_2, H)-Arrowing is in P for all H, where P_2 is the path graph on 2 vertices. This is because any coloring of an input graph G that does not contain a red P_2 must be entirely blue, and thereby (P_2, H)-Arrowing is equivalent to determining whether G is

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 59; pp. 59:1–59:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

H-free. Burr showed that (F, H)-Arrowing is coNP-complete when F and H are 3-connected graphs – these are graphs which remain connected after the removal of any two vertices, e.g., (K_5, K_6)-Arrowing [3]. More results of this type are discussed in Section 2.

In this work, we explore the simplest nontrival case for F, $F = P_3$, and provide a complete classification of the complexity when H is a 2-connected graph – a graph that remains connected after the removal of any one vertex. In particular, we prove:

▶ **Theorem 1.** *(P_3, H)-Arrowing is coNP-complete for all 2-connected H except when $H = K_3$, in which case the problem is in P.*

We do this by reducing an NP-complete SAT variant to (P_3, H)-Arrowing's complement, (P_3, H)-Nonarrowing (for fixed H, does there exist a (P_3, H)-good coloring of a given graph?), and showing how to construct gadgets for any 2-connected H. It is important to note that combining different copies of H can be troublesome; it is possible to combine graphs in a way so that we end up with more copies of it than before, e.g., combining two P_n's by their endpoints makes several new P_n's across the vertices of both paths. Results such as Burr's which assume 3-connectivity avoid such problems, in that we can combine several copies of 3-connected graphs without worrying about forming new ones. If G is 3-connected with vertices $u, v \in V(G)$ and we construct a graph F by taking two copies of G and identifying u across both copies, then identifying v across both copies, no new copies of G are constructed in this process; if a new G is created then it must be disconnected by the removal of the two identified vertices, contradicting F's 3-connectivity. This makes it easier to construct gadgets for reductions. To work with 2-connected graphs and show how to combine them carefully, we present a new measure of intra-graph connectivity called **edge pair linkage**, and use it to prove sufficient conditions under which two copies of a 2-connected graph G can be combined without forming new copies of G.

By targeting the (P_3, H) case we gain new insight and tools for the hardness of (F, H)-Arrowing in the general case since $F = P_3$ is the simplest case for F. We conjecture that if (P_3, H)-Arrowing is hard, then (F, H)-Arrowing is also hard for all nontrivial F, but this does not at all follow immediately. Towards the goal of categorizing the complexity of all (F, H)-Arrowing problems, we show how to extend the hardness results of (P_3, H)-Arrowing to other (F, H)-Arrowing problems in Section 5. These extensions are more intuitive and the resulting reductions are more palatable compared to constructing SAT gadgets. We believe that techniques similar to the ones shown in this paper can be used to eventually categorize the complexity of (F, H)-Arrowing for all (F, H) pairs.

The rest of the paper is organized as follows. Related work is discussed in Section 2. We present preliminaries in Section 3, wherein we also define and analyze edge pair linkage. Our complexity results for (P_3, H)-Arrowing are proven in Section 4. We show how our hardness results extend to other arrowing problems in Section 5, and we conclude in Section 6. All proofs omitted in the main text are provided in the appendix.

2 Related work

Complexity of (F, H)-Arrowing. Burr showed that (F, H)-Arrowing is in P when F and H are both star graphs, or when F is a matching [3]. Hassan et al. showed that (P_3, P_3)- and (P_3, P_4)-Arrowing are also in P [6]. For hardness results, Burr showed that (F, H)-Arrowing is coNP-complete when F and H are members of Γ_3, the family of all 3-connected graphs and K_3. The generalized (F, H)-Arrowing problem, where F and H are also part of the input, was shown to be Π_2^p-complete by Schaefer, who focused on constructions where F is a tree and H is a complete graph [13].[1] Hassan et al. recently showed that (P_k, P_ℓ)-Arrowing is coNP-complete for all k and ℓ aside from the exceptions listed above [6]. We note that (P_4, P_4)-Arrowing was shown to be coNP-complete by Rutenburg much earlier [12].

Matching removal. A matching is a collection of disjoint edges in a graph. Interestingly, there is an overlap between matching removal problems, defined below, and (P_3, H)-Arrowing.

▶ **Problem 2** (Π-Matching Removal [9]). *Let Π be a fixed graph property. For a given graph G, does there exist a matching M such that $G' = (V(G), E(G) - M)$ has property Π?*

Let Π be the property that G is H-free for some fixed graph H. Then, this problem is equivalent to (P_3, H)-Nonarrowing; a lack of red P_3's implies that only disjoint edges can be red, as in a matching, and the remaining (blue) subgraph must be H-free. Lima et al. showed that the problem is NP-complete when Π is the property that G is acyclic [7], or that G contains no odd cycles [8, 9].

Ramsey and Folkman numbers. The major research avenue involving arrowing is that of finding Ramsey and Folkman numbers. Ramsey numbers are concerned with the smallest complete graphs with arrowing properties, whereas Folkman numbers allow for any graph with some extra structural constraints. We refer the reader to surveys by Radziszowski [10] and Bikov [2] for more information on Ramsey and Folkman numbers, respectively.

3 Preliminaries

3.1 Notation and terminology

All graphs discussed in this work are simple and undirected. $V(G)$ and $E(G)$ denote the vertex and edge set of a graph G, respectively. We denote an edge in $E(G)$ between $u, v \in V(G)$ as (u, v). For two disjoint subsets $A, B \subsetneq V(G)$, $E_G(A, B)$ refers to the edges with one vertex in A and one vertex in B. For a subset $A \subseteq V(G)$, $G[A]$ denotes the induced subgraph on A. The neighborhood of a vertex $v \in V(G)$ is denoted as $N_G(v) = \{u \mid (u, v) \in E(G)\}$ and its degree as $d_G(v) := |N_G(v)|$. A connected graph is called k-connected if it has more than k vertices and remains connected whenever fewer than k vertices are removed.

Vertex identification is the process of replacing two vertices u and v with a new vertex w such that w is adjacent to all remaining neighbors $N(u) \cup N(v)$. For edges (u, v) and (p, q), edge identification is the process of identifying u with p, and v with q.

The path, cycle, and complete graphs on n vertices are denoted as P_n, C_n, and K_n, respectively. The complete graph on n vertices missing an edge is denoted as J_n. $K_{1,n}$ is the star graph on $n+1$ vertices. For $n \geq 3$, we define TK_n (tailed K_n) as the graph obtained by identifying a vertex of a K_2 and any vertex in K_n. The vertex of degree one in a TK_n is called the tail vertex of TK_n.

We introduce a new notion, defined below, to measure how "connected" a pair of edges in a graph is, which will be useful when identifying edges between multiple copies of the same graph. Examples have been shown in Figure 1.

▶ **Definition 2.** *For a pair of edges $e, f \in E(G)$, we define its* ***edge pair linkage****, $\mathrm{epl}_G(e, f)$, as the number of edges adjacent to both e and f. It is infinity if e and f share at least one vertex. Note that $\mathrm{epl}_G(e, f) \leq 4$ when e and f share no vertices. For a graph G, we define $\mathrm{mepl}(G) := \min_{e,f \in E(G)} \mathrm{epl}(e, f)$ as the minimum edge pair linkage across all edge pairs.*

[1] $\Pi_2^p = \mathrm{coNP}^{\mathrm{NP}}$, the class of all problems whose complements are solvable by a nondeterministic polynomial-time Turing machine having access to an NP oracle.

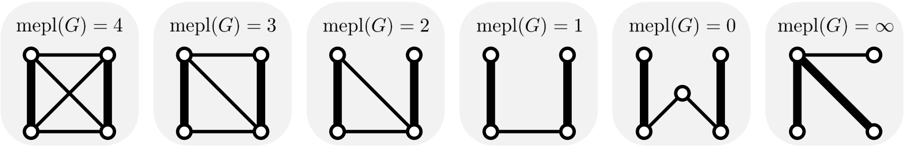


Figure 1 Graphs with different $\mathrm{mepl}(G)$ values. Bold edges have $\mathrm{epl}_G = \mathrm{mepl}(G)$.

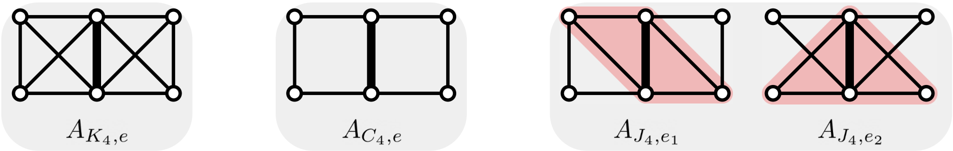


Figure 2 Proof for Lemma 3 when $|V(H)| = 4$. It is easy to see that $A_{H,e}$ for $H \in \{C_4, K_4\}$ has exactly two copies of H for arbitrary e. Moreover, constructing $A_{J_4,e}$ introduces a new J_4 (highlighted in red) for both nonisomorphic choices of $e \in E(J_4)$. Identified edges are bolded.

It is easy to see that the only graphs with $\mathrm{mepl}(G) = \infty$ are the star graphs, $K_{1,n}$, and K_3 since these are the only graphs that do not have disjoint edges. When the context is clear, the subscript G for $\mathrm{epl}_G(\cdot)$, $d_G(\cdot)$, etc. will be omitted.

An (F, H)-good coloring of a graph G is a red/blue coloring of $E(G)$ where the red subgraph is F-free, and the blue subgraph is H-free. We say that G is (F, H)-good if it has at least one (F, H)-good coloring. When the context is clear, we will omit (F, H) and refer to the coloring as a good coloring.

3.2 Combining graphs

Suppose H is a 3-connected graph. Consider the graph $A_{H,e}$, obtained by taking two disjoint copies of H and identifying some arbitrary $e \in E(H)$ from each copy. Observe that no new copy of H – referred to as a rogue copy of H in $A_{H,e}$ – is constructed during this process; if a new H is created then it must be disconnected by the removal of the two identified vertices, contradicting H's 3-connectivity. This is especially useful when proving the existence of good colorings; to show that a coloring of $A_{H,e}$ has no blue H, we know that only two copies of H need to be looked at, without worrying about any other rogue H. Unfortunately, this property does not hold for all 2-connected graphs. Instead, we use minimum edge pair linkage to explore sufficient conditions that allow us to combine multiple copies of graphs without concerning ourselves with any potential rogue copies.

▶ **Lemma 3.** *Suppose H is a 2-connected graph such that $|V(H)| \geq 4$ and $\mathrm{mepl}(H) \geq 2$. Given H and $e \in E(H)$, let $A_{H,e}$ be the graph obtained by taking two disjoint copies of H and identifying e from each copy. For all such H, except J_4, there exists $e \in E(H)$ such that $A_{H,e}$ has exactly two copies of H, i.e., no new copy of H is formed after identifying e.*

Proof. For $|V(H)| = 4$, the statement is easily observed as the only cases to consider are C_4, J_4, and K_4. See Figure 2. Suppose $|V(H)| \geq 5$. We will first construct $A_{H,e}$ using an arbitrary $e = (u, v) \in E(H)$ and assume that a new copy of H is constructed after identifying e. Let X and Y denote the subgraphs of $A_{H,e}$ corresponding to the two copies of H in $A_{H,e}$ that identify (u, v). It follows that, $V(X) \cap V(Y) = \{u, v\}$ and $V(X) \cup V(Y) = V(A_{H,e})$. Similarly, $E(X) \cap E(Y) = \{e\}$ and $E(X) \cup E(Y) = E(A_{H,e})$. Suppose Z is a subgraph corresponding to another copy of H in $A_{H,e}$, i.e. $V(Z) \neq V(X)$

and $V(Z) \neq V(Y)$. Let $V_{Z_X} = (V(X) - V(Y)) \cap V(Z)$ be the vertices of Z only in X, and $E_{Z_X} = \{(p,q) \in E(Z) \mid p,q \in V_{Z_X}\}$. V_{Z_Y} and E_{Z_Y} are defined similarly. In the following claim, we observe the properties of Z and the original graph H with $\text{mepl}(H) \geq 2$.

▷ Claim 4. If Z exists in $A_{H,e}$, the following must be true: *(1)* Both V_{Z_X} and V_{Z_Y} are nonempty, *(2)* $u \in V(Z)$ and $v \in V(Z)$, *(3)* at least one of E_{Z_X} and E_{Z_Y} is empty, and *(4)* there exists $w \in V(H)$ with $d_H(w) = 2$.

The proof of Claim 4 is given in Appendix A. Now, let $e = (u,v) \in E(H)$ such that $d_H(v) = 2$. Consider the graph $A_{H,e}$. Note that since $d_H(v) = 2$, we have $d_{A_{H,e}}(v) = d_H(v) + d_H(v) - 1 = 3$. Let w_X and w_Y be the neighbors of v in $V(X) - V(Y)$ and $V(Y) - V(X)$, respectively. We know that $V(Z)$ includes u and v from Claim 4(2). We now show that w_X and w_Y must also belong to $V(Z)$: if neither belong to $V(Z)$, then $d_Z(v) = 1$, contradicting H's 2-connectivity (removing u disconnects H). Suppose w.l.o.g., that $w_X \notin V(Z)$. Since V_{Z_X} is nonempty and H is connected, there is at least one vertex in V_{Z_X} connected to u. However, removing u would disconnect Z, again contradicting H's 2-connectivity. Thus, $\{u, v, w_X, w_Y\} \subseteq V(Z)$. Using a similar argument, we can also show that both (v, w_X) and (v, w_Y) must belong to $E(Z)$.

Let p be a vertex in $V(Z) - \{u, v, w_X, w_Y\}$. We know p exists since $|V(H)| \geq 5$. W.l.o.g., we assume that $p \in V_{Z_X}$. Note that p cannot be adjacent to v since $N_{A_{H,e}}(v) = \{u, w_X, w_Y\}$. We now consider the neighborhood of p in Z. If p has a neighbor $q \in V_{Z_X} - \{w_X\}$, then $\text{epl}_Z((p,q),(v,w_Y)) = 0$, contradicting our assumption that $\text{mepl}(H) \geq 2$. Since $d_Z(p) = d_H(p) \geq 2$ and the only options remaining for p's neighborhood are u and w_X, we must have that p is connected to both u and w_X. In this case, we have that $\text{epl}_Z((p,w_X),(v,w_Y)) \leq 1$, which is still a contradiction. ◀

4 The complexity of $(\mathrm{P}_3, \mathrm{H})$-Arrowing

In this section, we discuss our complexity results stated in Theorem 1. We first show that (P_3, K_3)-Arrowing is in P (Theorem 5). The rest of the section is spent setting up our hardness proofs for all 2-connected $H \neq K_3$, which we prove formally in Theorems 13 and 14.

▶ **Theorem 5.** *(P_3, K_3)-Arrowing is in P.*

Proof. Let G be the input graph. Let $\gamma : E(G) \to \mathbb{Z}_{\geq 0}$ be a function that maps each edge to the number of triangles it belongs to in G. Note that γ can be computed in $O(|E(G)|^3)$ time via brute-force. Let t be the number of triangles in G. Clearly, any (P_3, K_3)-good coloring of G corresponds to a matching of total weight $\geq t$; otherwise, there would be some K_3 in G in the blue subgraph of G. Now, suppose there exists a matching M with weight at least t that does not correspond to a (P_3, K_3)-good coloring of G. Then, there must exist a copy of K_3 in G with at least two of its edges in M. However, since K_3's maximal matching contains only one edge, M must contain a P_3, which is a contradiction. Thus, we can solve (P_3, K_3)-Arrowing by finding the maximum weight matching of (G, γ), which can be done in polynomial time [5], and checking if said matching has weight equal t. ◀

To show that (P_3, H)-Arrowing is coNP-complete we show that its complement, (P_3, H)-Nonarrowing, is NP-hard. We reduce from the following NP-complete SAT variant:

▶ **Problem 3** ($(2,2)$-3SAT [1]). *Let ϕ be a 3CNF formula where each clause has exactly three distinct variables, and each variable appears exactly four times: twice unnegated and twice negated. Does there exist a satisfying assignment for ϕ?*

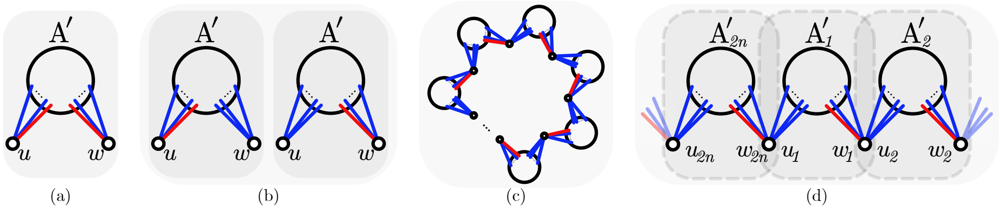


Figure 3 **(a)** The graph A' when u and w are adjacent to a red edge. **(b)** The graph A' when either u or w is adjacent to a red edge. **(c)** The graph B. **(d)** A zoomed in look at B.

Important definitions and the idea behind our reduction are provided in Section 4.1, while the formal proofs are presented in Section 4.2.

4.1 Defining special graphs and gadgets for proving hardness

We begin by defining a useful term for specific vertices in a coloring, after which we describe and prove the existence of some special graphs. We then define the two gadgets necessary for our reduction and describe how they provide a reduction from $(2, 2)$-3SAT.

▶ **Definition 6.** *For a graph G and a coloring c, a vertex $v \in V(G)$ is called a **free vertex** if it is not adjacent to any red edge in c. Otherwise, it is called a **nonfree vertex.***

Note that in any (P_3, H)-good coloring, a vertex can be adjacent to at most one red edge, otherwise, a red P_3 is formed. Intuition suggests that, by exploiting this restrictive property, we could freely force "any" desired blue subgraph if we can "append red edges" to it. This brings us to our next definition, borrowed from Schaefer's work on (F, H)-Arrowing [13]:

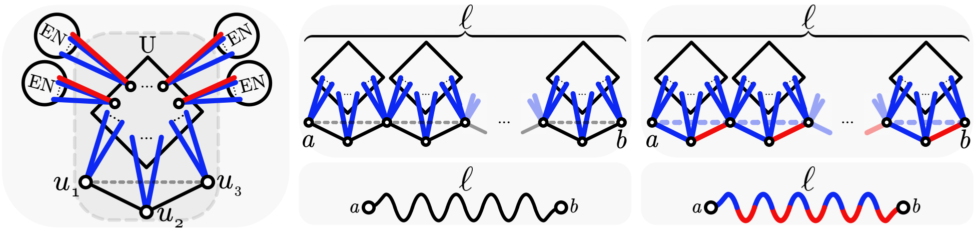
▶ **Definition 7** ([13]). *A graph G is called an **(F, H)-enforcer** with **signal vertex v** if it is (F, H)-good and the graph obtained from G by attaching a new edge to v has the property that this edge is colored blue in all (F, H)-good colorings.*

Throughout our text, when the context is clear, we will use the shorthand **append an enforcer to $u \in V(G)$** to mean we will add an (F, H)-enforcer to G and identify its signal vertex with u. We prove the existence of (F, H)-enforcers when $F = P_3$ below. This proof provides a good example of the role 2-connectivity plays while constructing our gadgets, showcasing how we combine graphs while avoiding constructing new copies of H. The arguments made are used frequently in our other proofs as well.

▶ **Lemma 8.** *(P_3, H)-enforcers exist for all 2-connected H.*

Proof. We extend an idea presented by Burr [3]. Let A be a "minimally bad" graph such that $A \to (P_3, H)$, but removing any edge e from A gives a (P_3, H)-good graph. Let $e = (u, w)$ and $A' = A - e$. This graph is illustrated in Figure 3. Observe that in any (P_3, H)-good coloring of A', at least one edge adjacent to u or w must be red; otherwise, such a coloring and a red (u, w) gives a good coloring for A, contradicting the fact that $A \to (P_3, H)$. If both u and w are adjacent to red edges in all good colorings, then A' is a (P_3, H)-enforcer, and either u or w can be the signal vertex .

If there exists a coloring where only one of $\{u, w\}$ is adjacent to a red edge, then we can construct an enforcer, B, as follows. Let $n = |V(H)|$. Make $2n$ copies of A', where u_i and w_i refer to the vertex u and w in the i^{th} copy of A', called A'_i. Now, identify each w_i with u_{i+1} for $i \in \{1, 2, \ldots, 2n - 1\}$, and identify w_{2n} with u_1 (see Figure 3). Although w_i and



(a) (b) (c)

Figure 4 **(a)** The (P_3, H)-signal extender described in Lemma 10, where enforcers are labeled EN and the copy of H is labeled U. Edges whose colors are fixed in all (P_3, H)-good colorings have been pre-colored. The edge (u_1, u_3) is dashed to signify it may or may not exist in the construction. **(b)** At the top, we show how extenders can be connected sequentially to form arbitrarily large extenders. The enforcers have been removed from the illustration for clarity. The in- and out-vertices are marked a and b, respectively. At the bottom, we show how signal extenders will be depicted in our figures, where ℓ is the number of concatenated constructed extenders. **(c)** At the top, we show the coloring of the signal extender when vertex a is a free vertex. At the bottom, we show the corresponding coloring of our representation of signal extenders.

u_{i+1} are now the same vertex in B, we will use their original names to make the proof easier to follow. It is easy to see that when w_1 is adjacent to a red edge in A'_1, then u_2 cannot be adjacent to any red edge in A'_2, causing w_2 to be adjacent to a red edge in A'_2, and so on. A similar argument holds when considering the case where u_1 is adjacent to a red edge in A'_1. Since every u_i and w_i is adjacent to a red edge, any of them can be our desired signal vertex.

Note that B must be (P_3, H)-good because each A'_i is (P_3, H)-good, and no new H is made during the construction of the graph; since H is 2-connected, H cannot be formed between two copies of A'_i's, otherwise there is a single vertex that can be removed to disconnect such an H, contradicting 2-connectivity. Thus, any new copy of H must go through all A_i's, which is not possible since such an H would have $\geq 2|V(H)|$ vertices. ◀

Using enforcers, we construct another graph that plays an important role in our reductions.

▶ **Definition 9.** *A graph G is called a* ***(P_3, H)-signal extender*** *with* ***in-vertex a*** *and* ***out-vertex b*** *if it is (P_3, H)-good and, in all (P_3, H)-good colorings, b is nonfree if a is free.*

▶ **Lemma 10.** *(P_3, H)-signal extenders exist for all 2-connected H.*

Proof. Let $n = |V(H)|$. Construct a graph G like so. Take a copy of H, and let $u_1, u_2, \ldots, u_n \in U$ be the vertices of H in G, such that (u_1, u_2) and (u_2, u_3) are edges of H. For each u_i for $i \in \{4, 5, \ldots, n\}$, append an enforcer to u_i. Observe that no "new" H is constructed during this process since H is 2-connected. Since each vertex except u_1, u_2, and u_3 is connected to an enforcer, each edge in $G[U]$, except (u_1, u_2), (u_2, u_3), and (u_1, u_3) must be blue. However, not all of them can be blue, otherwise $G[U]$ is a blue H. Therefore, in any good coloring, if u_1 is a free vertex, (u_2, u_3) must be red, making u_3 a nonfree vertex. Thus, u_1 is our in-vertex, and u_3 is our out-vertex. We illustrate G in Figure 4(a). ◀

Observe that multiple copies of these extenders can be used to form larger ones (see Figure 4). With the enforcer and extender graphs defined, we are ready to construct the gadgets for our reductions. Below, we define variable and clause gadgets and describe how they are used in our proofs. Recall that we are reducing from $(2, 2)$-3SAT. We will explain how our graphs encode clauses and variables after the definitions.

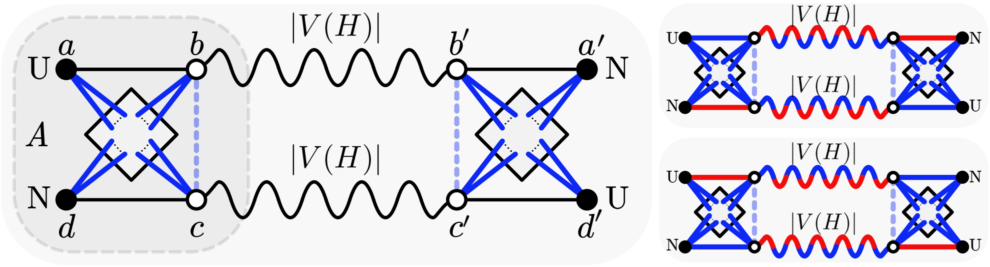


Figure 5 On the left, we show the variable gadget constructed using two copies of A as described in Theorem 13's proof and signal extenders. The vertices in the square are the vertices of A which had enforcers appended to them. The edge (b, c) is dashed to signify that it may or may not exist. Edges have been precolored wherever possible. Note that if (b, c) exists, it must be blue: if (b, c) is red, the attached signal extenders will force (a', b'), (b', c'), and (c', d') to be blue, forming a blue H. By symmetry, (b', c') must also be blue. Now, observe that at least one edge in $\{(a, b), (c, d)\}$ must be red, otherwise we form a blue H in A. Suppose (a, b) is red: the signal extender forces (a', b') to be blue. To avoid a blue H, (c', d') must be red, which forces (c, d) to be blue. In this case, the vertices marked **U** are nonfree vertices, and the vertices marked **N** are free. A similar pattern can be observed when we color (c, d) red instead, giving us colorings where vertices marked **U** are free and vertices marked **N** are nonfree.

▶ **Definition 11.** *For a fixed H, a **clause gadget** is a (P_3, H)-good graph CG containing vertices i_1, i_2, and i_3 – referred to as **input vertices**, such that if vertices outside the gadget, o_1, o_2 and o_3, are connected to i_1, i_2, and i_3, respectively, then each of the eight possible combinations of (o_j, i_j)'s colors should allow a (P_3, H)-good coloring for CG, except the coloring where all (o_j, i_j)'s are red, which should not allow a good coloring of CG.*

▶ **Definition 12.** *For a fixed H, a **variable gadget** is a (P_3, H)-good graph VG containing four **output vertices**: two **unnegated output vertices**, u_1 and u_2, and two **negated output vertices**, n_1 and n_2, such that:*

1. *In each (P_3, H)-good coloring of VG:*
   a. *If u_1 or u_2 is a free vertex, then n_1 and n_2 must be nonfree vertices.*
   b. *If n_1 or n_2 is a free vertex, then u_1 and u_2 must be nonfree vertices.*
2. *There exists at least one (P_3, H)-good coloring of VG where u_1 and u_2 are free vertices.*
3. *There exists at least one (P_3, H)-good coloring of VG where n_1 and n_2 are free vertices.*

Note how clause gadgets and their input vertices correspond to OR gates and their inputs; the external edges, denoted as (o_j, i_j)'s in the definition, behave like true or false signals: blue is true, and red is false. Similarly, output vertices of variable gadgets behave like sources for these signals. The reduction is now straightforward: for a formula ϕ, construct G_ϕ like so. For each variable and clause, add a corresponding gadget. Then, identify the output and input vertices according to how they appear in each clause. It is easy to see that any satisfying assignment of ϕ corresponds to a (P_3, H)-good coloring of G_ϕ, and vice versa. To complete the proof we must show that the gadgets described exist and that no new H is formed while combining the gadgets during the reduction.

Note that it is possible for a variable gadget to have a coloring where both unnegated and negated output vertices are nonfree. This does not affect the validity of the gadgets. A necessary restriction is that if variable x appears unnegated in clause L_1 and negated in L_2, then x cannot satisfy both clauses. Our gadgets clearly impose that restriction.

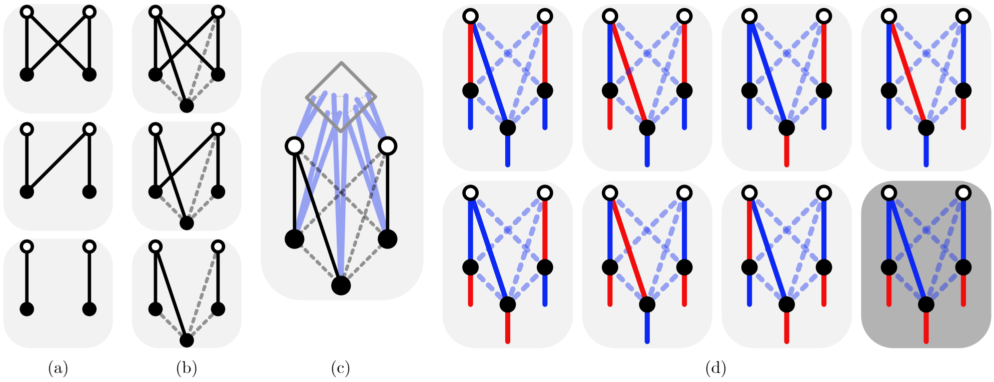


Figure 6 This figure shows the clause gadget used in the proofs of Theorem 13 and 14. **(a)** Each block represents the induced subgraph in a H when: *(1)* $\text{mepl}(H) = 2$ and H has an induced C_4, *(2)* $\text{mepl}(H) = 1$, and *(3)* $\text{mepl}(H) = 0$. **(b)** Each block represents how a fifth vertex, denoted e in the proofs of Theorems 13 and 14, may be connected to the induced subgraphs from (a). For each case, the solid line going from e represents an edge that must exist in $E(H)$ since $d_H(v) \geq 2$ for all $v \in H$ due to 2-connectivity. In the first case, any edge can be chosen w.l.o.g. due to the symmetry of C_4. The dashed edges may or may not exist, but their existence is inconsequential to the correctness of our gadget. **(c)** An illustration of the clause gadget, where each vertex of H attached to an enforcer is in the square. The input vertices have been filled in. **(d)** We show the eight possible combinations of inputs that can be given to the gadget. Observe that a (P_3, H)-good coloring is always possible unless the input is three red edges.

4.2 Hardness proofs

Using the ideas and gadgets presented in Section 4.1, we provide our hardness results below.

▶ **Theorem 13.** *(P_3, H)-Arrowing is coNP-complete when H is a 2-connected graph on at least four vertices with* $\text{mepl}(H) \leq 1$.

Proof. We reduce $(2, 2)$-3SAT to (P_3, H)-Nonarrowing as described in the end of Section 4.1; given a $(2, 2)$-3SAT formula ϕ, we construct a graph G_ϕ such that G_ϕ is (P_3, H)-good if and only if ϕ is satisfiable. Since we have $\text{mepl}(H) \leq 1$, we must have two edges $(a, b), (c, d) \in E(H)$ that have at most one edge adjacent to both. We construct a graph A like so: take a copy of H and append an enforcer to each vertex of H except a, b, c, and d. We construct the variable gadget using two copies of A joined by signal extenders, as shown in Figure 5. The vertices labeled **U** (resp., **N**) correspond to unnegated (resp., negated) output vertices. Note that there are no rogue H's made during this construction. Recall that because of 2-connectivity, a copy of H cannot go through a single vertex. Thus, if a copy of H other than the ones in A and the signal extenders exists, it must go through both copies of A and the signal extenders. However, this is not possible because by including the two signal extenders, we would have a copy of H with at least $2|V(H)|$ vertices.

We now describe our clause gadget. As observed in Section 3.2, there is no 2-connected graph with $\text{mepl}(\cdot) \leq 1$ on four vertices so we assume $|V(H)| \geq 5$. Let (a, b) and (c, d) be the edges that achieve this $\text{epl}(\cdot)$. Let e be a fifth vertex connected to at least one of $\{a, b, c, d\}$. We construct a clause gadget by taking a copy of H and appending an enforcer to each vertex except a, b, c, d, and e. In Figure 6 – which also includes a special case for $\text{mepl}(H) = 2$ used in Theorem 14 – we show how e and two vertices from $\{a, b, c, d\}$ can be used as input vertices so that a blue H is formed if and only if all three input vertices are connected to red external edges. Observe that we can make the clause gadget arbitrarily large by attaching

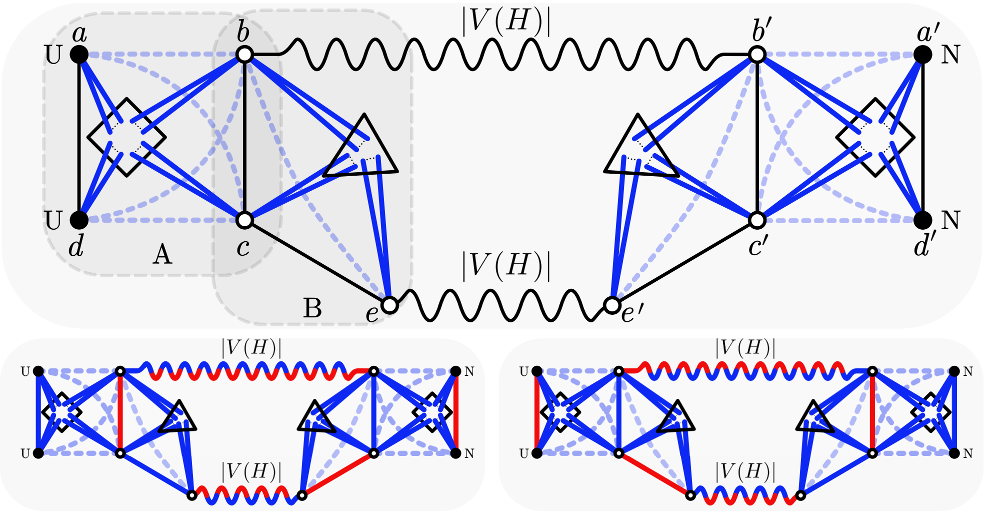


Figure 7 On the top, we show the variable gadget constructed using two copies of F as described in Theorem 13's proof and signal extenders. A and B have been marked. The vertices in the square (resp., triangle) are the vertices of A (resp., B) which had enforcers appended to them. Dashed edges signify edges that may or may not exist. Edges have been precolored wherever possible. Observe that (b, e) must be blue: if (b, e) is red, the attached extenders will force (b', c'), (b', e'), and (c', e') to be blue, forming a blue H in the copy of B on the right. We show that (a, b), (a, c), (b, d), and (c, d) must always be blue. Observe that at least one edge in $\{(b, c), (c, e)\}$ must be red, otherwise we form a blue H in B. Note that if (b, c) is red, the edges (a, b), (a, c), (b, d), and (c, d) must be blue. If (b, c) is blue, (c, e) is red. Thus, (c, d) and (a, c) are blue. Moreover, a red (c, e) forces (c', e') to be blue via the extender. Thus, (b', c') must be red to avoid a blue H. The extender on the top will in turn force edge (a, b) and (b, d) to be blue. Therefore, (a, b), (a, c), (b, d), and (c, d) are blue in all good colorings. By symmetry, (a', b'), (a', c'), (b', d'), (c', d'), and (b', e') must also be blue in all good colorings. Observe that when a vertex marked **U** is nonfree, i.e., (a, d) is blue, (b, c) must be red. Thus, (b', c') is blue, and (a', d') must be red, making the vertices marked **N** nonfree. A similar pattern can be observed when vertices marked **N** are free, wherein the vertices marked **U** are forced to be nonfree. These colorings are shown at the bottom of the figure.

the out-vertex of a signal extender to each input vertex of a clause gadget. We attach a signal extender with at least $|V(H)|$ vertices to each input vertex. This ensures that no copies of H other than the ones in each gadget and extender are present in G_ϕ. ◀

▶ **Theorem 14.** *(P_3, H)-Arrowing is coNP-complete when H is a 2-connected graph on at least four vertices with* $\operatorname{mepl}(H) \geq 2$.

Proof. We follow the same argument as in the proof of Theorem 13. We first discuss the variable gadget. Using Lemma 3 and the fact that $\operatorname{mepl}(H) \geq 2$, we know that we can construct a graph F with exactly two copies of H that share a single edge, unless $H = J_4$. The variable gadget for this exception is shown in Appendix A. For now, we assume $H \neq J_4$. Let A and B the copies of H in F and let (b, c) be the edge that was identified. Let $e \neq b$ be a vertex in B adjacent to c, and let (a, d) be an edge in A where $a, d \notin \{b, c\}$. Note that if such an (a, d) does not exist in H, then (b, c) must be an edge that shares a vertex with every other edge in H. However, it is easy to see that in this case two copies of H cannot be identified on (b, c) without forming new copies of H. Thus, (a, d) must exist. We now append enforcers to each vertex in A and B except a, b, c, d, and e. Our variable gadget is constructed using two copies of F joined via signal extenders as shown in Figure 7.

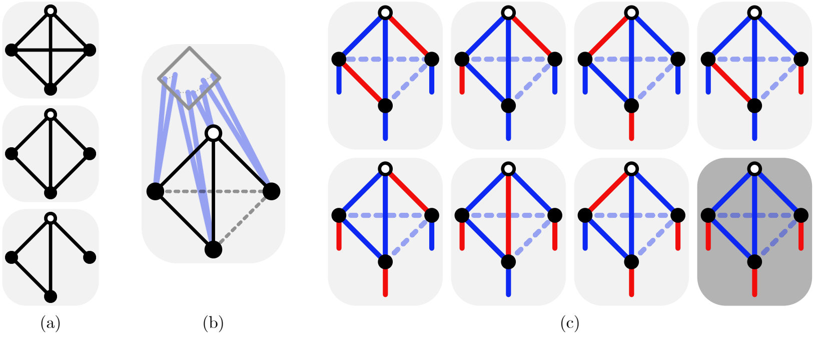


Figure 8 This figure shows the clause gadget used in Theorem 14 when H contains a TK_3. **(a)** Each block represents the induced subgraph in a H when: *(1)* $\mathrm{mepl}(H) = 4$, *(2)* $\mathrm{mepl}(H) = 3$, and *(3)* $\mathrm{mepl}(H) = 2$ and H has an induced TK_3. **(b)** An illustration of the clause gadget, where each vertex of H attached to an enforcer is in the square. The input vertices have been filled in. Dashed edges may or may not exist, but their existence is inconsequential to the correctness of our gadget. **(c)** We show the eight possible combinations of inputs that can be given to the gadget. Observe that a (P_3, H)-good coloring is always possible unless the input is three red edges.

We use a clause gadget similar to the one used in Theorem 13, but with some modifications. Since $\mathrm{mepl}(H) \geq 2$, we know that H must contain a C_4 or a TK_3. If H contains a TK_3, we can use the gadget shown in Figure 8. However, if $\mathrm{mepl}(H) = 2$ and we only have induced C_4's, we can use the gadget shown in Figure 6 when $|V(H)| \geq 5$ using the induced C_4 and another vertex e. The only case left is $H = C_4$, which is discussed in Appendix A. ◀

5 Extending the hardness of $(\mathrm{P_3}, \mathrm{H})$-Arrowing

In this section, we discuss how hardness results for (P_3, H)-Arrowing can be extended to other (F, H)-Arrowing problems. We believe this provides an easier method for proving hardness compared to constructing SAT gadgets. We discuss two methods in which our results can be extended: *(1)* showing that $G \to (P_3, H) \iff G \to (P_3, H')$ for some pairs of H and H' (Section 5.1), and *(2)* given a graph G, showing how to construct a graph G' such that $G \to (P_3, H) \iff G' \to (F, H)$ for some F (Section 5.2).

5.1 $\mathrm{P_3}$ versus tailed complete graphs

We first observe that edges not belonging to H can be removed while working on (P_3, H)-Arrowing for a graph G; we can always color said edge blue without forming a blue H.

▶ **Observation 15.** *Let G be a graph and $e \in E(G)$. If e does not belong to a copy of H in G, then $G \to (P_3, H)$ if an only if $G - e \to (P_3, H)$.*

▶ **Theorem 16.** *For $n \geq 3$, $G \to (P_3, TK_n)$ if and only if $G \to (P_3, K_n)$.*

Proof. Clearly if $G \to (P_3, TK_n)$ then $G \to (P_3, K_n)$ since K_n is a subgraph of TK_n. For the other direction, consider a graph G such that $G \to (P_3, K_n)$ but is (P_3, TK_n)-good. By Observation 15, we can assume that each edge in G belongs to a K_n. We can also assume that G is connected. Let c be a (P_3, TK_n)-good coloring of G. Since $G \to (P_3, K_n)$ there must exist a blue K_n in c. Let $U = \{u_1, u_2, \ldots, u_n\}$ be the vertices of said K_n. Let $e = (u_i, v)$ be an edge going from some $u_i \in U$ to a vertex $v \notin U$. We know that such an edge exists otherwise G is just a K_n, and a K_n is (P_3, K_n)-good, so this would contradict our assumption.

W.l.o.g., let $u_i = u_1$. Note that (u_1, v) must be red, otherwise, we have a blue TK_n. Since (u_1, v) is part of a K_n (Observation 15), at least one vertex w must be connected to both u_1 and v. Note that (u_1, w) and (v, w) must be blue to avoid a red P_3. If $w \in U$, then U and (v, w) form a blue TK_n. If $w \in V(G) - U$, then U and (u_1, w) form a blue TK_n. ◀

▶ **Corollary 17.** *(P_3, TK_n)-Arrowing is coNP-complete when $n \geq 4$ and in P when $n = 3$.*

With this result, we have categorized the complexity of all (P_3, H)-Arrowing problems for connected H with $|V(H)| \leq 4$; the star and path graphs were shown to be in P [4, 6].

5.2 Stars versus 2-connected graphs

Note that $P_3 = K_{1,2}$. Given a graph G, suppose we construct a graph G' by taking a copy of G and appending an edge (one for each vertex in G) to each vertex, where each appended edge is forced to be red in all colorings. It is easy to see that if a coloring of G contains a red $K_{1,2}$, then said coloring in G' contains a red $K_{1,3}$, using the appended red edge. Thus, if we can find a $(K_{1,3}, H)$-good graph F with an edge (u, v) such that, in all good colorings, (u, v) is red and no other edge adjacent to v is red, we could reduce $(K_{1,2}, H)$-Arrowing to $(K_{1,3}, H)$-Arrowing by appending a copy of F (identifying v) to each vertex of G. Recall that we do not have to worry about new copies of H due to its 2-connectivity.

Generalizing this argument, if we attach k red edges to each vertex of G, then a coloring of G with a red $K_{1,\ell}$ corresponds to a coloring of G' with a red $K_{1,k+\ell}$. In Appendix B, we show that for all $n \geq 3$, there exists some m for which there is a $(K_{1,n}, H)$-good graph with a vertex v that is always the center of a $K_{1,m}$ for some $1 \leq m < n$. This allows us to reduce $(K_{1,n-m}, H)$-Arrowing to $(K_{1,n}, H)$-Arrowing. Thus, we can assert the following:

▶ **Lemma 18.** *Suppose H is a 2-connected graph. If $(K_{1,k}, H)$-Arrowing is coNP-hard for all $2 \leq k < n$, then $(K_{1,n}, H)$-Arrowing is also coNP-hard.*

Also in Appendix B, we reduce $(2, 2)$-3SAT to $(K_{1,3}, K_3)$-Nonarrowing and show how that result can be extended to $(K_{1,n}, K_3)$-Arrowing for $n \geq 4$, giving us the following result:

▶ **Theorem 19.** *For all 2-connected H and $n \geq 2$, $(K_{1,n}, H)$-Arrowing is coNP-complete with the exception of $(K_{1,2}, K_3)$-Arrowing, which is in P.*

Finally, recall that (P_3, H)-Nonarrowing is equivalent to H-free Matching Removal. We can assert a similar equivalence between H-free b-Matching Removal and $(K_{1,b+1}, H)$-Nonarrowing, giving us the following corollary:

▶ **Corollary 20.** *For all 2-connected H, H-free b-Matching Removal is NP-complete for all $b \geq 1$, except the case where $b = 1$ and $H = K_3$, which is in P.*

6 Conclusion and future work

This paper provided a complete categorization for the complexity of (P_3, H)-Arrowing when H is 2-connected. We provided a polynomial-time algorithm when $H = K_3$, and coNP-hardness proofs for all other cases. Our gadgets utilized a novel graph invariant, minimum edge pair linkage, to avoid unwanted copies of H. We showed that our hardness results can be extended to (P_3, TK_n)- and $(K_{1,n}, H)$-Arrowing using easy-to-understand graph transformations.

Our ultimate goal is to categorize the complexity of all (F, H)-Arrowing problems. Our first objective is to categorize the complexity of (P_3, H)-Arrowing for all H, and to find more graph transformations to extend hardness proofs between different arrowing problems.

References

1 P. Berman, M. Karpiński, and A. Scott. Approximation Hardness of Short Symmetric Instances of MAX-3SAT. *ECCC*, 2003.

2 A. Bikov. *Computation and Bounding of Folkman Numbers.* PhD thesis, Sofia University "St. Kliment Ohridski", June 2018.

3 S.A. Burr. On the Computational Complexity of Ramsey-Type Problems. *Mathematics of Ramsey Theory, Algorithms and Combinatorics*, 5:46–52, 1990.

4 S.A. Burr, P. Erdős, and L. Lovász. On graphs of Ramsey type. *Ars Combinatoria*, 1(1):167–190, 1976.

5 J. Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

6 Z.R. Hassan, E. Hemaspaandra, and S. Radziszowski. The Complexity of (P_k, P_ℓ)-Arrowing. In *FCT 2023*, volume 14292, pages 248–261, 2023.

7 C.V.G.C. Lima, D. Rautenbach, U.S. Souza, and J.L. Szwarcfiter. Decycling with a Matching. *Information Processing Letters*, 124:26–29, 2017.

8 C.V.G.C. Lima, D. Rautenbach, U.S. Souza, and J.L. Szwarcfiter. Bipartizing with a Matching. In *COCOA 2018*, pages 198–213, 2018.

9 C.V.G.C. Lima, D. Rautenbach, U.S. Souza, and J.L. Szwarcfiter. On the Computational Complexity of the Bipartizing Matching Problem. *Ann. Oper. Res.*, 316(2):1235–1256, 2022.

10 S. Radziszowski. Small Ramsey Numbers. *Electronic Journal of Combinatorics*, DS1:1–116, January 2021. URL: `https://www.combinatorics.org/`.

11 V. Rosta. Ramsey Theory Applications. *Electronic Journal of Combinatorics*, DS13:1–43, December 2004. URL: `https://www.combinatorics.org/`.

12 V. Rutenburg. Complexity of Generalized Graph Coloring. In *MFCS 1986*, volume 233 of *Lecture Notes in Computer Science*, pages 573–581. Springer, 1986.

13 M. Schaefer. Graph Ramsey Theory and the Polynomial Hierarchy. *Journal of Computer and System Sciences*, 62:290–322, 2001.

A Proof of Claim 4 and missing gadgets

▷ Claim 4. If Z exists in $A_{H,e}$, the following must be true: *(1)* Both V_{Z_X} and V_{Z_Y} are nonempty, *(2)* $u \in V(Z)$ and $v \in V(Z)$, *(3)* at least one of E_{Z_X} and E_{Z_Y} is empty, and *(4)* there exists $w \in V(H)$ with $d_H(w) = 2$.

Proof.

1. This follows from our definition of Z.
2. If at most one vertex in $\{u, v\}$ were in Z, deleting it would disconnect said copy of H, contradicting the fact that H is 2-connected.
3. Suppose that both E_{Z_X} and E_{Z_Y} are nonempty. Let $f_1 \in E_{Z_X}$ and $f_2 \in E_{Z_Y}$. We have $\text{epl}_{A_{H,e}}(f_1, f_2) = 0$ since, by construction, $E_{A_{H,e}}(X - Y, Y - X) = \emptyset$. Since both of these edges belong to Z, which is isomorphic to H, we also have $\text{epl}_H(f_1, f_2) = 0$, which contradicts our assumption that $\text{mepl}(H) \geq 2$.
4. Note that k-connected graphs must have minimum degree at least k as a vertex with fewer neighbors could be disconnected from the rest of the graph with fewer than k vertex deletions. So, we have $d_H(w) \geq 2$ for each $w \in V(H)$. W.l.o.g., assume that $E_{Z_X} = \emptyset$. Then, each vertex in $w \in Z_X$ can only be connected to u and v. Thus, we have $d_H(w) = d_Z(w) \leq 2$, and consequently $d_H(w) = 2$. ◁

Missing gadgets for $(\mathbf{P_3}, \mathbf{H})$-Nonarrowing. In Figure 9 we show the variable gadget for (P_3, J_4)-Nonarrowing. In Figure 10 we show the clause gadget for (P_3, C_4)-Nonarrowing.

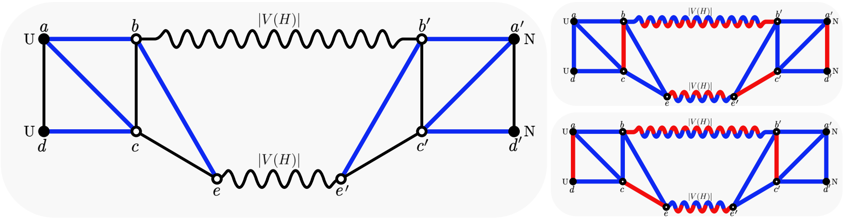


Figure 9 The variable gadget for (P_3, J_4)-Nonarrowing is shown on the left. The colorings on the right show that when vertices marked **U** (resp., **N**) are free, those marked **N** (resp., **U**) are nonfree.

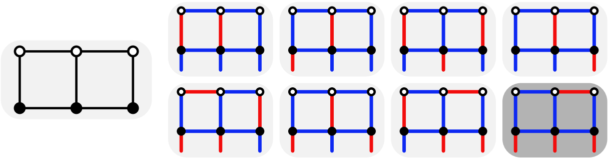


Figure 10 The clause gadget for (P_3, C_4)-Nonarrowing is shown on the left. Possible inputs for the gadget are shown on the right. A good coloring is possible unless the input is three red edges.

B Extending to stars

We first define the following special graph which will be used in our extension proof.

▶ **Definition 21.** *A graph G is called an (F, H)-leaf sender with leaf-signal edge (u, v) if it is (F, H)-good, (u, v) is red in all good colorings, and there exists a good coloring where (u, v) is not adjacent to any other red edge.*

When the context is clear, we will use the shorthand **append a leaf sender to $\boldsymbol{u} \in \boldsymbol{V(G)}$** to mean we will add an (F, H)-leaf sender to G and identify a vertex of its leaf-signal edge with u. This graph essentially simulates "appending a red edge" as described in Section 5.2.

B.1 Proof of Lemma 18

▶ **Lemma 18.** *Suppose H is a 2-connected graph. If $(K_{1,k}, H)$-Arrowing is coNP-hard for all $2 \le k < n$, then $(K_{1,n}, H)$-Arrowing is also coNP-hard.*

Proof. Suppose we are trying to prove the hardness of $(K_{1,n}, H)$-Arrowing for 2-connected H and $n \ge 3$. Let A be a "minimally bad" graph such that $A \to (K_{1,n}, H)$, but removing any edge e from A gives a $(K_{1,n}, H)$-good graph. Let $e = (u, w)$ and $A' = A - e$. Let $\mathcal{C}$ be the set of all good colorings of A'. For a coloring $c \in \mathcal{C}$ and vertex v, let $r_c(v)$ be the number of red edges that v is adjacent to in c. Let $r_{\mathcal{C}}(v) = \min_{c \in \mathcal{C}} r_c(v)$. We consider different cases for $r_{\mathcal{C}}(u)$. Since c is a good coloring, we know that $r_{\mathcal{C}}(u) \le n - 1$.

1. $r_{\mathcal{C}}(u) = n - 1$. In this case, it is easy to see that A' is a $(K_{1,n}, H)$-enforcer with signal vertex u. Let $(p, q) \in V(H)$. Construct a graph B like so. Take a copy of H and append an enforcer to each vertex of H except p and q. It is easy to see that (p, q) must be red in all good colorings, i.e., B is a $(K_{1,n}, H)$-leaf sender with leaf-signal edge (p, q). For any graph G, we append a leaf-sender to each of G's vertices to obtain a graph G' such that $G' \to (K_{1,n}, H) \iff G \to (K_{1,n-1}, H)$, as discussed in Section 5.2.
2. $r_{\mathcal{C}}(u) = 0$. Let $m = |V(H)|$. In this case, we can construct a $(K_{1,n}, H)$-enforcer, B, by combining $2m$ copies of A' as we did in Lemma 8. Note that in any good coloring c of A', we have that $r_c(u) \ge n - 1$ or $r_c(w) \ge n - 1$; if not, such a coloring and a red (u, w) gives

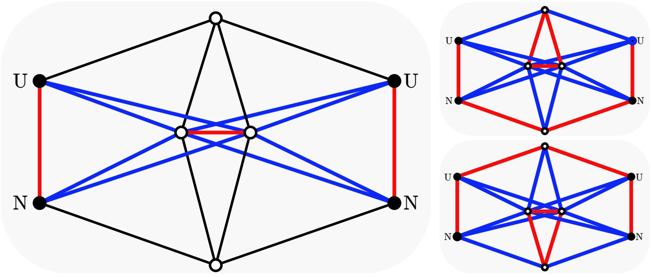


Figure 11 The variable gadget for $(K_{1,3}, K_3)$-Nonarrowing is shown on the left. Edges with the same color in all good colorings have been pre-colored. Both good colorings are shown on the right.

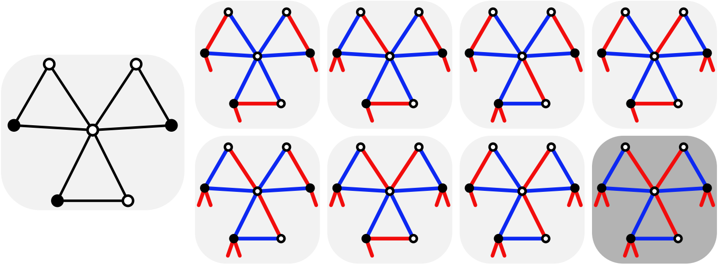


Figure 12 The clause gadget for $(K_{1,3}, K_3)$-Nonarrowing is shown on top. The eight combinations of inputs that can be given to the gadget are shown on the bottom. Observe that a $(K_{1,3}, K_3)$-good coloring is always possible unless the input is three red $K_{1,2}$'s.

a good coloring for A, contradicting the fact that $A \to (K_{1,n}, H)$. Make $2m$ copies of A', where u_i (resp., w_i) refers to the vertex u (resp., w) in the i^{th} copy of A', referred to as A'_i. Now, identify each w_i with u_{i+1} for $i \in \{1, 2, \ldots, 2m-1\}$, and identify w_{2m} with u_1. Observe that when w_1 is adjacent to $n-1$ red edges in A'_1, then u_2 cannot be adjacent to any red edge in A'_2, causing w_2 to be adjacent to $n-1$ red edges in A'_2, and so on. Since every u_i and w_i is adjacent to $n-1$ red edges, any of them can be our signal vertex v. We can now proceed as we did in the previous case to reduce from $(K_{1,n-1}, H)$-Arrowing.

3. $r_{\mathcal{C}}(u) \in \{1, 2, \ldots, n-2\}$. For any graph G, we can attach a copy of A' to each of G's vertices – identifying $u \in V(A')$ with each vertex – to obtain a graph G' such that $G' \to (K_{1,n}, H) \iff G \to (K_{1,n-r_{\mathcal{C}}(u)}, H)$, as discussed in Section 5.2, thereby providing a reduction from $(K_{1,n-r_{\mathcal{C}}(u)}, H)$-Arrowing. ◀

B.2 Hardness of $(\mathbf{K_{1,n}}, \mathbf{K_3})$-Nonarrowing for $n \geq 3$

To show that $(K_{1,3}, K_3)$-Arrowing is coNP-complete, we provide gadgets as we did for (P_3, H)-Arrowing. We provide gadgets in Figures 11 and 12 to show that $(2,2)$-3SAT can be reduced to $(K_{1,3}, K_3)$-Nonarowing. Note that the output vertices are either attached to a single red edge or two red edges. When they are attached to a single red edge, they behave like true output signals. When adjacent to two red edges, they behave like a false output signal. The clause gadget behaves like an OR gate, in that it has no good coloring when the three input vertices all have false inputs.

Recall that in the hardness proofs for (P_3, H)-Arrowing, we also had to show that no new H is constructed while combining gadget graphs to construct G_ϕ. It is easy to see that no new K_3 is constructed when our gadgets are combined; since each clause has unique literals, a cycle formed while constructing G_ϕ would have to go through at least two clause gadgets and at least two variable gadgets, but this cycle has more than three vertices (see Figure 13).

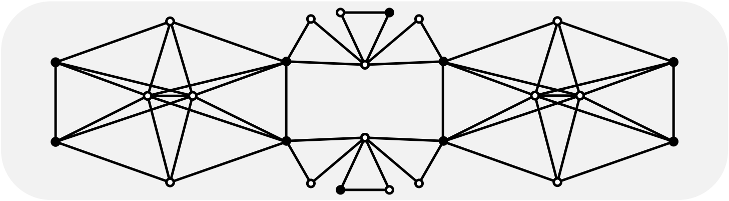


Figure 13 The smallest cycle made when joining $(K_{1,3}, K_3)$ variable and clause gadgets is a C_6.

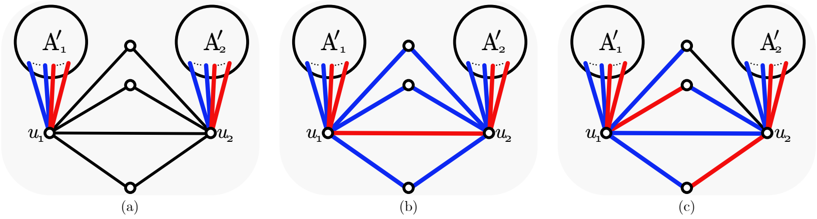


Figure 14 A'_1 and A'_2 are graphs with $r_{\mathcal{C}}(u_i) = n-2$. We show how to construct a new graph using these in **(a)**. In **(b)**, we show a good coloring where u_i's are now adjacent to $n-1$ red edges. Finally, in **(c)**, we observe that the coloring in **(b)** is the only good coloring since at most one edge from outside A'_i that is adjacent to u_i can be red.

To show the hardness of $(K_{1,n}, K_3)$-Arrowing for $n \geq 4$, we can proceed exactly as we did in Lemma 18. The only case where the proof fails is when $r_{\mathcal{C}}(u) = n-2$, because now the proof says we have to reduce $(K_{1,2}, K_3)$-Arrowing to $(K_{1,n}, K_3)$-Arrowing, which is unhelpful since $(K_{1,2}, K_3)$-Arrowing is in P. In Figure 14 we show how vertices attached to $n-2$ red edges can be combined to make a $(K_{1,n}, K_3)$-enforcer. Using the enforcer, we can create a $(K_{1,n}, K_3)$-leaf sender and reduce from $(K_{1,n-1}, K_3)$-Arrowing.

On the Complexity of Community-Aware Network Sparsification

Emanuel Herrendorf
Philipps-Universität Marburg, Fachbereich Mathematik und Informatik, Germany

Christian Komusiewicz
Friedrich Schiller University Jena, Institute of Computer Science, Germany

Nils Morawietz
Friedrich Schiller University Jena, Institute of Computer Science, Germany

Frank Sommer
Friedrich Schiller University Jena, Institute of Computer Science, Germany

Abstract

In the NP-hard Π-Network Sparsification problem, we are given an edge-weighted graph G, a collection $\mathcal{C}$ of c subsets of $V(G)$, called *communities*, and two numbers ℓ and b, and the question is whether there exists a spanning subgraph G' of G with at most ℓ edges of total weight at most b such that $G'[C]$ fulfills Π for each community $C \in \mathcal{C}$. We study the fine-grained and parameterized complexity of two special cases of this problem: Connectivity NWS where Π is the connectivity property and Stars NWS, where Π is the property of having a spanning star.

First, we provide a tight $2^{\Omega(n^2+c)}$-time running time lower bound based on the ETH for both problems, where n is the number of vertices in G even if all communities have size at most 4, G is a clique, and every edge has unit weight. For the connectivity property, the unit weight case with G being a clique is the well-studied problem of computing a hypergraph support with a minimum number of edges. We then study the complexity of both problems parameterized by the feedback edge number t of the solution graph G'. For Stars NWS, we present an XP-algorithm for t answering an open question by Korach and Stern [Discret. Appl. Math. '08] who asked for the existence of polynomial-time algorithms for $t = 0$. In contrast, we show for Connectivity NWS that known polynomial-time algorithms for $t = 0$ [Korach and Stern, Math. Program. '03; Klemz et al., SWAT '14] cannot be extended to larger values of t by showing NP-hardness for $t = 1$.

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

Keywords and phrases parameterized complexity, hypergraph support, above guarantee parameterization, exponential-time-hypothesis

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.60

Related Version *Continuously Updated Version*: https://arxiv.org/abs/2402.15494

Funding *Nils Morawietz*: Partially supported by the DFG, project OPERAH (KO 3669/5-1).
Frank Sommer: Partially supported by the DFG, project EAGR (KO 3669/6-1)

Acknowledgements Some of the results of this work are also contained in the first author's Masters thesis [23].

1 Introduction

A common goal in network analysis is to decrease the size of a given network to speed up downstream analysis algorithms or to decrease the memory footprint of the graphs. This leads to the task of network sparsification where one wants to reduce the number of edges of a network while preserving some important property Π [6, 31, 35]. Similarly, in network

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 60; pp. 60:1–60:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

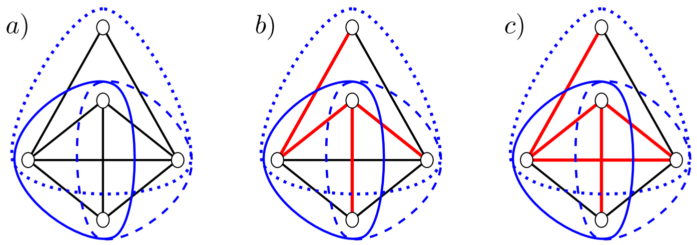


Figure 1 *a*) The communities (blue) and input graph of a Π-NWS instance. *b*) and *c*) Optimal solutions (red) for Unweighted Connectivity NWS, and Unweighted Stars NWS, respectively.

design the task is often to construct a minimum-size or minimum-weight network fulfilling a given property, the most famous example being Minimum-Weight Spanning Tree.

In many applications the input contains, in addition to a network, a hypergraph on the same vertex set [17, 24, 32]. The hyperedges of this hypergraph represent, for example, communities that are formed within the network. In presence of such community data, the sparsified network should preserve a property not for the whole network but instead for each community, that is, for each hyperedge of the hypergraph. Gionis et al. [19] called this task *community-aware network sparsification* and formalized it as follows.

Π-Network Sparsification (Π-NWS)

Input: A graph G, a collection $\mathcal{C}$ of c subsets of $V(G)$, called *communities*, an edge-weight function $\omega : E(G) \to \mathbb{R}_+$, an integer ℓ, and a positive real number b.
Question: Is there a graph $G' = (V(G), E')$ with $E' \subseteq E(G)$, $|E'| \leq \ell$, and total edge weight at most b such that for each community $C_i \in \mathcal{C}$ the subgraph of G' induced by C_i satisfies Π?

We say that a graph G' fulfilling the requirements is a *solution* for the instance I. A very well-studied property Π, considered by Gionis et al. [19] but also in many previous works [2, 7, 13, 15, 28] is that every community should induce a connected subgraph. A graph G that has this property for some hypergraph H, is called a *support* for H [4, 5, 28]. We denote the corresponding special case of Π-NWS as Connectivity NWS. Another variant of Π-NWS, also studied by Gionis et al. [19], is to demand that every community not only induces a connected subgraph but more strongly that it contains a *spanning star*. In other words, in the solution graph G', every community must be contained in the neighborhood of at least one of its vertices, called a *center vertex*. We refer to this variant as Stars NWS. An example instance and solutions for both problems are given in Figure 1.

Connectivity NWS and Stars NWS are both NP-hard [13, 10, 9, 19]. Motivated by this, we study both problems in terms of their parameterized and fine-grained complexity. We also investigate the versions of both problems where each edge has unit weight and refer to them as Unweighted Connectivity NWS and Unweighted Stars NWS.

Our two main results are as follows:

- We show that, based on the Exponential Time Hypothesis (ETH), Connectivity NWS and Stars NWS do not admit algorithms with running time $2^{o(n^2)+c}$, even if the input graph is a clique with unit weights and each community has size at most 4. This bound is matched by simple brute-force algorithms.
- We show that Stars NWS admits an XP-algorithm when parameterized by t, the feedback edge number of the solution graph. This positively answers the question of Korach and Stern [30] who asked whether there is a polynomial-time algorithm for finding an optimal solution for Stars NWS that is a tree. In fact, our algorithm extends the polynomial-time solvable cases to solutions that are tree-like.

We obtain several further results, for example a complexity dichotomy for Stars NWS and Unweighted Stars NWS parameterized by c, the number of communities.

Known results. Already the most basic variant of Connectivity NWS, where the edges have unit weights and the input graph G is a clique, appears in many applications, ranging from explanation of protein complexes [32] to combinatorial auctions [10] to the construction of P2P overlay networks in publish/subscribe systems [8, 24]. Thus, the problem has been studied intensively under various names [2, 7, 8, 13, 15, 24] from a parameterized complexity [7, 15, 24] and an approximation algorithms [2, 8, 24] perspective. For example, the problem is NP-hard even for instances with maximum community size 3 [13], and admits FPT-algorithms for the number of communities and for the largest community size plus the feedback edge number t of a solution [7]. A particular restriction of the problem is to determine whether there is an acyclic solution, called *tree support* or *clustered spanning tree.* It can be determined in polynomial time whether a hypergraph has a tree support and different polynomial-time algorithms have been described over the years [3, 10, 14, 16, 20, 27, 33, 34].

Unweighted Connectivity NWS with general input graphs G, has applications in the context of placing green bridges [17, 22]. Unweighted Connectivity NWS is NP-hard even when the maximum degree of G is 3 [22] and even for seven communities [17]. On the positive side, one can construct in polynomial time a tree support if one exists [21, 28, 29].

For Connectivity NWS where we may have arbitrary edge-weights, the distinction whether or not G is restricted to be a clique vanishes: any non-clique input graph G may be transformed into a clique by adding the missing edges with a prohibitively large edge weight. The problem of finding a minimum-weight tree support received attention due to its applications in network visualization [28]. As shown by Korach and Stern [29] and Klemz et al. [28], one can compute minimum-weight tree supports in polynomial time. Gioinis et al. [19] provided approximation algorithms for the general problem.

Stars NWS has received less attention than Connectivity NWS. Gionis et al. [19] showed NP-hardness and provided approximation algorithms. Korach and Stern [30] studied a variant of Stars NWS where the input graph is a clique and the solution is constrained to be a tree T where the closed neighborhood of the center vertex of a community C_i is exactly the community C_i. This implies that two different communities need to have different center vertices and thus restricts the allowed set of solution graphs strictly compared to Stars NWS. Korach and Stern [30] showed that this problem is solvable in polynomial time. As an open question, they ask whether this positive result can be lifted to Stars NWS.

Cohen et al. [9] studied the Minimum $\mathcal{F}$-Overlay problem which can be viewed as the following special case of Π-NWS: The input graph G is a clique and all edges have unit weight; $\mathcal{F}$ is a family of graphs and the property Π is to have some spanning subgraph which is contained in $\mathcal{F}$. Unweighted Connectivity NWS and Unweighted Stars NWS with clique input graphs are special cases of Minimum $\mathcal{F}$-Overlay. Cohen et al. [9] provide a complexity dichotomy with respect to properties of $\mathcal{F}$. For most cases of $\mathcal{F}$, Minimum $\mathcal{F}$-Overlay is NP-hard. In particular, Unweighted Stars NWS is NP-hard even when G is a clique [9]. Gionis et al. [19] also studied Π being that each community needs to induce a subgraph exceeding some prespecified density. Fluschnik and Kellerhals [17] considered further properties Π, for example the property of having small diameter.

Our results and organization of the work. To put our main results into context, we first summarize in Section 2 some complexity results that follow from simple observations or from previous work. They imply in particular that Stars NWS and Connectivity NWS have an FPT-algorithm for the parameter solution size ℓ and that they are W[1]-hard with respect to the dual parameter $k := m - \ell$ even in the unit weight case when G is a clique. Then,

Table 1 An overview of the parameterized complexity results. A ‡ indicates that this result also holds in the unweighted case and a † indicates that this result only holds in the unweighted case.

<table>
<tr><th>Parameter</th><th>Stars NWS</th><th>Connectivity NWS</th></tr>
<tr><td>ℓ</td><td colspan="2">FPT (Proposition 2.2, [17]), no polynomial kernel‡ (Proposition 2.4, [17])</td></tr>
<tr><td>$k := m - \ell$</td><td colspan="2">W[1]-hard‡ (Proposition 2.3)</td></tr>
<tr><td>t</td><td>XP (Theorem 4.1)</td><td>P for $t = 0$ ([28])
NP-h for $t = 1^\ddagger$ (Theorem 4.9)</td></tr>
<tr><td>c</td><td>FPT† (Theorem 5.2)
no polynomial kernel‡ (Theorem 5.3)
W[1]-h (Theorem 5.1)</td><td>NP-h for $c = 7^\ddagger$ ([17])</td></tr>
<tr><td>Δ</td><td>NP-h for $\Delta = 6^\ddagger$ (Corollary 3.3)</td><td>NP-h for $\Delta = 3^\ddagger$ ([22])</td></tr>
</table>

in Section 3 we show that Unweighted Connectivity NWS and Unweighted Stars NWS do not admit algorithms with running time $2^{o(n^2+c)}$ even when G is a clique unless the Exponential Time Hypothesis (ETH) [26] is false.

In Section 4, we consider parameterization by t, the feedback edge number of the solution graph G'. This is the minimum number of edges that need to be deleted to transform the solution into a forest.[1] The study of t is motivated as follows: The solution size ℓ is essentially at least as large as $n - 1$, and thus neither small in practice nor particularly interesting from an algorithmic point of view. Thus, t can be seen as a parameterization above the lower bound $n - 1$. Our first main result is an XP-algorithm for Stars NWS. This positively answers the question of Korach and Stern [30] who asked whether there is a polynomial-time algorithm for $t = 0$ and extends the tractability further to every constant value of t. We then show that, in contrast, Unweighted Connectivity NWS is NP-hard already if $t = 1$. Thus, the polynomial-time algorithms for $t = 0$ [29, 28] cannot be lifted to larger values of t.

Finally, in Section 5 we study Stars NWS parameterized by the number c of input communities. We obtain the following complexity classification: Unweighted Stars NWS is FPT with respect to c and Stars NWS is W[1]-hard in the most restricted case when G is a clique and all edges have weight 1 or 2 and in XP in the most general case.

For an overview of the parameterized complexity results, refer to Table 1.

Due to lack of space several proofs (marked with ($\star$)) are deferred to the full version.

2 Preliminaries and Basic Observations

Preliminaries. For a set X, we denote by $\binom{X}{2}$ the collection of all size-two subsets of X. Moreover, for positive integers i and j with $i \leq j$, we denote by $[i, j] := \{k \in \mathbb{N} : i \leq k \leq j\}$.

An *undirected graph* $G = (V, E)$ consists of a set of vertices V and a set of edges $E \subseteq \binom{V}{2}$. We denote by $V(G)$ and $E(G)$ the vertex and edge set of G, respectively, and let $n = n(G) := |V(G)|$ and $m := |E(G)|$. For a vertex set $V' \subseteq V$, we denote by $E_G(V') := \{\{u, v\} \in E : u, v \in V'\}$ the edges between the vertices of V' in G. If G is clear from the context, we may omit the subscript. A graph G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. Moreover, G' is *spanning* if $V(G') = V(G)$. For a vertex set V', we denote by $G[V'] := (V', E_G(V'))$ the *subgraph of G induced by V'*. A set $S \subseteq V(G)$ with

[1] The parameter t can be computed in polynomial time as discussed in Section 2.

$E_G(S) = \binom{S}{2}$ is called a *clique*. A graph G is a *star* of size $n-1$ with *center* $z \in V(G)$ if $E(G) = \{\{z, v\} : v \in V(G) \setminus \{z\}\}$. A graph G contains a *spanning star* if some subgraph G' of G is a star of size $n-1$. The center of this star is *universal* for G.

An edge set $E' \subseteq E(G)$ is a *feedback edge set* of G, if the graph $G' := (V(G), E(G) \setminus E')$ is acyclic. Two vertices u and v are *connected* in G if G contains a path between u and v. A graph G is *connected* if each pair of vertices $u, v \in V(G)$ is connected. A set $S \subseteq V(G)$ is a *connected component* of G, if $G[S]$ is connected and S is maximal with this property. Connectivity in hypergraphs is defined similarly: Two vertices u and v are *connected* if there exists a sequence $C_1, C_2, \dots, C_p$ of hyperedges such that $u \in C_1$, $v \in C_p$, and consecutive communities have nonempty intersection. A *connected component* of a hypergraph is a maximal set of connected vertices. The number x of connected components of a hypergraph can be computed in polynomial time, for example by BFS. Note that for a minimal solution G' for Stars NWS and Connectivity NWS, the connected components of G' are exactly the connected components of the community hypergraph. Thus, $t = \ell - n + x$ and the parameter t can be computed in polynomial time for a given input instance.

For details about parameterized complexity and the ETH refer to [12, 11].

Basic observations. To put our main results for Connectivity NWS and Stars NWS into context, we state some results that either follow easily from previous work or from simple observations.

The naive brute-force approach for each Π-NWS is to perform an exhaustive search over the $\mathcal{O}(2^m)$ possibilities to select at most ℓ edges from the input graph G. This leads to the following general statement for Π-NWS problems.

▶ **Proposition 2.1.** *Let Π be a property which can be decided in $\operatorname{poly}(n)$ time. Then, Π-NWS is solvable in $2^m \cdot c \cdot \operatorname{poly}(n)$ time.*

For the solution size parameter ℓ, one can obtain the following running time.

▶ **Proposition 2.2** ($\star$). *Connectivity NWS and Stars NWS can be solved in $\ell^{\mathcal{O}(\ell)} \cdot \operatorname{poly}(n+c)$ time.*

The fixed-parameter tractability of Unweighted Connectivity NWS with respect to ℓ was also shown by Fluschnik and Kellerhals via a kernelization [17].

A further natural parameter that can be considered is $k := m - \ell$, a lower-bound on the number of edges of G that any solution must omit.

▶ **Proposition 2.3** ($\star$).

- *Connectivity NWS and Stars NWS are NP-hard for $k = 0$.*
- *Unweighted Connectivity NWS and Unweighted Stars NWS can be solved in $n^{2k} \cdot \operatorname{poly}(n)$ time and are $\mathrm{W}[1]$-hard with respect to k even if G is a clique and if each community has size at most 3.*

For Unweighted Connectivity NWS, Fluschnik and Kellerhals showed that a polynomial kernel for ℓ and (thus for n) is unlikely, even on planar series-parallel graphs [17]. This result can be also given for the case when the input graph G is a clique.

▶ **Proposition 2.4** ($\star$). *Unweighted Connectivity NWS and Unweighted Stars NWS do not admit a polynomial kernel for n even if G is a clique, unless NP $\subseteq$ coNP/poly.*

One can also show that the simple brute-force algorithm that considers all $\mathcal{O}(2^m)$ subsets of $E(G)$ cannot be improved substantially.

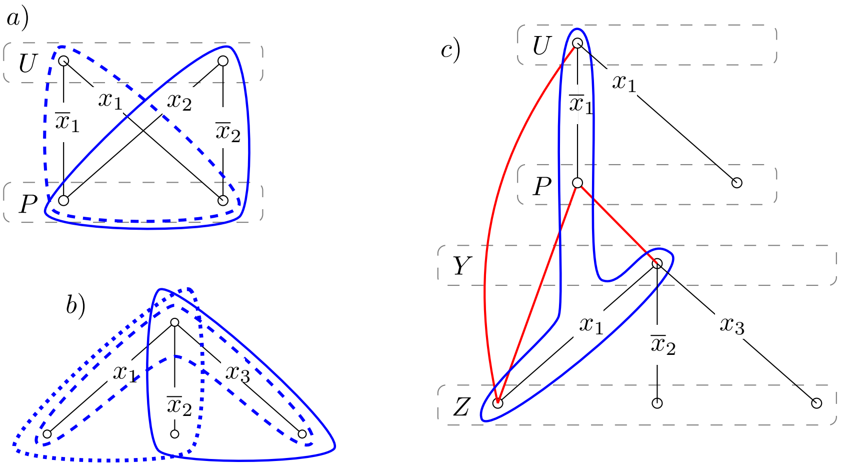


Figure 2 Sketch of the construction of Theorem 3.1. The communities are blue (solid, dashed and dotted). We only show edges which are contained in at least one community and only some fixed edges (red). *a)* Part of the variable gadget for x_1 and x_2. *b)* The variable communities for a clause $q = x_1 \vee \overline{x_2} \vee x_3$. *c)* The assignment gadget for the first literal x_1 of the clause q. Here, the red edges are the fixed edges with one endpoint in the variable gadget and one in the clause gadget.

▶ **Proposition 2.5** ($\star$). *If the ETH is true, then* Unweighted Stars NWS *and* Unweighted Connectivity NWS *cannot be solved in* $2^{o(n+m+c)} \cdot \mathrm{poly}(n+c)$ *time, even if restricted to instances with community size at most* 3.

3 A Stronger ETH-Bound

In Proposition 2.5 we observed that algorithms with running time $2^{o(n+m+c)}$ for Unweighted Connectivity NWS and Unweighted Stars NWS would violate the ETH. We now provide a stronger $2^{\Omega(n^2+c)}$-time lower bound for both problems. Notably, this lower bound also applies to the case when all communities have constant size.

First, we present the lower bound for Unweighted Stars NWS.

▶ **Theorem 3.1.** *If the ETH is true, then* Unweighted Stars NWS *cannot be solved in* $2^{o(n^2+c)}$ *time, even if* G *is a clique and if each community has size at most* 4.

Proof. We reduce from 3-SAT to Unweighted Stars NWS such that the resulting instance has $\mathcal{O}(\sqrt{|\phi|})$ vertices and $\mathcal{O}(|\phi|)$ communities, where ϕ denotes the total formula length. Then, the existence of an $2^{o(n^2+c)}$-time algorithm for Unweighted Stars NWS implies the existence of a $2^{o(|\phi|)}$-time algorithm for 3-SAT violating the ETH [25, 26]. The input formula ϕ is over the variable set X and each clause $q \in \Gamma$ contains exactly three literals. For a literal y, we denote by $\overline{y}$ its complement. A visualization of the construction is given in Figure 2. In all gadgets, we add several communities of size 2. These communities enforce that each solution has to contain the edge of this community. In the following we call such edges *fixed*.

Variable gadget G_X. Recall that G_X is a clique. The idea is to create for each variable a community C of size 3 with one *fixed* edge. The two remaining edges of C are called *selection* edges. The idea is that each solution contains exactly one selection edge of C. One selection edge represents the positive literal, the other one represents the negative literal. The fixed edge of the triangle is used to model that one literal must be set to `true`. The selection edges are arranged compactly, to guarantee that $|V(G_X)| \in \mathcal{O}(\sqrt{|\phi|})$. In the following, we describe the graph G_X together with communities fulfilling the above-described properties. An example of a variable gadget is shown in part $a)$ of Figure 2.

Let $V(G_X) = U \cup P$ where $U \coloneqq \{u_1, \dots, u_{n_x}\}$, $P = P_1 \cup P_2$, and $P_i \coloneqq \{p^i_1, \dots, p^i_{n_x}\}$ for $i \in [2]$ consist of $n_x = \lceil \sqrt{|X|} \rceil$ vertices each. It remains to describe the communities: For each variable $x \in X$, we add a community $C_x \coloneqq \{u_j, p^1_s, p^2_s\}$ for $j, s \in [\, n_x \,]$. This is possible since $n_x \cdot n_x \geq |X|$. These communities are called the *variable communities* $\mathcal{C}^X$. Afterwards, we set $\theta(x) \coloneqq \{u_j, p^1_s\}$ and $\theta(\overline{x}) \coloneqq \{u_j, p^2_s\}$ to assign the positive and negative literal of x to an edge of the variable gadget. Now, we fix the edges of $G[P] = G[P_1 \cup P_2]$. In other words, for each edge $\{p^{i_1}_{j_1}, p^{i_2}_{j_2}\}$ having both endpoints in $P_1 \cup P_2$, we add a community $\{p^{i_1}_{j_1}, p^{i_2}_{j_2}\}$.

Observe that the sets of selection edges corresponding to two distinct variables are disjoint:

▷ Claim 1 ($\star$). Each selection edge of $E(G_X)$ is contained in only one subgraph induced by a variable community in $\mathcal{C}^X$.

Clause gadget. We continue by describing the construction of the clause gadget G_Γ. The idea is that each clause is represented by four vertices of $V(G_\Gamma)$ in which a triangle is *fixed*. All three remaining edges of this size-4 clique are referred to as *free*. Note that these free edges form a star with three leaves. Each free edge represents one literal of the clause. For each pair containing two of these three edges, we then create a community containing the three endpoints of these two edges. As in the vertex gadget, these induced subgraphs are arranged compactly, to achieve a clause gadget with $|V(G_\Gamma)| \in \mathcal{O}(\sqrt{|\Gamma|})$.

Let $V(G_\Gamma) = Y \cup Z$ where $Y = \{y_1, \dots, y_{n_c}\}$, $Z = Z_1 \cup Z_2 \cup Z_3$, and $Z_i = \{z^1_1, \dots, z^i_{n_c}\}$ for $i \in [3]$ consist of $n_c = \lceil \sqrt{|\Gamma|} \rceil$ vertices each. In the following, we assign each clause to a clique of G_Γ having vertex set y_j, z^1_s, z^2_s, z^3_s for $j, s \in [n_c]$. This is possible since $n_c \cdot n_c \geq |\Gamma|$. In this clique, we *fix* the triangle having its endpoints in $Z_1 \cup Z_2 \cup Z_3$. Formally, for each clause $q = \{q_1, q_2, q_3\} \in \Gamma$ we add three communities $C^1_q = \{y_j, z^2_s, z^3_s\}$, $C^2_q = \{y_j, z^1_s, z^3_s\}$ and $C^3_q = \{y_j, z^1_s, z^2_s\}$. We refer to these communities as the *clause* communities $\mathcal{C}^\Gamma$. Afterwards, we set $\nu(q, q_1) \coloneqq \{y_j, z^1_s\}$, $\nu(q, q_2) \coloneqq \{y_j, z^2_s\}$, and $\nu(q, q_3) \coloneqq \{y_j, z^3_s\}$ to assign each literal in clause q to an edge of the clause gadget. These edges are referred to as *free*. Second, we fix the edges of the clique $Z_1 \cup Z_2 \cup Z_3$.

Observe that the sets of free edges corresponding to two distinct clauses are disjoint:

▷ Claim 2 ($\star$). Each free edge of $E(G_\Gamma)$ is contained in exactly one subgraph induced by a clause community in $\mathcal{C}^\Gamma$.

Connecting the gadgets. We complete the construction by connecting the variable and clause gadget, using new *assignment* communities. The idea is to add a new community containing the endpoints of a free edge describing a literal in a clause together with the endpoints of the selection edge describing the same opposite literal in the variable gadget. These communities model occurrences of variables in the clauses. Roughly speaking, these communities are satisfied if the selection edge of the variable gadget or the free edge of the clause gadget is part of the solution. To enforce this, we fix further edges of G.

We create for each clause $q = \{q_1, q_2, q_3\} \in \Gamma$ three assignment communities $C^{q_1}_q = \nu(q, q_1) \cup \theta(\overline{q_1})$, $C^{q_2}_q = \nu(q, q_2) \cup \theta(\overline{q_2})$, and $C^{q_3}_q = \nu(q, q_3) \cup \theta(\overline{q_3})$. We denote the assignment communities with $\mathcal{C}^X_\Gamma$. To enforce that each solution contains the selection edge or the free edge of each assignment community, we fix all edges between the vertex sets U and Z, between the vertex sets P and Y, and between the vertex sets P and Z.

Finally, we set $\ell \coloneqq |X| + 2 \cdot |\Gamma| + \binom{|P|}{2} + \binom{|Z|}{2} + |U| \cdot |Z| + |P| \cdot |Y| + |P| \cdot |Z|$.

Correctness. The correctness is based on the facts that each solution for I contains *a*) all fixed edges, *b*) exactly $|X|$ selection edges, and *c*) exactly $2|\Gamma|$ free edges. Fact *b*) ensures that this models an assignment β of the variables of X and fact *c*) ensures that each clause is satisfied by at least one literal of β. The detailed correctness proof is deferred to the full version. ◀

An adapted construction yields further results for d-DIAM NWS [17] (Π is "having diameter at most d") and DENSITY NWS [19] (Π is "exceeding some density threshold").

▶ **Corollary 3.2** ($\star$). *If the ETH is true, then UNWEIGHTED CONNECTIVITY NWS, d-DIAM NWS for each $d \geq 2$, and DENSITY NWS cannot be solved in $2^{o(n^2+c)}$ time even if G is a clique and each community has size at most* 4.

▶ **Corollary 3.3** ($\star$). *UNWEIGHTED STARS NWS remains NP-hard and, assuming the ETH, cannot be solved in $2^{o(n+m+c)}$ time on graphs with maximum degree six and community size at most* 4.

4 Parameterization by the Feedback Edge Number of a Solution

The parameter ℓ, the number of edges in the solution is in most cases not independent from the size of the input instance of STARS NWS or CONNECTIVITY NWS: if the hypergraph $(V, \mathcal{C})$ is connected, a solution G' has at least $n-1$ edges. In other words, $n-1$ is a lower bound for ℓ in this case. In this section, we study STARS NWS and CONNECTIVITY NWS parameterized above this lower bound. Formally, the parameter t is defined as the size of a minimum feedback edge set of any optimal solution of an instance of STARS NWS or CONNECTIVITY NWS. Thus, the parameter t measures how close the solution is to a forest. Formally, the definition is $t := \ell - n + x$ where x denotes the number of connected components of G'. Recall that t can be computed in polynomial time (see Section 2.).

4.1 An XP-Algorithm for Stars NWS

In this subsection, we show that STARS NWS parameterized by t admits an XP-algorithm.

▶ **Theorem 4.1.** *STARS NWS can be solved in $m^{4t} \cdot \mathrm{poly}(|I|)$ time.*

Our XP-algorithm exploits the fact that there are two different kinds of cycles in G': First, there are *global* cycles. These are the cycles in the solutions that are directly caused by cycles in the input hypergraph. No solution may avoid these cycles. Second, there are *local* cycles. These are cycles which are entirely contained in the subgraph induced by two communities. Since in each solution, each community contains a spanning star, local cycles can only have length 3 or 4. This allows us to bound the number of possible local cycles and thus to consider all possibilities for the local cycles in XP-time with respect to t. Then, the crux of our algorithm is that after all local cycles have been fixed, all remaining cycles added by our algorithm have to be global and are thus unavoidable. Using this fact, we show that in polynomial time we can compute an optimal solution with feedback edge number at most t that extends a fixed set of local cycles without introducing any further local cycles. To do this, for each community C, we store a set of *potential centers*, that is, vertices of C that may be the center of a spanning star of C in any solution that does not produce new local cycles. We define several *operations* that restrict the potential centers of each community. We show that after all operations have been applied exhaustively, one can greedily pick the best remaining center for each community.

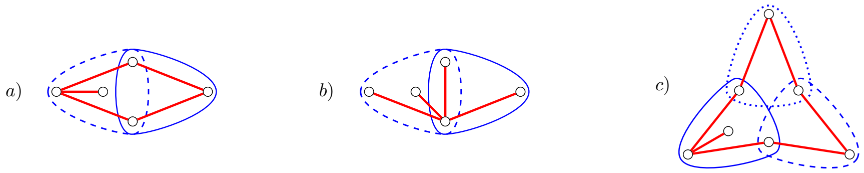


Figure 3 Examples for solutions with and without local cycles. Red edges indicate the edges of the solution. Part a) shows an example, where both communities induce a local cycle. Part b) shows an example, where the two communities do not induce local cycle. Finally, part c) shows an example, where the solution contains a (global) cycle but no two communities induce a local cycle.

Algorithm-specific notation. Next, we present the formal definition of local cycles; an example is shown in Figure 3. For a spanning subgraph H of G and a community $C \in \mathcal{C}$, let $\text{univ}_H(C)$ denote the vertices of C that are *universal* for C in H. Note that $\text{univ}_H(C) \subseteq \text{univ}_G(C)$. In the following, we assume that for each community $C \in \mathcal{C}$, $\text{univ}_G(C) \neq \emptyset$, as otherwise, there is no solution for the instance I of Stars NWS.

For a solution G', we say that two distinct communities C_1 and C_2 *induce a local cycle* if for each $i \in \{1,2\}$, there is a vertex $c_i \in \text{univ}_{G'}(C_i)$ such that the graph $S_1 \cup S_2$ contains a cycle. Here, for each $i \in \{1,2\}$, S_i is the spanning star of C_i with center c_i and $S_1 \cup S_2$ is the union of both these stars defined by $S_1 \cup S_2 := (C_1 \cup C_2, \{\{c_i, w_i\} \colon w_i \in C_i \setminus \{c_i\}, i \in \{1,2\}\})$. Moreover, we say that each cycle of $S_1 \cup S_2$ is a *local cycle* in G'. Note that each local cycle has length at most four, and if C_1 and C_2 induce a local cycle, then $|C_1 \cap C_2| \geq 2$.

As described above, the first step of the algorithm behind Theorem 4.1 is to test each possibility for the local cycles of the solution. For a fixed guess, we let E^* denote the set of all edges contained in at least one local cycle and in the following we refer to them as *local edges*. Moreover, we call a minimum solution G' *fitting for* E^* if each local cycle of G' uses only edges of E^* and each edge of E^* is contained in G'. Hence, to determine whether the choice of local edges E^* can lead to a solution, we only have to check, whether there is a fitting solution for E^*. In the following, we show that this can be done in polynomial time.

▶ **Theorem 4.2.** *Let $I = (G = (V,E), \mathcal{C}, \omega, \ell, b)$ be an instance of* Stars *NWS, and let $E^* \subseteq E$. In polynomial time, we can*

- *find a solution $G' = (V, E')$ for I with $E^* \subseteq E'$ or*
- *correctly output that there is no minimum solution that is fitting for E^*.*

Based on the definition of fitting solutions, we define for each community $C \in \mathcal{C}$ a set $\text{fit}_{E^*}(C)$ of possible centers. We initialize $\text{fit}_{E^*}(C) := \text{univ}_G(C)$ for each community $C \in \mathcal{C}$. The goal is to reduce $\text{fit}_{E^*}(C)$ of each community C as much as possible while preserving the following property, which trivially holds for the initial $\text{fit}_{E^*}(C)$ for each community $C \in \mathcal{C}$.

▶ **Property 1.** *For each minimum solution G' which is fitting for E^* and each community $C \in \mathcal{C}$, we have $\text{univ}_{G'}(C) \subseteq \text{fit}_{E^*}(C)$.*

Note that if Property 1 is fulfilled and if $\text{fit}_{E^*}(C) = \emptyset$ for some $C \in \mathcal{C}$, then we can correctly output that there is no fitting solution for E^*. Next, we define several operations that for some communities $C \in \mathcal{C}$ remove vertices from $\text{fit}_{E^*}(C)$ which – when taken as a center vertex for C – would introduce new local cycles, violating the properties of a fitting solution. We show that these operations preserve Property 1 and that after these operations are applied exhaustively, the task of Theorem 4.2 can be performed greedily based on fit_{E^*}. Examples for each of our operations are shown in Figure 5.

In the following, we say that a vertex $v \in V$ is *locally universal* for a vertex set $A \subseteq V$, if for each vertex $w \in A \setminus \{v\}$, the vertex pair $\{v, w\}$ is a local edge.

▶ **Operation 1.** *Let $C \in \mathcal{C}$ be a community and let $\{y, z\} \subseteq C$ be a local edge. Remove each vertex v from $\operatorname{fit}_{E^*}(C)$ which is not locally universal for $\{y, z\}$.*

The following lemma shows that Operation 1 preserves Property 1.

▶ **Lemma 4.3** (⋆). *Let G' be a minimum solution for I, let C be a community of $\mathcal{C}$ and let $x \in \operatorname{univ}_{G'}(C)$ such that x is not locally universal for some local edge $\{y, z\} \subseteq C$. Then, G' is not fitting for E^*.*

Note that after the exhaustive application of Operation 1, for each community $C \in \mathcal{C}$ with at least one local edge, the vertices of $\operatorname{fit}_{E^*}(C)$ induce a clique with only local edges.

Next, we define a partition $\mathfrak{C}$ of the communities of $\mathcal{C}$. The idea of this partition is that in each fitting solution for E^*, all communities of the same part of the partition $\mathfrak{C}$ have the same unique center. The definition of the partition $\mathfrak{C}$ is based on the following lemma.

▶ **Lemma 4.4** (⋆). *Let C and D be distinct communities of $\mathcal{C}$ with $|C \cap D| \geq 3$ and where no vertex $v \in C \cup D$ is locally universal for $C \cap D$. Let G' be a solution such that there is no vertex $w \in C \cap D$ with $\operatorname{univ}_{G'}(C) = \operatorname{univ}_{G'}(D) = \{w\}$. Then, C and D induce a local cycle in G' that uses at least one edge which is not a local edge.*

Consider the auxiliary graph $G_{\mathfrak{C}}$ with vertex set $\mathcal{C}$ and where two distinct communities C and D are adjacent if and only if *a*) $|C \cap D| \geq 3$ and *b*) there is no locally universal vertex for $C \cap D$ in $C \cup D$. The partition $\mathfrak{C}$ consists of the connected components of $G_{\mathfrak{C}}$ and for a community $C \in \mathcal{C}$, we denote by $\mathfrak{C}(C)$ the collection of communities in the connected component of C in $G_{\mathfrak{C}}$. An example is shown in Figure 4.

By Lemma 4.4 and due to transitivity, we obtain the following.

▶ **Corollary 4.5.** *For each community $C \in \mathcal{C}$ with $|\mathfrak{C}(C)| \geq 2$ and each fitting solution G' for E^*, there is a vertex $v \in \bigcap_{\widetilde{C} \in \mathfrak{C}(C)} \widetilde{C}$ such that $\operatorname{univ}_{G'}(\widetilde{C}) = \{v\}$ for each $\widetilde{C} \in \mathfrak{C}(C)$.*

This implies that the following operation preserves Property 1.

▶ **Operation 2.** *Let $C \in \mathcal{C}$. Remove each vertex v from $\operatorname{fit}_{E^*}(C)$ if v is not contained in $\bigcap_{\widehat{C} \in \mathfrak{C}(C)} \operatorname{fit}_{E^*}(\widehat{C})$.*

Next, we define an operation for each possibility how two communities may intersect.

▶ **Operation 3.** *Let $C \in \mathcal{C}$ such that C contains no local edge. Moreover, let $D \in \mathcal{C}$ such that $|C \cap D| \geq 2$. Remove all vertices from $\operatorname{fit}_{E^*}(C)$ that are not contained in $C \cap D$.*

▶ **Operation 4.** *Let $C \in \mathcal{C}$ such that C contains at least one local edge. Moreover, let $D \notin \mathfrak{C}(C)$ be a community, such that $|C \cap D| = 2$ and $\{x, y\} \coloneqq C \cap D$ is not a local edge.*

1. *If $\operatorname{fit}_{E^*}(C) \cap \{x, y\} = \emptyset$, then remove x and y from $\operatorname{fit}_{E^*}(D)$ or*
2. *if $\operatorname{fit}_{E^*}(C) \cap \{x, y\} = \{x\}$, then set $\operatorname{fit}_{E^*}(C) \coloneqq \{x\}$.*

▶ **Operation 5.** *Let $C \in \mathcal{C}$ be a community containing at least one local edge. Moreover, let $D \notin \mathfrak{C}(C)$ such that $|C \cap D| \geq 3$. For each pair of distinct vertices x and y of $C \cap D$, where $\{x, y\}$ is not a local edge, remove x and y from $\operatorname{fit}_{E^*}(D)$.*

▶ **Lemma 4.6** (⋆). *Operation 3 preserves Property 1. Moreover, if Operation 1 is exhaustively applied, then Operation 4 and Operation 5 preserve Property 1.*

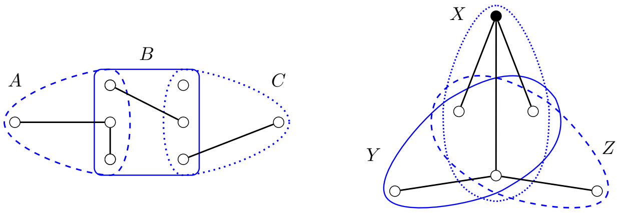


Figure 4 Examples for parts of the partition $\mathfrak{C}$. Only the local edges are shown. Note that $A, C \in \mathfrak{C}(B)$, since A and C share at least three vertices with B and no vertex of $A \cup B$ or $C \cup B$ is locally universal for $A \cap B$ or $C \cap B$, respectively. Hence, after exhaustive application of Operation 2, $\text{fit}_{E^*}(A) = \text{fit}_{E^*}(B) = \text{fit}_{E^*}(C) = \emptyset$, since A and C share no vertices. Furthermore, $Y \in \mathfrak{C}(Z)$, since no vertex of $Y \cup Z$ is locally universal for $Y \cap Z$. Note that $X \notin \mathfrak{C}(Z)$, since the black vertex of X is locally universal for $X \cap Y$ and $X \cap Z$. Observe that an exhaustive application of Operation 2 yields $\text{fit}_{E^*}(Z) \subseteq Y \cap Z$ and an exhaustive application of Operation 5 yields $\text{fit}_{E^*}(Z) \cap (X \cap Z) = \text{fit}_{E^*}(Z) \cap (Y \cap Z) = \emptyset$, since X contains at least one local edge and $X \cap Z$ contains no local edge. Hence, for both shown hypergraphs, there is no fitting solution for the given set of local edges.

Algorithm 1 Algorithm solving the problem described in Theorem 4.2.

```
Input  : $I = (G = (V, E), \mathcal{C}, \omega, \ell, b), E^* \subseteq E$
Output : A solution $G' = (V, E')$ with at most $\ell$ edges and total weight at most $b$, or
         no, if there is no minimal solution which is fitting for $E^*$
Compute the partition $\mathfrak{C}$ of $\mathcal{C}$
For each $C \in \mathcal{C}$, initialize $\text{fit}_{E^*}(C) \leftarrow \text{univ}_G(C)$ and apply Operation 1
Apply Operations 1–5 exhaustively
if $\text{fit}_{E^*}(C) = \emptyset$ for some $C \in \mathcal{C}$ then return no
$G_A \leftarrow (V, E^*)$
forall $\mathcal{L} \in \mathfrak{C}$ do
    $C \leftarrow$ some community of $\mathcal{L}$
    $V_{\mathcal{L}} \leftarrow \bigcup_{\widetilde{C} \in \mathcal{L}} \widetilde{C}$
    $y \leftarrow \arg\min_{u \in \text{fit}_{E^*}(C)} \omega(\{\{u, v\} : v \in V_{\mathcal{L}} \setminus \{u\}\} \setminus E^*)$
    add all edges of $\{\{y, v\} : v \in V_{\mathcal{L}} \setminus \{y\}\}$ to $G_A$
if $|E(G_A)| \leq \ell$ and $\omega(E(G_A)) \leq b$ then return $G_A$
return no
```

Based on these operations, we are now able to present the algorithm (see Algorithm 1) behind Theorem 4.2 working as follows: First, we apply Operations 1–5 exhaustively. Next, if there is a community $C \in \mathcal{C}$ with $\text{fit}_{E^*}(C) = \emptyset$, then we return no. This is correct, since Operations 1–5 preserve Property 1. Afterwards, we start with an auxiliary graph G_A with vertex set V and edge set E^* and we iterate over the partition $\mathfrak{C}$. Recall that since Operation 2 is exhaustively applied, for each $\mathcal{L} \in \mathfrak{C}$, $\text{fit}_{E^*}(C) = \text{fit}_{E^*}(D)$ for any two communities C and D of $\mathcal{L}$. For each $\mathcal{L} \in \mathfrak{C}$, we find a vertex $y \in \text{fit}_{E^*}(C)$ that minimizes the total weight of non-local edges required to make y the center of all communities of $\mathcal{L}$, where C is an arbitrary community of $\mathcal{L}$. Finally, we add all edges between y and each vertex of any community of $\mathcal{L}$ to G_A. After the iteration over the partition $\mathfrak{C}$ is completed, we output G_A if it contains at most ℓ edges and has total weight at most b. Otherwise, we return that there is no fitting solution for E^*. It remains to show that this greedy choice for the center vertices is correct.

▶ **Lemma 4.7.** *Algorithm 1 is correct.*

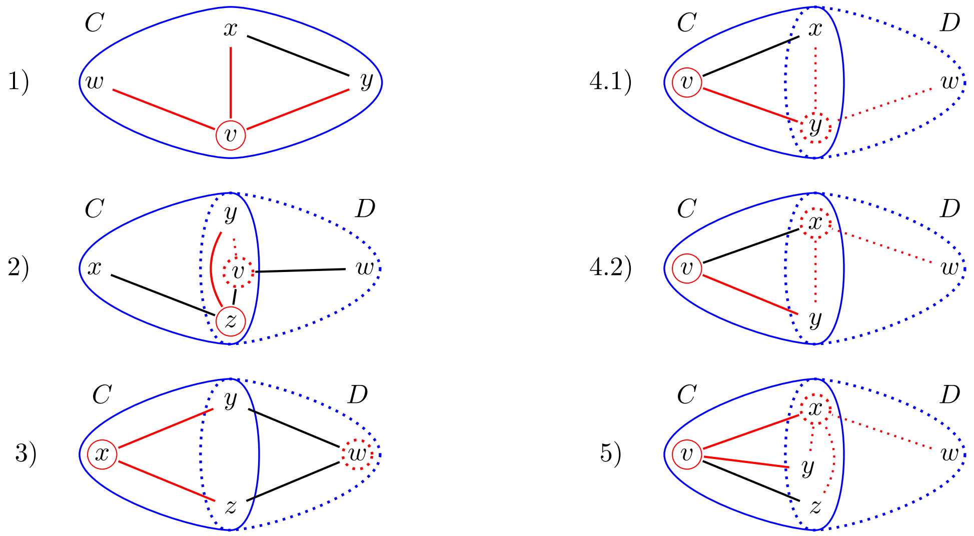


Figure 5 Examples of applications of Operations 1–5. The black edges represent the local edges, the solid (for C) or dashed (for D) red edges show the non-local edges resulting from choosing the respective center for community C or D. For example in 2), z is the center of community C and v is the center of community D, and the edges $\{z, y\}$ and $\{v, y\}$ are non-local edges in the solution. For each operation, the violation of the property of being a fitting solution is shown, if a vertex a is selected as a center of a community A where the application of the corresponding operation would remove a from $\mathrm{fit}_{E^*}(A)$. In 1), 2), 3), and 4.2), the vertex selected as center for community C is removed from $\mathrm{fit}_{E^*}(C)$ by the respective operation. For example, in 4.2), (assuming $\mathrm{fit}_{E^*}(C) \cap \{x, y\} = \{x\}$) Operation 4 removes v from $\mathrm{fit}_{E^*}(C)$, as otherwise selecting v as center of C results in the depicted non-fitting solution. In 4.1) and 5), the vertex selected as center for community D is removed from $\mathrm{fit}_{E^*}(D)$ by the respective operation. For example in 4.1), (assuming $\mathrm{fit}_{E^*}(C) \cap \{x, y\} = \emptyset$) Operation 4 removes y from $\mathrm{fit}_{E^*}(D)$, as otherwise selecting y as center of D results in the depicted non-fitting solution.

Proof. If Algorithm 1 outputs "no" in Line 4, then this is correct, since fit_{E^*} fulfills Property 1. Otherwise, let G_A denote the graph constructed by Algorithm 1 and let for each community $C \in \mathcal{C}$, $\mathrm{center}(C)$ denote the vertex y chosen to be the center of all communities of $\mathfrak{C}(C)$ in Line 9. By construction, G_A is a solution since for each community $C \in \mathcal{C}$, $\mathrm{center}(C)$ is a vertex of $\mathrm{fit}_{E^*}(C) \subseteq \mathrm{univ}_G(C)$. If G_A contains at most ℓ edges and has total weight at most b, then the algorithm correctly outputs the solution G_A which is fitting for E^*.

Thus, in the following we assume that G_A contains more than ℓ edges or has weight more than b. Assume towards a contradiction that there is a fitting solution G_F for E^* such that $\mathrm{Agree}(G_F) := \{C \in \mathcal{C} : \mathrm{center}(C) \in \mathrm{univ}_{G_F}(C)\}$ is as large as possible.

Case 1. $\mathrm{Agree}(G_F) = \mathcal{C}$**.** By construction, G_A contains all edges of E^* and only the required edges to achieve that for each community $C \in \mathcal{C}$, $\mathrm{center}(C) \in \mathrm{univ}_{G_A}(C)$. Consequently, G_A is a subgraph of G_F and thus G_F contains more than ℓ edges or has weight more than b, a contradiction.

Case 2. There is a community $C \in \mathcal{C} \setminus \mathrm{Agree}(G_F)$**.** In the following, we define a fitting solution G'_F for E^* with $\mathrm{Agree}(G'_F) \supsetneq \mathrm{Agree}(G_F)$. By definition, $\mathrm{center}(C) = \mathrm{center}(\widetilde{C})$ for each community $\widetilde{C} \in \mathfrak{C}(C)$. Let $V_C := \bigcup_{\widetilde{C} \in \mathfrak{C}(C)} \widetilde{C}$ and let $y := \mathrm{center}(C)$. Moreover, let x be an arbitrary vertex of V_C such that $x \in \mathrm{univ}_{G_F}(\widetilde{C})$ for each community $\widetilde{C} \in \mathfrak{C}(C)$. Due to Corollary 4.5 and since G_F is fitting for E^*, this vertex exists and is unique if $|\mathfrak{C}(C)| \geq 2$. Note that $C \in \mathcal{C} \setminus \mathrm{Agree}(G_F)$ implies that $x \neq y$. This also implies that C has size at least 3, and thus, each community of $\mathfrak{C}(C)$ has size at least 3. We obtain G'_F as follows: First, initialize G'_F as G_F. Second, for each community $\widetilde{C} \in \mathfrak{C}(C)$, remove all edges that are not local edges of $G_F[\widetilde{C}]$ from G'_F. Finally, for each community $\widetilde{C} \in \mathfrak{C}(C)$, add the minimum number of edges to G'_F such that $y \in \mathrm{univ}_{G'_F}(\widetilde{C})$, that is, the edges $\{\{y,v\} : v \in V_{\widetilde{C}} \setminus \{y\}\} \setminus E^*$.

First, we show that G'_F contains at most as many edges as G_F. To this end, we first observe the following.

▷ Claim 3 (⋆). For each $z \in V_C \setminus \{x, y\}$, the edge $\{x, z\}$ is a local edge if and only if $\{y, z\}$ is a local edge.

Recall that each edge which is in G'_F and not in G_F is incident with y and some vertex of $V_C \setminus \{x, y\}$. Hence, for each $z \in V_C \setminus \{x, y\}$ where the edge $\{y, z\}$ was added to obtain G'_F, the edge $\{x, z\}$ was removed to obtain G'_F. Thus, G'_F contains at most as many edges as G_F. This implies that the difference between the total weight of G'_F and the total weight of G_F is at most $\rho = \omega(\{\{y, z\} : z \in V_C \setminus \{y\}\} \setminus E^*) - \omega(\{\{x, z\} : z \in V_C \setminus \{x\}\} \setminus E^*)$. Due to Line 9, ρ is not positive. Thus, since G_F has total weight at most b, G'_F has total weight at most b.

To show that G'_F is a solution, it remains to show that each community $C \in \mathcal{C}$ has at least one center in G'_F. For this, it suffices to show that all communities outside of $\mathfrak{C}(C)$ have the same centers in G_F and G'_F, since y is a center of all communities of $\mathfrak{C}(C)$.

▷ Claim 4. For each community $D \in \mathcal{C} \setminus \mathfrak{C}(C)$, $\mathrm{univ}_{G_F}(D) = \mathrm{univ}_{G'_F}(D)$.

Proof. Due to symmetry, we only show that $\mathrm{univ}_{G_F}(D) \subseteq \mathrm{univ}_{G'_F}(D)$. Assume towards a contradiction that there is a vertex $z \in \mathrm{univ}_{G_F}(D) \setminus \mathrm{univ}_{G'_F}(D)$. Since $z \notin \mathrm{univ}_{G'_F}(D)$, there is an edge $\{z, w\}$ which is contained in G_F but not in G'_F. Moreover, $\{z, w\}$ is not a local edge, since G'_F contains all local edges. This further implies that there is a community $\widetilde{C} \in \mathfrak{C}(C)$ such that $\{z, w\} \subseteq \widetilde{C}$. Since G_F is fitting for E^*, x is one endpoint of $\{z, w\}$, as otherwise, $\widetilde{C}$ and D induce a local cycle in G_F on the vertices $\{x, z, w\}$ and the edge $\{z, w\}$ is not a local edge. Next, we distinguish the cases whether $\widetilde{C}$ contains a local edge.

Case 1. There is no local edge in $\widetilde{C}$**.** Since Operation 3 is exhaustively applied, $\{x, y\} \subseteq \mathrm{fit}_{E^*}(\widetilde{C}) \subseteq \widetilde{C} \cap D$. Hence, if $|\widetilde{C} \cap D| = 2$, then $x = z$ and $y = w$, or vice versa. Consequently, the edge $\{z, w\}$ is contained in G'_F, a contradiction. Otherwise, assume $|\widetilde{C} \cap D| \geq 3$. We show that in this case, there is no fitting solution for E^*. Since D is not in $\mathfrak{C}(C)$, there is some vertex of $\widetilde{C} \cup D$ which is locally universal for $\widetilde{C} \cap D$. Hence, $\mathrm{fit}_{E^*}(\widetilde{C}) \subseteq \widetilde{C} \cap D$, since $\widetilde{C}$ contains no local edge and Operation 3 is exhaustively applied. Moreover, since Operation 5 is exhaustively applied and there is no local edge between any two vertices of $\widetilde{C} \cap D$, $\mathrm{fit}_{E^*}(\widetilde{C}) \cap (\widetilde{C} \cap D) = \emptyset$. We conclude that $\mathrm{fit}_{E^*}(\widetilde{C}) = \emptyset$, which implies that there is no fitting solution for E^*, a contradiction to the fact that G_F is a fitting solution for E^*.

Case 2. There is some local edge in $\widetilde{C}$**.** Recall that Operation 4 and Operation 5 are applied exhaustively with respect to $\widetilde{C}$. If $x = z$ and $y = w$, or vice versa, then the edge $\{z, w\}$ is contained in G'_F, a contradiction. Otherwise, let w^* be the unique vertex of $\{z, w\} \setminus \{x\}$. Since $\{x, w^*\} = \{z, w\}$ is not a local edge, $x \in \mathrm{fit}_{E^*}(\widetilde{C})$, and Operation 1 is exhaustively applied, no vertex of $\mathrm{fit}_{E^*}(\widetilde{C})$ is locally universal for w^* and $w^* \notin \mathrm{fit}_{E^*}(\widetilde{C})$. Hence, if $|\widetilde{C} \cap D| = 2$, then since Operation 4 is exhaustively applied, $\mathrm{fit}_{E^*}(\widetilde{C})$ has size at most one, a contradiction. Otherwise, if $|\widetilde{C} \cap D| \geq 3$, then since Operation 5 is exhaustively applied $x \notin \mathrm{fit}_{E^*}(D)$ and $w^* \notin \mathrm{fit}_{E^*}(D)$. Consequently, $z \notin \mathrm{fit}_{E^*}(D)$, a contradiction. ◁

Since G_F is a solution, for each community $D \in \mathcal{C} \setminus \mathfrak{C}(C)$, Claim 4 implies that $\text{univ}_{G'_F}(D) = \text{univ}_{G_F}(D)$ is nonempty. Hence, G'_F is a solution. Moreover, since G_F is a minimum solution, Claim 3 implies that G'_F is a minimum solution.

Next, we show that G'_F is fitting for E^*. To show that G'_F is a fitting solution for E^*, it remains to show that each local cycle of G'_F uses only edges of E^*.

▷ Claim 5 (⋆). Each local cycle of G'_F uses only edges of E^*.

Finally, we show that $\text{Agree}(G'_F)$ is a proper superset of $\text{Agree}(G_F)$. By construction, $\mathfrak{C}(C) \subseteq \text{Agree}(G'_F)$, and due to Claim 4, for each community $D \in \mathcal{C} \setminus \mathfrak{C}(C)$, $\text{univ}_{G'_F}(D) = \text{univ}_{G_F}(D)$. Hence, $\text{Agree}(G_F) \subseteq \text{Agree}(G'_F)$. Moreover, since $C \notin \text{Agree}(G'_F) \setminus \text{Agree}(G_F)$ we obtain that $\text{Agree}(G'_F)$ is a proper superset of $\text{Agree}(G_F)$. Altogether, G'_F is a fitting solution for E^* with $\text{Agree}(G'_F) \supsetneq \text{Agree}(G_F)$. This contradicts our choice of G_F.

Hence, if G_A contains more than ℓ edges or has weight more than b, then the algorithm correctly outputs that there is no solution which is fitting for E^*. ◀

Proof of Theorem 4.2. Clearly, the partition $\mathfrak{C}$ of $\mathcal{C}$ and also the initialization of fit_{E^*} in Lines 1 and 2 can be computed in polynomial time. Note that Operations 1–5 can be exhaustively applied in polynomial time by iterating over all local edges and all pairs of communities, since for each community $C \in \mathcal{C}$, $\text{fit}_{E^*}(C)$ initially has size at most $|C| < n$ and each application of any operation may only remove elements from $\text{fit}_{E^*}(C)$. Hence, Lines 3–5 can be performed in polynomial time. Afterwards, Lines 6–10 can be performed in polynomial time since for each partite set of $\mathfrak{C}$ we compute the vertex y with minimal cost such that y serves as the center of all communities in this partite set. Finally, the check whether the solution has at most ℓ edges and weight at most b can be done in polynomial time. Thus, Algorithm 1 runs in polynomial time. ◀

Finding the correct edge set E^*. To solve Stars NWS, the main algorithmic difficulty now lies in finding an edge set E^* that contains all edges of local cycles of any optimal solution of I. Hence, to prove Theorem 4.1, it remains to show that such an edge set can be found in $m^{4t} \cdot \text{poly}(n + c)$ time, if it exists.

▶ **Lemma 4.8** (⋆). *If I is a yes-instance of Stars NWS, then for every optimal solution $G' = (V, E')$, there is an edge set $E^* \subseteq E'$ of size at most $4t$ such that the edge set of each local cycle of G' is a subset of E^*.*

Proof of Theorem 4.1. The algorithm works as follows: iterate over all possible edge sets E^* of size at most $4t$ and apply the algorithm behind Theorem 4.2. If I is a yes-instance, then for some edge set E^*, Theorem 4.2 yields an optimal solution for I with at most ℓ edges and weight at most b. Since there are $\mathcal{O}(m^{4t})$ edges sets of size at most $4t$, the algorithm achieves the stated running time. ◀

The concrete algorithm for $t = 0$. Recall that Theorem 4.1 affirmatively answers the question by Korach and Stern [30] who asked whether there is a polynomial-time algorithm for finding an optimal solution for Stars NWS with $t = 0$. For this case, the concrete algorithm is much simpler since most of the described operations are never applicable. This is due to the fact that for $t = 0$, no local edges exist and Operations 1, 4, and 5 require at least one local edge to be applicable. In the following, we give an intuitive description of the concrete much simpler algorithm for $t = 0$.

First, we set $E^* = \emptyset$ and initialize $\text{fit}_{E^*}(C) := \text{univ}_G(C)$ for each community $C \in \mathcal{C}$. Afterwards, we again compute the partition $\mathfrak{C}$ of the communities of $\mathcal{C}$. Recall that this is done by defining an auxiliary graph $G_{\mathfrak{C}}$ with vertex set $\mathcal{C}$ where two distinct communities C and D are adjacent if and only C and D have an intersection of size at least 3. Note that in the original definition one had to also check that no vertex is locally optimal for the intersection. This now always holds since there are no local edges. The partition $\mathfrak{C}$ then consists of the connected components of $G_{\mathfrak{C}}$.

Second, we exhaustively apply Operations 2 and 3. Operation 2 ensures that all communities of the same part of $\mathfrak{C}$ will have the same potential centers according to fit_{E^*}. Moreover, Operation 3 ensures that if two communities C and D have an intersection of size at least 2, then the potential centers of both communities will be within $C \cap D$ according to fit_{E^*}.

After exhaustive application of these two operations, the remaining lines of Algorithm 1 are executed, that is, if $\text{fit}_{E^*}(C) = \emptyset$ for at least one community $C \in \mathcal{C}$, we correctly output that the instance under consideration is a no-instance of Stars NWS. Otherwise, we greedily select for each part $\mathcal{L}$ of the partition $\mathfrak{C}$ a vertex y as center for all communities $\mathcal{L}$, such that y is a potential center of each community of $\mathcal{L}$ and such that the cost of selecting y as center of all these communities is minimum under all such potential centers. Finally, if these choices of center vertices result in more than ℓ edges or total weight more than b, we correctly output that the input instance is a no-instance of Stars NWS. Otherwise, the chosen center vertices induce a solution with $t = 0$.

4.2 Connectivity NWS

Korach and Stern presented an $\mathcal{O}(c^4 n^2)$-time algorithm for Connectivity NWS where G is a clique and $t = 0$ [29] which was improved by Klemz et al. [28] who provided an $\mathcal{O}(m \cdot (c + \log(n)))$-time algorithm for Connectivity NWS with $t = 0$. Guttmann-Beck et al. [21] presented a similar algorithm for Unweighted Connectivity NWS with $t = 0$.

Next, we show that the positive result for $t = 0$ cannot be lifted to $t = 1$; in this case Connectivity NWS is NP-hard. We obtain our result by reducing from the NP-hard Hamiltonian Cycle-problem [1, 18], which asks for a given graph $G = (V, E)$ if there is a *Hamiltonian cycle* in G, that is, a cycle containing each vertex of G exactly once.

▶ **Theorem 4.9** ($\star$)**.** *Let Π be a graph class on which Hamiltonian Cycle is* NP*-hard, then Unweighted Connectivity NWS is* NP*-complete on Π even if $t = 1$.*

Proof. Let $I := (V, E)$ be an instance of Hamiltonian Cycle containing at least three vertices. We obtain an equivalent instance $I' := (G = (V, E), \mathcal{C}, \ell)$ of Unweighted Connectivity NWS as follows: We start with an empty set $\mathcal{C}$ and add for each vertex $v \in V$ a community $C_v := V \setminus \{v\}$ to $\mathcal{C}$. Finally, we set $\ell := |V|$. Note that $t = \ell - n + x$, where x is the number of connected components of the graph. Thus, $t = n - n + 1 = 1$.

The detailed correctness proof is deferred to the full version. ◀

▶ **Corollary 4.10.** *Unweighted Connectivity NWS is NP-complete even if $t = 1$ on subcubic bipartite planar graphs.*

5 Stars NWS Parameterized by the Number of Communities

Unweighted Connectivity NWS is NP-hard even for $c = 7$ [17, Proposition 3]. In contrast, Stars NWS admits an XP-algorithm for c with running time $n^{\mathcal{O}(c)}$: For each community C, test each of the at most $|C| \leq n$ potential center vertices. Then, for each potential solution check whether it consists of at most ℓ edges of total weight at most b. For

Stars NWS, we show that it is unlikely that this brute-force algorithm can be improved, by showing W[1]-hardness. For Unweighted Stars NWS, we obtain an FPT-algorithm for c.

▶ **Theorem 5.1** (⋆). *Stars NWS is* W[1]*-hard when parameterized by c even if G is a clique and each edge weight is* 1 *or* 2*.*

Proof. We provide a parameter-preserving reduction from the W[1]-hard Regular Multicolored Clique problem [11]. The input consists of an r-regular graph G, an integer κ, and a partition $(V_1, \dots V_\kappa)$ of $V(G)$. The question is whether there exists a clique of size κ containing exactly one vertex of each partite set V_i.

We construct an equivalent instance $(G', \mathcal{C}, \omega, \ell, b)$ of Stars NWS as follows. The vertex set of $V(G')$ consists of a copy of $V(G)$ and κ additional vertex sets S_i, $i \in [\kappa]$, each of size $n(G)^3$. We make G' a clique by adding all edges between vertices of $V(G')$. To complete the construction, we specify the communities and edge weights. First, for each color class $i \in [1, \kappa]$, we add a community $C_i := V(G) \cup S_i$. Afterwards, we define the edge weights: For each edge $\{a, b\} \in E(G')$ such that $\{a, b\} \in E(G)$, we set $\omega(\{a, b\}) := 2$, for each edge $\{a, b\}$ with $a \in S_i$ and $b \notin V_i$, we set $\omega(\{a, b\}) := 2$, for each edge $\{a, b\}$ with $a \in S_i$ and $b \in S_j$, we set $\omega(\{a, b\}) := 2$, and for each remaining edge $\{a, b\} \in E(G')$ we set $\omega(\{a, b\}) := 1$. Finally, we set $\ell := \kappa \cdot (n(G)^3 + n(G) - 1) - \binom{\kappa}{2}$ and $b := \kappa \cdot (n(G)^3 + n(G) - 1 + r) - 2\binom{\kappa}{2}$. Note that $c = \kappa$, it thus remains to show the equivalence of the two instances. The detailed correctness proof is deferred to the full version. ◀

▶ **Theorem 5.2** (⋆). *Unweighted Stars NWS is solvable in $\mathcal{O}(4^{c^2} \cdot (n + m) + n^2 \cdot c)$ time.*

To complete the parameterized complexity picture, we show that a polynomial kernel for c is unlikely.

▶ **Theorem 5.3** (⋆). *Unweighted Stars NWS parameterized by c does not admit a polynomial kernel unless NP ⊆ coNP/poly.*

6 Conclusion

Presumably the most interesting open question is whether Stars NWS parameterized by t admits an FPT-algorithm. In Theorem 4.2 we showed that Stars NWS can be solved in polynomial time if for some optimal solution the edge set of all local cycles is known. Thus, to obtain an FPT-algorithm, it is sufficient to find such an edge set in FPT-time. Also, it is open whether Unweighted Connectivity NWS can be solved in polynomial time when t is constant and the input graph is a clique. In other words, it is open whether a minimum-edge hypergraph support can be found in polynomial time when it has a constant feedback edge number.

It is also interesting to close the gap between the running time lower bound of $2^{\Omega(c)} \cdot \mathrm{poly}(|I|)$ (see Proposition 2.5) and the upper bound of $2^{\mathcal{O}(c^2)} \cdot \mathrm{poly}(|I|)$ (see Theorem 5.2) for Unweighted Stars NWS. Also, we may ask the following: are there properties Π such that Π-NWS is NP-hard but can be solved in $2^{\mathcal{O}(n)} \cdot \mathrm{poly}(n + c)$ time?

References

1 Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the Hamiltonian cycle problem for bipartite graphs. *Journal of Information Processing*, 3(2):73–76, 1980.

2 Dana Angluin, James Aspnes, and Lev Reyzin. Network construction with subgraph connectivity constraints. *Journal of Combinatorial Optimization*, 29(2):418–432, 2015.

3 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

4 Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, and Arnaud Sallaberry. Path-based supports for hypergraphs. *Journal of Discrete Algorithms*, 14:248–261, 2012.

5 Kevin Buchin, Marc J. van Kreveld, Henk Meijer, Bettina Speckmann, and Kevin Verbeek. On planar supports for hypergraphs. *Journal of Graph Algorithms and Applications*, 15(4):533–549, 2011.

6 Chandra Chekuri, Thapanapong Rukkanchanunt, and Chao Xu. On element-connectivity preserving graph simplification. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA '15)*, volume 9294 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2015.

7 Jiehua Chen, Christian Komusiewicz, Rolf Niedermeier, Manuel Sorge, Ondrej Suchy, and Mathias Weller. Polynomial-time data reduction for the subset interconnection design problem. *SIAM Journal on Discrete Mathematics*, 29(1):1–25, 2015.

8 Gregory V. Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing scalable overlays for pub-sub with many topics. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '07)*, pages 109–118. ACM, 2007.

9 Nathann Cohen, Frédéric Havet, Dorian Mazauric, Ignasi Sau, and Rémi Watrigant. Complexity dichotomies for the minimum F-overlay problem. *Journal of Discrete Algorithms*, 52-53:133–142, 2018.

10 Vincent Conitzer, Jonathan Derryberry, and Tuomas Sandholm. Combinatorial auctions with structured item graphs. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI '04)*, pages 212–218. AAAI Press / The MIT Press, 2004.

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

12 Rodney G. Downey and Michael Ralph Fellows. *Fundamentals of Parameterized Complexity.* Springer Science & Business Media, 2013.

13 Ding-Zhu Du and Zevi Miller. Matroids and subset interconnection design. *SIAM Journal on Discrete Mathematics*, 1(4):416–424, 1988.

14 Pierre Duchet. Propriete de Helly et problemes de representation. *Colloque International CNRS, Problemes Conbinatoires et Theorie du Graphs*, 260:117–118, 1978.

15 Hongbing Fan, Christian Hundt, Yu-Liang Wu, and Jason Ernst. Algorithms and implementation for interconnection graph problem. In *Proceedings of the Second International Conference on Combinatorial Optimization and Applications (COCOA '08)*, volume 5165 of *Lecture Notes in Computer Science*, pages 201–210. Springer, 2008.

16 Claude Flament. Hypergraphes arborés. *Discrete Mathematics*, 21(3):223–227, 1978.

17 Till Fluschnik and Leon Kellerhals. Placing green bridges optimally, with a multivariate analysis. *Theory of Computing Systems*, 2024. To appear.

18 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

19 Aristides Gionis, Polina Rozenshtein, Nikolaj Tatti, and Evimaria Terzi. Community-aware network sparsification. In *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM '17)*, pages 426–434. SIAM, 2017.

20 Nili Guttmann-Beck, Roni Rozen, and Michal Stern. Vertices removal for feasibility of clustered spanning trees. *Discrete Applied Mathematics*, 296:68–84, 2021.

21 Nili Guttmann-Beck, Zeev Sorek, and Michal Stern. Clustered spanning tree - conditions for feasibility. *Discrete Mathematics & Theoretical Computer Science*, 21(1), 2019.

22 Maike Herkenrath, Till Fluschnik, Francesco Grothe, and Leon Kellerhals. Placing green bridges optimally, with habitats inducing cycles. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, (IJCAI '22)*, pages 3825–3831. ijcai.org, 2022.

23 Emanuel Herrendorf. On the complexity of community-aware network sparsification. Master's thesis, Philipps-Universität Marburg, 2022. URL: `https://www.fmi.uni-jena.de/fmi_femedia/24200/master-emanuel-pdf.pdf`.

24 Jun Hosoda, Juraj Hromkovic, Taisuke Izumi, Hirotaka Ono, Monika Steinová, and Koichi Wada. On the approximability and hardness of minimum topic connected overlay and its special instances. *Theoretical Computer Science*, 429:144–154, 2012.

25 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

26 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

27 David S. Johnson and Henry O. Pollak. Hypergraph planarity and the complexity of drawing venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987.

28 Boris Klemz, Tamara Mchedlidze, and Martin Nöllenburg. Minimum tree supports for hypergraphs and low-concurrency Euler diagrams. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT '14)*, volume 8503 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2014.

29 Ephraim Korach and Michal Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 98(1-3):385–414, 2003.

30 Ephraim Korach and Michal Stern. The complete optimal stars-clustering-tree problem. *Discrete Applied Mathematics*, 156(4):444–450, 2008.

31 Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, and Dorothea Wagner. Structure-preserving sparsification of social networks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM '15)*, pages 448–454. ACM, 2015.

32 Natsu Nakajima, Morihiro Hayashida, Jesper Jansson, Osamu Maruyama, and Tatsuya Akutsu. Determining the minimum number of protein-protein interactions required to support known protein complexes. *PloS one*, 13(4):e0195545, 2018.

33 Peter J Slater. A characterization of soft hypergraphs. *Canadian Mathematical Bulletin*, 21(3):335–337, 1978.

34 Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

35 Fang Zhou, Sébastien Mahler, and Hannu Toivonen. Network simplification with minimal loss of connectivity. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM '10)*, pages 659–668. IEEE Computer Society, 2010.

$\mathcal{H}$-Clique-Width and a Hereditary Analogue of Product Structure

Petr Hliněný
Masaryk University, Brno, Czech Republic

Jan Jedelský
Masaryk University, Brno, Czech Republic

Abstract

We introduce a novel generalization of the notion of clique-width which aims to bridge the gap between classical hereditary width measures and the recently introduced graph product structure theory. Bounding the new $\mathcal{H}$-clique-width, in the special case of $\mathcal{H}$ being the class of paths, is equivalent to admitting a hereditary (i.e., induced) product structure of a path times a graph of bounded clique-width. Furthermore, every graph admitting the usual (non-induced) product structure of a path times a graph of bounded tree-width, has bounded $\mathcal{H}$-clique-width and, as a consequence, it admits the usual product structure in an induced way. We prove further basic properties of $\mathcal{H}$-clique-width in general.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases product structure, hereditary class, clique-width, twin-width

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.61

Related Version *Full Preprint*: `https://arxiv.org/abs/2403.16789` [17]

Acknowledgements We acknowledge helpful discussions with Jakub Gajarský on the questions and problems posed in Section 5, and especially on the question of $\mathcal{H}$-clique-width possibly being complement-closed.

1 Introduction

A prominent structural result by Dujmović, Joret, Micek, Morin, Ueckerdt and Wood [11], known as the *Planar product structure theorem*, claims that every planar graph can be found as a subgraph in the strong product ($\boxtimes$) of a path and a graph of small tree-width. We refer to Section 2 and Theorem 2.3 for the definitions and details.

The original motivation for this rather recent Product structure theorem was to bound the queue number of planar graphs, but the theorem has quickly found interesting applications and follow-up results, among which we may mention [1,9,10,12,13,23]. Namely, the product structure theory has been used to study non-repetitive colourings [10], to design short labelling schemes [1,9], or to bound the twin-width of planar graphs [4,16,19].

The basic goal of the product structure theory can be seen in studying graph classes which admit such product structure, that is, they can be constructed as subgraphs of the strong product of a path and a graph of small tree-width. Within this setting, there are two major restrictions; first that the containment (subgraph) relation is not induced, and second that this kind of a superstructure can exist only for sparse graph classes.

We aim to give a different perspective on the product structure (Definition 2.1) addressing both mentioned issues, that is, getting graphs which admit the traditional product structure as *induced* subgraphs in such strong product, and allowing also dense graphs to occur.

Our alternative view is two-sided and is closely related to another classical structural notion in graph theory – the clique-width measure. On one hand, any graph admitting the traditional product structure can be obtained as an induced subgraph of the strong product of a path and a suitable graph of bounded clique-width and even bounded tree-width (Theorem 4.6). On the other hand, a graph G admits the induced product structure with bounded clique-width (of the relevant factor), if and only if G has bounded $\mathcal{H}$-clique-width where $\mathcal{H}$ is the class of reflexive paths (Theorem 4.1). This dual nature of our view is another promising enhancement. Moreover, we believe this view can contribute to finding potential algorithmic applications of product structure theory.

The wide scope of our definition suggests to study $\mathcal{H}$-clique-width for other graph families $\mathcal{H}$ in addition to paths, too. For instance, in relation to the aforementioned product-structure works, one may consider $\mathcal{H}$ to be the class of the graphs $P_n \boxtimes K_k$ or of $P_n \boxtimes P_m$.

We study and characterize relations of $\mathcal{H}$-clique-width to ordinary clique-width (Theorem 3.1), to local clique-width (Theorem 3.4), and in parts to twin-width (Corollary 4.3). The full potential of this new concept when $\mathcal{H}$ is a family of specific graphs other than paths is yet to be explored, especially in the case of $\mathcal{H}$ formed by suitable dense graphs. We conclude with a number of open questions related to the new concept (Section 5).

2 Preliminaries

We consider finite simple graphs, i.e., graphs without parallel edges or loops, but in one specific context (Definition 2.1) we allow graphs with optional (self-)loops, thereafter called loop graphs. Precisely, a *loop graph* is a multigraph allowing loops (at most one per vertex), but not allowing parallel edges. In the context of loop graphs, we specially call a graph G a *reflexive (loop) graph* if every vertex of G has a loop. We naturally use terms *reflexive path*, *reflexive clique*, and *reflexive independent set* to denote ordinary paths, cliques, and independent sets, respectively, with loops added to all vertices. We write $G_1 \subseteq_i G_2$ to say that G_1 is an induced subgraph of G_2. Note that if $G_1 \subseteq_i G_2$ for loop graphs, then possible loops of G_2 on vertices of G_1 are also inherited.

A graph G is a *matching* if G is simple and all vertex degrees in G are 1. A graph $G \subseteq K_{n,n}$ is an *antimatching* if G is obtained from $K_{n,n}$ by removing a matching of n edges. A graph G is a *half-graph* if G is a bipartite simple graph with the bipartition $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_n\}$, such that $u_i v_j \in E(G)$ if and only if $i \leq j$.

Width measures. As a traditional structural decomposition, a *tree decomposition* of a graph G is a tuple $(T, \mathcal{X})$ where T is a tree, and $\mathcal{X} = \{X_t : t \in V(T)\}$ where $X_t \subseteq V(G)$ is a collection of *bags* which satisfy the following: (i) $\bigcup_{t \in V(T)} X_t = V(G)$, (ii) for every vertex $v \in V(G)$, the set of the nodes $t \in V(T)$ such that $v \in X_t$ forms a subtree in T, and (iii) for every edge $uv \in E(G)$, there exists $t \in V(T)$ such that $\{u, v\} \subseteq X_t$. The *tree-width* of G is the minimum of $\max_{t \in V(T)} |X_t| - 1$ over all tree decompositions of G.

Our work is closely related to another measure, which is a "dense counterpart" of tree-width. The *clique-width* of a graph G is the minimum integer ℓ such that G (irrespective of labelling) is the value of an algebraic *ℓ-expression* defined by the following operations:

- create a new vertex of label (colour) i for some $i \in \{1, \ldots, \ell\}$;
- take the disjoint union of two labelled graphs;
- for $1 \leq i \neq j \leq \ell$, add all (missing) edges between a vertex of label i and a vertex of label j;
- for $1 \leq i \neq j \leq \ell$, recolour each vertex of label i to have label j.

In the same direction, let the *local clique-width* of a graph G be the integer function λ defined as follows; for an integer distance $r \geq 1$, $\lambda(r)$ is the maximum clique-width of the r-neighbourhood of a vertex v in G, over all $v \in V(G)$. We say that a graph class $\mathcal{G}$ is of *bounded local clique-width* if there exists an integer function upper-bounding the local clique-width of every member of $\mathcal{G}$. For instance, the class of grids is of bounded local clique-width, but of unbounded clique-width.

The last measure we mention, twin-width, was introduced a few years ago by Bonnet et al. in [3]. A *trigraph* is a simple graph G in which some edges are marked as *red*, and with respect to the red edges only, we naturally speak about *red neighbours* and *red degree* in G. For a pair of (possibly not adjacent) vertices $x_1, x_2 \in V(G)$, we define a contraction of the pair x_1, x_2 as the operation creating a trigraph G' which is the same as G except that x_1, x_2 are replaced with a new vertex x_0 (said to *stem from* x_1, x_2) such that:

- the (full) neighbourhood of x_0 in G' (i.e., including the red neighbours), denoted by $N_{G'}(x_0)$, equals the union of the neighbourhoods $N_G(x_1)$ of x_1 and $N_G(x_2)$ of x_2 in G except x_1, x_2 themselves, that is, $N_{G'}(x_0) = (N_G(x_1) \cup N_G(x_2)) \setminus \{x_1, x_2\}$, and
- the red neighbours of x_0, denoted here by $N^r_{G'}(x_0)$, inherit all red neighbours of x_1 and of x_2 and add those in $N_G(x_1)\Delta N_G(x_2)$, that is, $N^r_{G'}(x_0) = \big(N^r_G(x_1) \cup N^r_G(x_2) \cup (N_G(x_1)\Delta N_G(x_2))\big) \setminus \{x_1, x_2\}$, where Δ denotes the symmetric set difference.

A *contraction sequence* of a trigraph G is a sequence of successive contractions turning G into a single vertex, and its width d is the maximum red degree of any vertex in any trigraph of the sequence. The *twin-width* of a trigraph G is the minimum width over all possible contraction sequences of G. The twin-width of a multigraph or a loop graph is defined as the twin-width of its simplification.

Introducing $\mathcal{H}$-clique-width. Our main contribution builds on the following new concept.

▶ **Definition 2.1** ($\mathcal{H}$-clique-width)**.** *Let $\mathcal{H}$ be a family of loop graphs, and $\ell > 0$ be an integer. Consider* labels *of the form (i, v) where $i \in \{1, \ldots, \ell\}$ and $v \in V(H)$ for some (fixed) $H \in \mathcal{H}$.*

a) *For $H \in \mathcal{H}$, let an* (H, ℓ)-expression *be an algebraic expression using the following four operations on vertex-labelled graphs:*
- *creating a new vertex with single label (i, v) for some $i \in \{1, \ldots, \ell\}$ and $v \in V(H)$;*
- *taking the disjoint union of two labelled graphs;*
- *for $1 \le i \ne j \le \ell$,* adding edges between i and j, *which means to add all edges between the vertices of label (i, v) and the vertices of label (j, w) over all pairs $(v, w) \in V(H)^2$ such that $vw \in E(H)$ (including the case of a single vertex $v = w$ with a loop, which will often be assumed to exist for the graphs H); and*
- *for $1 \le i \ne j \le \ell$,* recolouring i to j, *which means to relabel all vertices with label (i, v) where $v \in V(H)$ to label (j, v).*

b) *The* $\mathcal{H}$-clique-width $\mathcal{H}\text{-cw}(G)$ *of a simple graph G is defined as the smallest integer ℓ such that (some labelling of) G is the value of an (H, ℓ)-expression for some $H \in \mathcal{H}$. If it is not possible to build G this way, then let $\mathcal{H}\text{-cw}(G) = \infty$.*

Given an (H, ℓ)-expression of value (a labelled graph) G, we use the following terminology; the graph H is the parameter *of the expression, and when referring to a label (i, v) of $x \in V(G)$, the integer i is the* colour *and v the* parameter vertex *of x.*

Observe that, throughout an (H, ℓ)-expression φ valued G, the colours of vertices of G may arbitrarily change by the recolouring operations, but the parameter vertex of every $x \in V(G)$ stays the same (is uniquely determined for x) in φ.

It is obvious that $\mathcal{H}$-clique-width (similarly to ordinary clique-width) is monotone under taking induced subgraphs. On the other hand, it is not apriori clear whether $\mathcal{H}$-clique-width is (at least functionally) closed under taking the complement of a graph; we will address this interesting issue in the concluding section. Another remark concerns the family $\mathcal{H}$ which should be generally treated as an infinite class of (finite) loop graphs, due to $\mathcal{H}$-clique-width being asymptotically the same as ordinary clique-width in the case of finite $\mathcal{H}$ – see Theorem 3.1.

To further briefly illustrate Definition 2.1, we add few more easy observations:

▷ Claim 2.2.

a) If $\mathcal{H} = \{K_1\}$, then $\mathcal{H}$-cw$(G) < \infty$ if and only if G has no edges. If $\mathcal{H} = \{K_1^\circ\}$, where K_1° stands for a single vertex with a loop, then $\mathcal{H}$-cw$(G) = \mathrm{cw}(G) < \infty$.

b) If $\mathcal{H}$ contains a loop graph with at least one loop, then $\mathcal{H}$-cw$(G) \leq \mathrm{cw}(G)$.

c) If $\mathcal{H} = \{H\}$, then $\mathcal{H}$-cw$(H_1) \leq 2$ holds for every simple graph H_1 obtained from an induced subgraph of H by removing loops.

d) If $\mathcal{H} = \{H\}$ and H is disconnected, then for every connected simple graph G we have $\mathcal{H}$-cw$(G) = \{H_0\}$-cw(G) for some connected component H_0 of H.

e) If $\mathcal{H} = \{K_2\}$ (no loops), then $\mathcal{H}$-cw$(G) < \infty$ if and only if G is a simple bipartite graph. More generally, for any $\mathcal{H}$, we have $\mathcal{H}$-cw$(G) < \infty$ for a simple graph G, if and only if G has a homomorphism into some $H \in \mathcal{H}$.

f) For any $k \geq 3$ and $\mathcal{H} = \{K_k\}$, it is NP-hard to decide whether $\mathcal{H}$-cw$(G) < \infty$.

g) Let $\mathcal{H}$ be a family containing arbitrarily long reflexive paths. If G is any square grid, then $\mathcal{H}$-cw$(G) \leq 5$ (while cw(G) is unbounded in such case).

Proof.

a) There is no edge in K_1, and so Definition 2.1 cannot create an edge in G. On the other hand, (K_1°, ℓ)-expressions in Definition 2.1 exactly coincide with traditional ℓ-expressions of clique-width (replacing every label i with (i, v) where $\{v\} = V(K_1^\circ)$).

b) We pick $v \in V(H)$ for $H \in \mathcal{H}$ such that v has a loop in H. Then, in an ordinary cw(G)-expression for G, we replace every label i with (i, v) to get an $(H, \mathrm{cw}(G))$-expression for G, similarly to part a).

c) We simply make an $(H, 2)$-expression for H_1 as follows; in an arbitrary order $V(H_1) = \{v_1, \ldots, v_n\}$ of the vertices, for $k = 1, \ldots, n$, we add a new vertex labelled $(2, v_k)$, add edges between 1 and 2, and recolour 2 to 1. This creates exactly the non-loop edges of H_1.

d) This follows from the facts that the recolouring operation of Definition 2.1 does not allow to change the initially assigned parameter vertex of H, and hence every edge of G created within an (H, ℓ)-expression has a preimage edge in H. So, an expression creating connected G may only use parameter vertices of a connected component of H.

e) Considering the previous argument turned around, every edge created within an (H, ℓ)-expression has a unique homomorphic image in H (possibly a loop). In the opposite direction, for a homomorphism $h : G \to H \in \mathcal{H}$, we make an $(H, |V(G)|)$-expression starting with the disjoint union of vertices labelled $(i_x, h(x))$ for all $x \in V(G)$ where $i_x \neq i_y$ for $x \neq y$, and then simply add the edges of G one by one using the colours (i.e., i_x).

f) By e), we have $\mathcal{H}$-cw$(G) < \infty$ if and only if G is k-colourable.

g) Let G be an $a \times b$ grid, i.e., $|V(G)| = ab$. We take $H \in \mathcal{H}$ such that $|V(H)| \geq b$, choose a consecutive subpath on $\{v_1, \ldots, v_b\} \subseteq V(H)$, and make an $(H, 5)$-expression valued G as follows. As in c), we define an $(H, 3)$-(sub)expression creating a “vertical” copy P_1 of the path on b vertices, but now using three colours such that the resulting labels of P_1 are with alternating colours 2 and 3, precisely as $(2, v_1)$, $(3, v_2)$, $(2, v_3)$, $(3, v_4), \ldots$. We likewise create a copy P_2 of the same path with alternating colours in labels $(4, v_1)$, $(5, v_2)$, $(4, v_3)$, $(5, v_4), \ldots$. Then we make a disjoint union and add edges between colours $2, 4$ and between $3, 5$ – this creates precisely the “horizontal” edges between the labels $(2, v_i)$ and $(4, v_i)$, and between $(3, v_{i+1})$ and $(5, v_{i+1})$, for $i = 1, 3, \ldots$. In a subsequent round, we recolour colours $2, 3$ to 1 (this concerns only P_1), and continue the same process with adding a path P_3 with alternating colours again 2 and 3, and adding the "horizontal" edges. After $a - 1$ rounds, we build the desired $a \times b$ square grid G. ◁

Product structure theory. The *strong product* $G_1 \boxtimes G_2$ of two simple graphs is the graph G on the vertex set $V(G) := V(G_1) \times V(G_2)$ such that, for any $[u, u']$, $[v, v'] \in V(G)$, we have $\{[u, u'], [v, v']\} \in E(G)$ if and only if ($uv \in E(G_1)$ and $u'v' \in E(G_2)$) or ($u = v$ and $u'v' \in E(G_2)$) or ($uv \in E(G_1)$ and $u' = v'$).

For an illustration, the strong product $P \boxtimes Q$ of two paths P, Q is the square grid with diagonals. It may be interesting to observe that, in the context of loops graphs, if both G_1 and G_2 are reflexive, then the definition of the strong product $G_1 \boxtimes G_2$ could be shortened as "$uv \in E(G_1)$ and $u'v' \in E(G_2)$", and the result would be the same except that all vertices would have loops.

Origins of graph product structure theory go back to the mentioned seminal paper [11]:

▶ **Theorem 2.3** ([11], improved in [23]). *Every planar graph is a subgraph of the strong product $P \boxtimes M$ where P is a path and M is a planar graph of tree-width at most 6.*

There exist alternative refined formulations of Theorem 2.3, such as using the strong product $P \boxtimes K_3 \boxtimes M$ where M is now of tree-width at most 3 which is of importance in some applications (such as in refining the upper bound on the queue number of planar graphs).

Our main goal is to refine, using Definition 2.1, the statement of Theorem 2.3 with the induced subgraph relation in Theorem 4.6; admittedly, at the cost of worse absolute constants.

3 Properties of $\mathcal{H}$-Clique-Width

We first characterize the asymptotic difference between the ordinary clique-width and the $\mathcal{H}$-clique-width for families of loop graphs $\mathcal{H}$.

We recall the concept of neighbourhood diversity by Lampis [20]. Two vertices x, y of a simple graph G are of the same *neighbourhood type* if and only if they have the same set of neighbours in $V(G) \setminus \{x, y\}$. We shall use an adjusted version of this concept, suitable for our loop graphs; Two vertices x, y of a simple loop graph G are of the same *total neighbourhood type*, if and only if they have the same set of neighbours in $V(G)$ when x counts as a neighbour of x if there is a loop on x (and likewise with y). A loop graph G is of *total neighbourhood diversity* at most d if $V(G)$ can be partitioned into d parts such that every pair in the same part have the same total neighbourhood type.

The slight, but very important in our context, difference of these two notions in presence of loops can be observed, e.g., on: loopless cliques K_n (neighbourhood diversity 1 and total neighbourhood diversity n) vs. reflexive cliques K_n° (total neighbourhood diversity 1), or loopless stars $K_{1,n}$ (both neighbourhood diversities equal 2) vs. reflexive stars $K_{1,n}^\circ$ (total neighbourhood diversity n).

A loop graph class $\mathcal{G}$ is of *component-bounded* total neighbourhood diversity if there exists an integer d such that each connected component of every graph of $\mathcal{G}$ is of total neighbourhood diversity at most d.

▶ **Theorem 3.1.** *Let $\mathcal{H}$ be a family of loop graphs. There exists a function f such that,* $\mathrm{cw}(G) \leq f(\mathcal{H}\text{-}\mathrm{cw}(G))$ *holds for all simple graphs G, if and only if $\mathcal{H}$ is of component-bounded total neighbourhood diversity.*

Proof. In the "$\Leftarrow$" direction, we may assume G is connected (we will later take the maximum over connected components). By Claim 2.2 d), $\mathcal{H}$-cw$(G) = \{H_0\}$-cw$(G) = \ell$ for a connected component H_0 of some $H \in \mathcal{H}$. The total neighbourhood diversity of H_0 is at most some constant d, by the theorem assumption. Then, in an (H_0, ℓ)-expression for G, we may equivalently replace the parameter vertices of H_0 by d new colours, giving a $d\ell$-expression for G. So, $\mathrm{cw}(G) \leq d \cdot \mathcal{H}$-cw$(G)$.

A proof of the "$\Rightarrow$" direction is based on the following natural technical claim:

▷ Claim 3.2 (Ding et al. [7, Corollary 2.4]). For every k there exists m such that the following holds. If F is a bipartite connected simple graph with the bipartition $V(F) = A \cup B$, $|A| \geq m$ and the vertices of A have pairwise different neighbourhood types (in B), then F contains an induced subgraph isomorphic to one of the following graphs on $2k$ vertices: a matching, an antimatching, or a half-graph.

Having Claim 3.2 at hand, we continue as follows. Assume that $\mathcal{H}$ is *not* of component-bounded total neighbourhood diversity. Let $H \in \mathcal{H}$ (or a component thereof) be a connected loop graph of total neighbourhood diversity $\geq c_1$, and $C \subseteq V(H)$ be vertices representing these c_1 total neighbourhood types. By Ramsey theorem, for sufficiently large c_1 we find a subset $C_1 \subseteq C$, $|C_1| = 2c_2 - 1$, such that C_1 induces a clique or an independent set in G, and then we can select $C_2 \subseteq C_1$, $|C_2| = c_2$ such that either all vertices of C_2 have loops, or none has. We have got one of the two possibilities:

- C_2 is a reflexive independent set or a loopless clique in G.
- Or, all vertices of C_2 have the same total neighbourhood type in C_2 (empty or full C_2), and so they have pairwise different neighbourhood types in $D := V(G) \setminus C_2$. Consequently, we may apply Claim 3.2 to the bipartite subgraph "between" C_2 and D.

Regarding the second point, in more detail, we say that a bipartite graph F_1 with a fixed bipartition $V(F_1) = A_1 \cup B_1$ is a *bi-induced subgraph* of a graph H, if $F_1 \subseteq H$ such that every edge of H with one end in A_1 and the other end in B_1 belongs to F_1. Claim 3.2 hence implies that one of the three claimed subgraphs is bi-induced in H.

Altogether, for every k and sufficiently large c_1 depending on k, we have connected $H \in \mathcal{H}$ containing one of the five said substructures; an induced reflexive independent set or an induced loopless clique on k vertices, or a bi-induced matching, a bi-induced antimatching, or a bi-induced half-graph on $2k$ vertices. In each of these five cases, we can construct a "grid-like" graph of bounded $\mathcal{H}$-clique-width whose ordinary clique-width grows linearly with k. This is provided by subsequent Lemma 3.3, in which one can easily check that its assumptions cover all five cases of $H \in \mathcal{H}$ listed in this proof. ◀

▶ **Lemma 3.3.** *Let $k \geq 3$ be an integer, and H_1 be a loop graph satisfying the following:*

- *H_1 is connected.*
- *There exist sets $A, B \subseteq V(H_1)$, $|A| = |B| = k$, such that either $A = B$, or $A \cap B = \emptyset$.*
- *We can write $A = \{u_1, \dots, u_k\}$ and $B = \{u'_1, \dots, u'_k\}$ such that, for some of the following three conditions on integers $C(i,j) \in \{$'$i < j$','$i = j$','$i \neq j$'$\}$ we have; for all $(i,j) \in \{1, \dots, k\}^2$, $\{u_i, u'_j\} \in E(H_1)$ if and only if $C(i,j)$ is false. (Note that, if $A = B$, we assume $u_i = u'_i$ and deal also with loops.)*

Then, there exists a constant ℓ_0 independent of k such that the class of graphs of $\{H_1\}$-clique-width at most ℓ_0 has ordinary clique-width $\Omega(k)$.

Proof. We construct, via an $(H_1, \mathcal{O}(1))$-expression, a graph G_k of ordinary clique-width $\Omega(k)$ as follows.

Similarly as in Claim 2.2 c), we create a loopless copy G_1' of the graph H_1, such that every vertex $x \in V(G_1')$ which is a copy of a vertex $v \in B$ has the label with colour 2 and parameter vertex v and, if $A \neq B$, every vertex $x \in V(G_1')$ which is a copy of $w \in A$ has the label with colour 1 and parameter vertex w. Vertices of $V(G_i')$ that are not copies of $A \cup B$ have labels with colour 0 and the respective parameter vertex from $V(H_1) \setminus (A \cup B)$.

We set $G_1 := G_1'$, and for $a = 2, \ldots, k$ we do:

- We, likewise, create a loopless copy G_a' of H_1, now with colour 3 in the labels of the copy of A in $V(G_a')$ and, if $A \neq B$, with colour 4 in the labels of the copy of B in $V(G_a')$. The labels of $V(G_a')$ besides the copies of $A \cup B$ are again with colour 0.
- Then we make a disjoint union $G_a := G_{a-1} \mathbin{\dot\cup} G_a'$, and add edges between colours 2 and 3.
- If $A = B$, we recolour 2 to 1 and 3 to 2. If $A \neq B$, we recolour 2 and 3 to 1 and 4 to 2.

Altogether, the graph G_k has $k \cdot |V(H_1)|$ vertices, k disjoint copies G_a' of H_1, and every copy G_a' has k vertices which are, in a well-defined way – cf. condition $C(i,j)$, adjacent to corresponding k vertices of the subsequent copy G_{a+1}' (if $a < k$). There are no other edges in G_k. For clarity (and in resemblance to Theorem 4.1), we imagine the copy G_a' of H_1 as "column a" of G_k, and the set of the copies of u_i and u_i' of H_1 as "row i" of G_k. Possible remaining vertices of G_a' (those of colour 0 in their label) are not part of any row, as they do not participate in inter-column edges of G_k. Observe that, if $A \neq B$, the adjacency pattern occurring between columns a and $a+1$ is exactly the same as the edges between B and A in H_1, and so the same as the "mirrored" adjacency pattern between the copies of A and of B within column a (or $a+1$).

Now, assume we have an (ordinary) ℓ-expression φ valued G_k for some integer ℓ. We apply an argument which is folklore in this area. There must exist a subexpression φ_1 of φ making a subset of vertices $X \subseteq V(G_k)$ (it is irrelevant which of the edges of $G_k[X]$ this φ_1 makes), such that $\frac{1}{3}|V(G_k)| \leq |X| \leq \frac{2}{3}|V(G_k)|$. Let $\bar{X} = V(G_k) \setminus X$.

Consider any $1 \leq a < k$; then the columns a and $a+1$ differ with respect to X in at most 3ℓ rows ($\leq \ell$ if $A = B$); meaning that in $\leq 3\ell$ rows i we have a situation that a vertex of row i in one of the columns a or $a+1$ belongs to X, and a vertex of row i in the other column belongs to $\bar{X}$. This follows since we have at most ℓ different colours in φ_1 which can be used to further distinguish different adjacencies, as given by the condition $C(i,j)$ of the lemma, between the columns a and $a+1$, or within each one of the columns a or $a+1$ if $A \neq B$.

Likewise, at most ℓ columns are such that they intersect both X and $\bar{X}$. This follows similarly since every column is a copy of connected H_1, and so it needs in φ_1 a special colour for its (at least one) "private" edge from X to $\bar{X}$. The two latter conditions together are in a clear contradiction with $\frac{1}{3}|V(G_k)| \leq |X| \leq \frac{2}{3}|V(G_k)|$ if $\ell \in o(k)$. ◀

Secondly, there is an interesting relation to established concepts in the case of parameter families $\mathcal{H}$ of bounded degrees.

▶ **Theorem 3.4.** *Let $\mathcal{H}$ be a family of loop graphs of maximum degree Δ. Then the class of graphs of $\mathcal{H}$-clique-width at most ℓ is of bounded local clique-width in terms of Δ and ℓ.*

Proof. Let $H \in \mathcal{H}$ and G be a graph that is a value of an (H,ℓ)-expression φ. Choose $x \in V(G)$, and assume a vertex $y \in V(G)$ at distance at most r from x in G. Let $v, w \in V(H)$ be the parameter vertices in φ of x and y, respectively. As argued in Claim 2.2 e), there is a homomorphism $G \to H$ taking a path between x and y into a walk between v and w in H, and so the distance from v to w in H is at most r. Since $\Delta(H) \leq \Delta$, the r-neighbourhood of v in H has at most $(\Delta+1)^r$ vertices, and hence φ restricted to the r-neighbourhood of x in G uses only at most $(\Delta+1)^r$ parameter vertices which can be replaced in φ by unique colours.

This way we obtain an (ordinary) $\ell \cdot (\Delta+1)^r$-expression whose value is the r-neighbourhood of x in G. We can thus set $f(r) := \ell \cdot (\Delta+1)^r$ (independently of $H \in \mathcal{H}$ and G) to certify bounded local clique-width of every G such that $\mathcal{H}$-cw$(G) \leq \ell$. ◀

A similar structural relation of $\mathcal{H}$-clique-width to the parameter twin-width is stated later in Corollary 4.3, as a consequence of a product-structure-like characterization.

From Theorem 3.4 we, for instance, immediately get tractability of FO model checking, which is FPT for all classes of bounded local clique-width – this well-known fact follows by a combination of the ideas of Frick and Grohe [15] and of Dawar, Grohe and Kreutzer [5]:

▶ **Corollary 3.5.** *For every family $\mathcal{H}$ of loop graphs, the FO model checking problem of a graph G is in FPT when parameterized by the formula, the maximum degree of $\mathcal{H}$ and the $\mathcal{H}$-clique-width of G.* ◀

Furthermore, it may be interesting to ask to which extent Theorem 3.4 can be reversed. This cannot be done straightforwardly since there are families $\mathcal{H}$ of unbounded degrees, such that classes of bounded $\mathcal{H}$-clique-width not only have bounded local clique-width, but even bounded ordinary clique-width. One example is $\mathcal{H}_1$ the class of all reflexive cliques by Theorem 3.1. On the other hand, e.g., for $\mathcal{H}_2$ being the class of all reflexive stars, there are graphs whose $\mathcal{H}_2$-clique-width is bounded by a constant, and they contain arbitrarily large induced grids and a universal vertex adjacent to everything (a construction similar to Claim 2.2 g)). Such graph hence have unbounded local clique-width.

4 Approaching Induced Product Structure

In this section we restrict our attention to families $\mathcal{H}$ formed by *reflexive* loop graphs (i.e., all vertices have loops in the graphs of $\mathcal{H}$), which makes most natural sense with respect to the strong-product structure studied.

▶ **Theorem 4.1.** *Let $\mathcal{H}$ be a family of reflexive loop graphs, and $\mathcal{H}'$ be the family of simple graphs obtained from the graphs of $\mathcal{H}$ by removing all loops. For every integer $\ell \geq 2$ the following holds. A simple graph G is of $\mathcal{H}$-clique-width at most ℓ, if and only if G is isomorphic to an induced subgraph of the strong product $H' \boxtimes M$ where $H' \in \mathcal{H}'$ and M is a simple graph of clique-width at most ℓ.*

Proof. In the "$\Leftarrow$" direction, it is enough to show that $\mathcal{H}$-cw$(G) \leq \ell$ for $G := H' \boxtimes M$. Let φ be an ℓ-expression of the graph M, and let H° be obtained from H' by adding loops to all vertices. We are going to transform φ into an (H°, ℓ)-expression as follows. First, for each $x \in V(M)$ we independently construct a copy H'_x of H', using only $2 \leq \ell$ colours by Claim 2.2 c). That is, the parameter vertex of every $v_x \in V(H'_x)$ is the preimage $v \in V(H')$ of v_x. Then, at every moment the expression φ introduces a new vertex $y \in V(M)$ of colour i, we take (substitute) the copy H'_y and recolour it to i. The remaining operations (union, recolouring, and adding edges) stay in place in φ, but are now applied according to Definition 2.1.

We claim that the value G of the resulting transformed (H°, ℓ)-expression σ is $H' \boxtimes M$. Indeed, the vertex set is $V(G) = V(H') \times V(M)$, and for each $m \in V(M)$ the subgraph induced on $V(H') \times \{m\}$ is a copy of H'. For any $[v_1, m_1], [v_2, m_2] \in V(G)$ and $m_1 \neq m_2$; if $\{[v_1, m_1], [v_2, m_2]\} \in E(G)$, then $v_1v_2 \in E(H^\circ)$ by Definition 2.1, and $m_1m_2 \in E(M)$ by the definition of σ. Hence $\{[v_1, m_1], [v_2, m_2]\} \in E(H' \boxtimes M)$. Conversely, if $\{[v_1, m_1], [v_2, m_2]\} \in E(H' \boxtimes M)$, then, by the definition of $\boxtimes$, $v_1v_2 \in E(H')$ or $v_1 = v_2$, meaning $v_1v_2 \in E(H^\circ)$, and $m_1m_2 \in E(M)$. So, the edge $\{[v_1, m_1], [v_2, m_2]\}$ has been created by σ.

In the "⇒" direction, let σ be an (H°,ℓ)-expression valued G, for some $H^\circ \in \mathcal{H}$. Let $H' \in \mathcal{H}'$ be the simple graph of H°. We are going to construct an ℓ-expression φ valued M on the vertex $V(M) = V(G)$, such that $G \subseteq_i H' \boxtimes M$. The expression φ simply discards parameter vertices (cf. Definition 2.1) from the labels in σ. Hence, we clearly get $M \supseteq G$. To prove that $G \subseteq_i H' \boxtimes M$, consider any vertices $x \neq y \in V(G)$ labelled (i,v) and (j,w) by σ (for here, indefinite i and j are irrelevant, and v and w are uniquely determined by σ). We claim that the vertices x and y as of G can be represented by $[v,x]$ and $[w,y]$ of the product $H' \boxtimes M$. If $xy \in E(G)$, then $xy \in E(M)$ by previous $M \supseteq G$, and $vw \in E(H^\circ)$ by Definition 2.1. Consequently, $\{[v,x],[w,y]\} \in E(H' \boxtimes M)$ by $\boxtimes$. On the other hand, if $\{[v,x],[w,y]\} \in E(H' \boxtimes M)$, then $vw \in E(H')$ or $v = w$, and so $vw \in E(H^\circ)$. Moreover, $xy \in E(M)$ since $x \neq y$, and so $xy \in E(G)$ since the (original) (H°,ℓ)-expression σ creates the edge xy by Definition 2.1. ◀

Theorem 4.1 can be used also to bound the twin-width of graphs of bounded $\mathcal{H}$-clique-width. To show this, we first prove the following ad-hoc upper bound.

▶ **Proposition 4.2.** *Let P be a reflexive path and G a simple graph. Then the twin-width of G is at most $5 \cdot (\{P\}\text{-cw}(G)) - 2$. Consequently, denoting by $\mathcal{P}^\circ$ the class of all reflexive paths, the twin-width of any simple graph G is at most $5 \cdot (\mathcal{P}^\circ\text{-cw}(G)) - 2$.*

Proof. Let G be the value of a (P,ℓ)-expression φ, where $\ell = \{P\}\text{-cw}(G)$. When constructing a contraction sequence for G, we proceed recursively (bottom-up) along the expression tree of φ; processing only the union and recolouring nodes, and at each node contracting together all vertices of the same label.

Consider a situation at a node with a subexpression φ_0 of φ, where $X_0 \subseteq V(G)$ is the vertex set generated by φ_0, and let $x^0_{(i,v)}$ denote the vertex resulting from the contractions of all vertices of X_0 that are of label (i,v) by φ_0. The core observation is that every vertex of $V(G) \setminus X_0$ has the same adjacency to all vertices forming $x^0_{(i,v)}$ by Definition 2.1, and so the only possible red neighbours of $x^0_{(i,v)}$ in a contraction of G are those that stem from X_0.

The only possible neighbours of $x^0_{(i,v)}$ in the described contraction of the induced subgraph $G[X_0]$ are $x^0_{(j,w)}$ where $j \in \{1,\dots,\ell\}$ and $vw \in E(P)$ – altogether at most $3\ell - 1$ choices of potential red neighbours of $x^0_{(i,v)}$ in the contraction of $G[X_0]$. If a recolouring operation i to j is encountered after the node of φ_0, we simply contract each former $x^0_{(i,v)}$ with $x^0_{(j,v)}$ over all $v \in X_0$, not increasing the previous bound on the red degree.

Consider now a union node making $X_2 := X_0 \dot{\cup} X_1$, where X_1 has been generated by a sibling subexpression φ_1 of φ, and let $x^1_{(i,v)}$ analogously denote the vertices resulting from contractions of X_1. Let the vertices of P be $V(P) = (v_1,\dots,v_a)$ in the natural order along the path. For $k = 1,\dots,a$, and subsequently for $i = 1,\dots,\ell$, we make $x^2_{(i,v_k)}$ by contracting $x^0_{(i,v_k)}$ with $x^1_{(i,v_k)}$. Considering the corresponding successive contractions of the induced subgraph $G[X_2]$, the only possible red neighbours of $x^2_{(i,v_k)}$ are the ℓ vertices $x^2_{(i',v_{k-1})}$, the up to $2(\ell-1)$ vertices $x^2_{(j,v_k)}$ for $j < i$, or $x^0_{(j,v_k)}$, $x^1_{(j,v_k)}$ for $j > i$, and the 2ℓ vertices $x^2_{(i'',v_{k+1})}$, $x^2_{(i'',v_{k+1})}$. The maximum possible encountered red degree is thus $\ell + 2(\ell-1) + 2\ell = 5\ell - 2$. ◀

▶ **Corollary 4.3.** *Let $\mathcal{H}$ be a family of reflexive loop graphs of maximum degree Δ and twin-width at most t. Then the twin-width of any simple graph G is at most $\mathcal{O}(t + \Delta \cdot \mathcal{H}\text{-cw}(G))$.*

Proof. By Theorem 4.1, we have $G \subseteq_i H' \boxtimes M$, where H' is of maximum degree at most Δ and twin-width at most t and M is of clique-width at most $\ell := \mathcal{H}\text{-cw}(G)$. Since twin-width is monotone under taking induced subgraphs, it is enough to bound it for the graph $H' \boxtimes M$.

By Proposition 4.2 applied to P being a single reflexive vertex, and Claim 2.2 a), M is of twin-width at most $k := 5\ell - 2$. Then, by Bonnet et al. [2] (bounding twin-width of a strong product), $H' \boxtimes M$ is of twin-width at most $\max\{t + \Delta,\, k(\Delta + 1) + 2\Delta\} = \mathcal{O}(t + \Delta\ell)$. ◀

Notice that, for constant Δ, the bound $\mathcal{O}(t+\Delta\cdot\mathcal{H}\text{-cw}(G))$ in Corollary 4.3 is asymptotically best possible; a linear dependence on t (the maximum twin-width of $\mathcal{H}$) is necessary due to Claim 2.2 c), and a linear dependence on $\mathcal{H}$-cw(G) is, on the other hand, required already by the subcase of ordinary clique-width. It is not clear whether the linear dependence on Δ in the bound of Corollary 4.3 is really necessary, however, the next construction shows that the bound has to depend on Δ, the maximum degree of $\mathcal{H}$:

▶ **Proposition 4.4.** **(*)** *Let H_n denote the half-graph (cf. Section 2) on $2n$ vertices. Then the twin-width of the graphs $H_n \boxtimes H_n$ grows with n. Furthermore, the class $\{H_n \boxtimes H_n : n \in \mathbb{N}_+\}$ is monadically independent.*

Since Proposition 4.4 was not part of the reviewed MFCS submission, we leave its proof to the full preprint [17].

If $\mathcal{H}$ is the family of reflexive half-graphs (which is of unbouded maximum degree), then $\mathcal{H}$-cw$(H_n \boxtimes H_n) \leq$ cw$(H_n) \leq 3$ using Theorem 4.1 and a trivial 3-expression for H_n, and likewise the twin-width of $\mathcal{H}$ is constant. So, from Proposition 4.4 we get that the bound in Corollary 4.3 must grow with $n = \Delta(H_n)$, too.

4.1 Relations to the traditional product structure

In regard of the Planar product structure theorem, as introduced in Section 2, we are especially interested in $\mathcal{H}$-clique-width for $\mathcal{H} = \mathcal{P}^\circ$ where $\mathcal{P}^\circ$ is the *class of reflexive paths.* We get the following as another immediate consequence of Theorem 4.1:

▶ **Corollary 4.5.** *For every integer $\ell \geq 2$ the following holds. A graph G is isomorphic to an induced subgraph of the strong product $P \boxtimes M$ where P is a path and M is a simple graph of clique-width at most ℓ, if and only if $\mathcal{P}^\circ$-cw$(G) \leq \ell$.* ◀

There is, however, a more direct connection between our concept and the original Planar product structure theorem, which constitutes the main new contribution of the paper:

▶ **Theorem 4.6.** *Assume that a graph G is a subgraph (not necessarily induced) of the strong product $G \subseteq P \boxtimes M$ where P is a path and M is a simple graph of tree-width at most k. Then $\mathcal{P}^\circ$-cw$(G) \leq 6(k+1)\cdot 8^{k+1}$. Moreover, there exists a graph M_1 of tree-width at most $6(k+1)\cdot 8^{k+1}$ such that G is isomorphic to an* induced *subgraph of the strong product $P \boxtimes M_1$.*

Proof. We start with proving the first part of the statement, that $\mathcal{P}^\circ$-cw$(G) \leq 6(k+1)\cdot 8^{k+1}$. Although, we remark that we could as well jump straight into a proof of the second part, the product $P \boxtimes M_1$, and then refer to Theorem 4.1 to conclude with a bound (albeit weaker) on $\mathcal{P}^\circ$-cw(G). We believe that the presented approach to the proof is more accessible for the readers.

We assume a rooted tree decomposition $(T, \mathcal{X})$ of width k of the graph M, such that every node of T has at most two children. For a node $t \in V(T)$, let $X_t^+ \subseteq V(M)$ denote the union of X_s where s ranges over t and all descendants of t. Let $p(t)$ denote the parent node of t in T, and let $Y_t = X_t^+ \setminus X_{p(t)}$ denote the vertices of M which occur *only* in the bags of t and its descendants. For the root r of T, let specially $Y_r = X_r^+ = V(M)$. Observe that all neighbours of a vertex $m \in Y_t$ in $V(M) \setminus Y_t$ must belong to the set $X_t \setminus Y_t$, by the interpolation property of a tree decomposition. Let $q : V(M) \to \{0, 1, \dots, k\}$ be a function such that q is injective on each of the bags X_t over $t \in V(T)$ – such q is easily constructed along the tree T in the root-to-leaves order (in fact, q can be seen as a monotone cop search strategy on the decomposition of M).

Analogously to the treatment in the proof of Theorem 4.1, we refer to the vertices of $G \subseteq P \boxtimes M$ as to the pairs $[p, m]$ where $p \in V(P)$ and $m \in V(M)$ in the natural correspondence. When constructing an expression for the graph G, we follow on a high level the tree T; at a node $t \in V(T)$, we will construct precisely the subgraph G^t of G induced on the vertex set $(V(P) \times Y_t) \cap V(G)$. By the previous, all neighbours of $V(G^t)$ in the rest of G belong to the set $W_t := \big(V(P) \times (X_t \setminus Y_t)\big) \cap V(G)$ where $|X_t \setminus Y_t| \le |X_t| \le k + 1$. It will thus be enough to encode in the colour of each $x \in V(G^t)$ information about which vertices of W_t are actual neighbours of x in G and, moreover, the colours used can be "recycled modulo 3" along the path P. This way we will prove that $\mathcal{P}^\circ$-cw(G) is bounded in terms of k; more precisely, that $\{P^\circ\}$-cw$(G) \le 6(k + 1) \cdot 8^{k+1}$ for P° being the reflexive closure of P.

Let the vertices of P be $V(P) = (p_1, \dots, p_n)$ in the natural order along the path, and let $t_m \in V(T)$ for $m \in V(M)$ denote the node closest to the root such that $m \in X_{t_m}$ (so, $m \in Y_{t_m}$). In more detail, for each $m \in V(M)$ we create, in a trivial way, a subexpression for the subgraph G_m of G induced by $\big(V(P) \times \{m\}\big) \cap V(G)$ (which is a copy of a subpath of P), such that the labels in G_m are as follows.

For each vertex $x \in V(G_m)$ where $x = [p_i, m]$, we give x the label (c_x, p_i) such that the colour of x is a tuple $c_x = (0, i \bmod 3,\ b_0, b_1, \dots, b_k,\ d)$ satisfying the following:

- i is the index of p_i, and $b_j \in \{0, 1\}^3$ and $d \in \{0, \dots, k\}$ are prescribed below;
- for every $j \in \{0, 1, \dots, k\} \setminus q(X_{t_m})$ we let $b_j = (0, 0, 0)$;
- for every $j = q(m')$ where $m' \in X_{t_m}$ (recall that q is injective on each bag, and so m' is unique such), we define $b_j = (b^1, b^2, b^3)$ where
  - $b^1 = 1$ if and only if $\{x, [p_{i-1}, m']\} \in E(G)$,
  - $b^2 = 1$ if and only if $\{x, [p_i, m']\} \in E(G)$, and
  - $b^3 = 1$ if and only if $\{x, [p_{i+1}, m']\} \in E(G)$;
- we set $d = q(m'')$ where $m'' \in V(M)$ is a neighbour of m in M such that all neighbours of m belong to $Y_{t_{m''}}$ (informally, m'' is any topmost w.r.t. T neighbour of m in M).

For an informal explanation in relation the above "sketch of encoding", the colour c_x in the label of x covers all desired information about the neighbours of x in the set W_{t_m} in G, for which adjacencies will created later in the coming expression for G. And again on an informal level, the purpose of the component d of the colour c_x is to encode the moment at which all edges of m in M are already created going bottom-up along the tree T.

Let $\Gamma_k := \{0, 1\} \times \{0, 1, 2\} \times \{0, 1\}^{3 \cdot (k+1)} \times \{0, 1, \dots, k\}$ be our set of colours (where, as above, we have $c_x \in \Gamma_k$ for every x, and "a half" of the colour space – with the first component equal to 1, remains unused so far), and let $\ell = |\Gamma_k| = 2 \cdot 3 \cdot 2^{3(k+1)} \cdot (k + 1) = 6(k + 1) \cdot 8^{k+1}$. We construct a (P°, ℓ)-expression $\varphi = \varphi^r$ valued G recursively, making subexpressions φ^t valued G^t along the nodes $t \in V(T)$, as follows:

I. For a node $t \in V(T)$ (including leaves), in the leaf-to-root tree order, we start with an empty expression φ_0 and $G^t_0 = \emptyset$ if t is a leaf. If t has one child s, then we take the expression φ_0 already constructed at s, and $G^t_0 = G^s$. If t has two children s, s', then we let φ_0 be the union operation over the expressions constructed at s and s', and $G^t_0 = G^s \dot\cup G^{s'}$. Note that, in the latter case, M has no edges between the sets Y_s and $Y_{s'}$ by the interpolation property, and so there are no edges between the (disjoint) subgraphs G^s and $G^{s'}$ in G.

II. Subsequently, for $Y'_t := Y_t \cap X_t$ (informally, Y'_t are the vertices of M whose last bag in T is right at t) we choose an arbitrary order $Y'_t = (m_1, \dots, m_a)$, $a = |Y'_t|$, of these vertices – possibly $a = 0$ if $Y'_t = \emptyset$. For $i = 1, \dots, a$, we repeat the following:

a) We start the expression φ_i by making a union of previous φ_{i-1} (if nonempty) and of the above subexpression constructing G_{m_i}.

b) Now we add in φ_i all required edges between G_{m_i} and $\big(G^t_0 \cup G_{m_1} \cup \dots \cup G_{m_{i-1}}\big)$. Using information stored in the labels of φ_{i-1} and the labels of vertices of G_{m_i}, this is a routine task as follows. For simplicity, we write $*$ for an arbitrary value.

- Let $j = q(m_i)$. For $(\alpha, \beta) \in \{(0,2), (1,0), (2,1)\}$, we add edges between each colour $(1, \alpha,\ *, \dots, b_j, \dots, *,\ *)$ where $b_j = (1, *, *)$, and each colour $(0, \beta,\ *, \dots, *,\ *)$. (Note that only vertices of G_{m_i} may currently hold colours starting with 0.)
- Similarly, for $(\alpha, \beta) \in \{(0,0), (1,1), (2,2)\}$, we add edges between each colour $(1, \alpha,\ *, \dots, b_j, \dots, *,\ *)$ where $b_j = (*, 1, *)$, and each colour $(0, \beta,\ *, \dots, *,\ *)$.
- And again, for $(\alpha, \beta) \in \{(0,1), (1,2), (2,0)\}$, we add edges between each colour $(1, \alpha,\ *, \dots, b_j, \dots, *,\ *)$ where $b_j = (*, *, 1)$, and each colour $(0, \beta,\ *, \dots, *,\ *)$.

c) Then we recolour every colour $c = (0, \beta,\ b_0, \dots, b_k,\ d)$ in the previous, where $\beta \in \{0,1,2\}$ and $b_0, \dots, b_k,\ d$ are arbitrary, to colour $c' = (1, \beta,\ b_0, \dots, b_k,\ d)$. This finishes the expression φ_i constructing a subgraph on $V(G^t_0 \cup G_{m_1} \cup \dots \cup G_{m_i})$.

III. Continuing on the expression φ_a from the previous point, we for all $i \in \{1, \dots, a\}$ do the following. We recolour every colour $c = \big(1, \beta,\ b_0, \dots, b_k,\ q(m_i)\big)$ in φ_a, where $\beta \in \{0,1,2\}$ and $b_0, \dots, b_k$ are arbitrary, to colour $c' = \big(1, \beta,\ (0,0,0)^{k+1},\ q(m_i)\big)$. (The purpose is to prevent creation of further edges from the recoloured vertices which got finished.) This finishes the sought expression φ^t with intended value G^t at the node t.

Now, the constructed (P°, ℓ)-expression $\varphi = \varphi^r$ clearly creates (precisely) all vertices and (at least) all edges of G, and uses at most $\ell = 6(k+1) \cdot 8^{k+1}$ colours. The proof will be finished once we prove that no other edges than those of G have been created by φ in G^r.

There are three points in verification of the last task.

- First, colouring in the process of construction of φ ensures that no additional edges are created within each of the graphs G_m above, and no edges are ever created between G_m and $G_{m'}$ if $mm' \notin E(M)$. The only operation adding edges in φ, besides the subexpressions making each G_m, is as defined in item IIb) above, and so it always adds only edges from $V(G_{m_i})$ to the rest of the current subgraph.
- Second, the operations in IIb) indeed add precisely those edges between G_{m_i} and $\big(G^t_0 \cup G_{m_1} \cup \dots \cup G_{m_{i-1}}\big)$ which exist in G, thanks to our definition of the colours c_x.
- And third (which relates to both previous points), the "hash" function $q : V(M) \to \{0, 1, \dots, k\}$ used in the construction of our colours indeed unambiguously identifies neighbours we want to make adjacent as in G thanks to assumed injectivity of q on each bag of $(T, \mathcal{X})$ and the recolouring performed in item III.

Regarding the second part of the Theorem, the graph M_1, we cannot directly employ Theorem 4.1 since that would give us only a factor (of the strong product) of bounded clique-width, but containing unbounded bipartite cliques in the worst case. We instead provide an ad-hoc construction of the desired factor M_1 which is closely related to fine details of the (P°, ℓ)-expression φ^r of G described above.

Let $\Gamma'_k := \{0,1\} \times \{0,1,2\} \times \{0,1\}^{3\cdot(k+1)}$, i.e., we have got $\Gamma'_k \times \{0, 1, \dots, k\} = \Gamma_k$ as above. Recall also that $Y'_t := Y_t \cap X_t$ for $t \in V(T)$ denotes the vertices of the graph M whose last bag in T (going bottom-up) is right at t, and that, when defining the expression φ of G, we have ordered the members of Y'_t where $a = |Y'_t|$ as $Y'_t = (m_1, \dots, m_a)$.

We define the sought graph M_1 such that $V(M_1) := V(M) \times \Gamma'_k$, and $E(M_1) \subseteq F$ where $F = \{\{(m,c),(m',c')\} : m = m' \vee mm' \in E(M),\ c,c' \in \Gamma'_k\}$. This setting clearly implies that tree-width of M_1 is going to be at most $(k+1) \cdot |\Gamma'_k| < 6(k+1) \cdot 8^{k+1}$, regardless of the detailed definition of its edges. We finish the definition of M_1 as follows:

i. For each $m \in V(M)$ and $\iota \in \{0,1\}$, we have $\{(m,c),(m,c')\} \in E(M_1)$ if and only if $c = (\iota, *,\ *, \ldots, *)$ and $c' = (\iota, *,\ *, \ldots, *) \neq c$.

ii. For each $m \neq m' \in V(M)$ such that $mm' \in E(M)$, up to symmetry, we either have $m \in Y'_t$ and $m' \in Y'_u$ where u is closer to the root of T than t, or $m, m' \in Y'_t$ and m precedes m' in the order $Y'_t = (m_1, \ldots, m_a)$ mentioned above. We define $\{(m,c),(m',c')\} \in E(M_1)$ if and only if, for $j = q(m')$ and some $\alpha \in \{0,1,2\}$, one of the following holds:
- $c = (*, \alpha,\ *, \ldots, b_j, \ldots, *)$, $c' = (*, (\alpha+2) \bmod 3,\ *, \ldots, *)$ and $b_j = (1,*,*)$, or
- $c = (*, \alpha,\ *, \ldots, b_j, \ldots, *)$, $c' = (*, \alpha,\ *, \ldots, *)$ and $b_j = (*,1,*)$, or
- $c = (*, \alpha,\ *, \ldots, b_j, \ldots, *)$, $c' = (*, (\alpha+1) \bmod 3,\ *, \ldots, *)$ and $b_j = (*,*,1)$.

iii. No other edges exist in M_1.

It remains to identify an isomorphism of $G \subseteq P \boxtimes M$ to an induced subgraph of the product $P \boxtimes M_1$. To each vertex $x \in V(G)$ such that $x = [p_i, m]$ in $P \boxtimes M$, we assign $x \mapsto [p_i, (m, c_i)]$ in $P \boxtimes M_1$ where $c_i = (a_i,\ i \bmod 3,\ b_0, \ldots, b_k)$ is determined in the following:

(I) The first component a_i of c_i is defined inductively by i (for each fixed m) as follows; it is $a_1 = 0$, and for each $i > 1$ we let $a_i = 1 - a_{i-1}$ if there is $y = [p_{i-1}, m] \in V(G)$ such that $xy \notin E(G)$, and $a_i = a_{i-1}$ otherwise.

(II) For each $j \in \{0,1,\ldots,k\}$ where $j = q(m')$ for some $m' \in X_{t_m}$ (and recall that there is at most one such m' for j since q is injective on each bag), the component b_j is determined as $b_j = (b^1, b^2, b^3)$ where, for $k = 1,2,3,$ $b^k = 1$ if and only if $\{x, [p_{i+k-2}, m']\} \in E(G)$.

(III) If undetermined by the previous point, b_j may be chosen arbitrarily.

Let G' be the induced subgraph of $P \boxtimes M_1$ determined by the previous assignment $\mapsto$; our remaining task is to prove that $\mapsto$ is an isomorphism of G to G'. By $\boxtimes$ and the definition of M_1, we know that edges of G and of G' are of the form $e = \{[p_i, m], [p_j, m']\}$ and $e' = \{[p_i, (m,*)], [p_j, (m',*)]\}$, respectively, where $j \in \{i-1, i, i+1\}$ and $mm' \in E(M)$ or $m = m'$.

In the case of $m = m'$, we get $e \in E(G) \iff e' \in E(G')$ already by the definition of the component a_i (a_j) in the point (I). For $mm' \in E(M)$, we get the same straightforwardly from the definition of the edge set of M_1, precisely the point ii. above, and from the (matching) point (II) of the definition of $\mapsto$. ◀

▶ Remark 4.7. Theorem 4.6 has a natural and straightforward extension to graphs $P \in \mathcal{H}$ where $\mathcal{H}$ is any graph class of bounded maximum degree (instead of the class of paths). We skip details due to their additional technical difficulty.

5 Concluding Remarks

The primary focus of our paper is an introduction of a new concept of potential interest, and as such it naturally brings many questions and open problems, (some of) which we briefly survey in this last section.

From the TCS perspective, the probably most important question is about the complexity of computing the $\mathcal{H}$-clique-width. Computing traditional clique-width exactly is NP-hard [14], and hence the same holds for computing $\mathcal{H}$-clique-width exactly in general. However, a question is whether for some special classes $\mathcal{H}$ one could compute exact $\mathcal{H}$-clique-width faster. This is trivially possible, by Claim 2.2 c), when $\mathcal{H}$ is the class of all graphs – which is uninteresting. Is it true that computing $\mathcal{H}$-clique-width exactly is NP-hard for every fixed family $\mathcal{H}$ except in "similarly trivial" cases?

On the other hand, traditional clique-width can be approximated in FPT time with respect to the solution value [18,21]. A big goal would be to extend this approximation result to $\mathcal{H}$-clique-width, perhaps with an additional parameter capturing some properties of $\mathcal{H}$. In particular, with respect to Section 4, we emphasize:

▷ Problem 5.1. Let $\mathcal{P}^\circ$ denote the class of reflexive paths. Can one, for input graph G, approximate $\mathcal{P}^\circ$-cw(G) in FPT time with respect to the solution value?

Next group of questions concerns combinatorial properties proved in this paper. In regard of Section 3, we bring the following one:

▷ Problem 5.2 (cf. Theorem 3.4). Can we characterize families $\mathcal{H}$ of loop graphs such that, for all graphs, bounded $\mathcal{H}$-clique-width implies bounded local clique-width?
This question may be interesting even when restricted to particular graph classes which are of unbounded local clique-width.

A more interesting and natural question, however, comes in a direct relation to the Planar product structure theorem and to Theorem 4.6. We know that graphs of bounded clique-width that do not contain large $K_{t,t}$ subgraphs are as well of bounded tree-width. A natural counterpart of this claim in the context of $\mathcal{P}^\circ$-clique-width would be:

▷ Problem 5.3 (cf. Theorem 4.1, Theorem 4.6). Assume a fixed integer t and an arbitrary graph G such that $\mathcal{P}^\circ$-cw$(G) \leq t$ and G has no $K_{t,t}$ subgraph. Is it then true that $G \subseteq P \boxtimes M$ where P is a path and M is a suitable graph of tree-width bounded in terms of t?

Another question, already mentioned in Section 2, is whether $\mathcal{H}$-clique-width is (at least asymptotically) closed under taking graph complement. This is a prominent and desired property of ordinary clique-width. It would be natural to ask whether, having any simple graph G and its complement $\bar{G}$, we can bound $\mathcal{H}$-cw(G) in terms of $\mathcal{H}$-cw$(\bar{G})$. However, classes $\mathcal{H}$ of bounded degree have bounded local clique-width (Theorem 3.4) and this property is not closed under taking complement.

Instead, we ask whether, for every graph H, there is a graph H' such that for all graphs G, the $\{H'\}$-clique-width of the complement $\bar{G}$ is bounded by a function of the $\{H\}$-clique-width of G. Although we do not have a simple concrete counterexample at hand, we conjecture this is not possible with arbitrary H. One can thus, when being closed under complements is a desirable property, consider only classes $\mathcal{H}$ which are closed under complements themselves (but even that subcase is not trivial), or enrich Definition 2.1 with an operation of adding edges between labels (i, v) and (j, w) over all pairs $(v, w) \in V(H)^2$ such that $vw \notin E(H)$ (note that the latter is much stronger than simply requiring $\mathcal{H}$ to be complement-closed).

We also suggest to study the special case of $\mathcal{T}^\circ$-clique-width when $\mathcal{H} = \mathcal{T}^\circ$ is the family of (all) reflexive trees. The related question in the context of the traditional product structure – that is which graph classes (other than, say, planar graphs) can be expressed as subgraphs of the strong products $T \boxtimes M$ where T is an arbitrary tree and M is of bounded tree-width, does not seem to be explicitly studied yet. We, however, do not have any progress in this direction so far.

Our last batch of questions concerns possible relations of $\mathcal{H}$-clique-width to the currently hot trend of studying structural graph properties through the lens of FO logic on graphs and of FO transductions – transformations of one graph into another defined by FO formulas. In this we use some logic-oriented terms which are not formally defined here (such as, in particular, of *FO transductions*) and we refer for their definitions, e.g., to [6].

First, one may ask for which families $\mathcal{H}$, classes of bounded $\mathcal{H}$-clique-width are monadically dependent, i.e., such that one cannot FO-transduce all finite graphs from graphs of bounded $\mathcal{H}$-clique-width. A partial answer is provided by Theorem 3.4 and Proposition 4.4, but a full characterization of such classes $\mathcal{H}$ is currently out of our reach. As witnessed by Proposition 4.4, classes of bounded $\mathcal{H}$-clique-width can be monadically independent even if $\mathcal{H}$ itself is monadically dependent.

Second, it is interesting to investigate whether and when, having a graph class $\mathcal{G}$ obtained as an FO transduction of a class of bounded $\mathcal{H}$-clique-width, one can find a class $\mathcal{H}^+$ depending on $\mathcal{H}$ (e.g., $\mathcal{H}^+$ an FO transduction of $\mathcal{H}$), such that the $\mathcal{H}^+$-clique-width of $\mathcal{G}$ is bounded. In relation to the Planar product structure, we formulate the following two specific questions in this direction:

▷ Problem 5.4. Assume that a graph class $\mathcal{G}$ is obtained from the class of planar graphs by an FO transduction τ. Is it true that one can give an FO transduction σ, depending on τ, such that the $\mathcal{P}^\sigma$-clique-width of $\mathcal{G}$ is bounded where $\mathcal{P}^\sigma$ is the class of loop graphs obtained from the class of all paths by σ?

Problem 5.4 seems to be much easier if we, instead of requiring bounded $\mathcal{P}^\sigma$-clique-width of every member of $\mathcal{G}$, require only that every graph from $\mathcal{G}$ has a bounded perturbation of bounded $\mathcal{P}^\sigma$-clique-width. This is possibly extensible to classes $\mathcal{H}$ of bounded degree.

▷ Problem 5.5. Let $\mathcal{P}^\sigma$ be the class of loop graphs obtained from the class of all paths by an FO transduction σ. Assume that $\mathcal{G}$ is a graph class of bounded $\mathcal{P}^\sigma$-clique-width and $\mathcal{G}$ is monadically stable, meaning that one cannot define on graphs of $\mathcal{G}$ an arbitrarily long linear order using FO formulas. Is it then true that there exists an FO transduction τ, such that $\mathcal{G}$ is obtained, by τ, from a graph class that admits the traditional product structure?

Third, Theorem 3.4 and Corollary 3.5 can be read as that if a class $\mathcal{H}$ is of bounded degree, then the FO model checking problem is FPT on all classes of bounded $\mathcal{H}$-clique-width. Classes of bounded degree are a prime example of those having FO model checking in FPT [22]. Unfortunately, a bold conjecture claiming that for every class $\mathcal{H}$ having complexity of the FO model checking problem in FPT, the complexity of the FO model checking problem is again in FPT on every class of bounded $\mathcal{H}$-clique-width (perhaps assuming a given decomposition), is very likely false due to the construction given in Proposition 4.4 – the constructed graphs there actually FO transduce all graphs. Hence, it follows from the results of Dreier, Mählmann, and Toruńczyk [8] that FO model checking on classes of bounded $\mathcal{H}_{half}$-clique-width, where $\mathcal{H}_{half}$ denotes the class of reflexive half-graphs, is AW[⋆]-hard. However, a weaker version of this conjecture, with a suitable additional restriction on $\mathcal{H}$, might still be true.

References

1 Marthe Bonamy, Cyril Gavoille, and Michal Pilipczuk. Shorter labeling schemes for planar graphs. *SIAM J. Discret. Math.*, 36(3):2082–2099, 2022. doi:10.1137/20M1330464.

2 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *SODA*, pages 1977–1996. SIAM, 2021. doi:10.1137/1.9781611976465.118.

3 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

4 Édouard Bonnet, O-joung Kwon, and David R. Wood. Reduced bandwidth: a qualitative strengthening of twin-width in minor-closed classes (and beyond). *CoRR*, abs/2202.11858, 2022. arXiv:2202.11858.

5 Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *LICS*, pages 270–279. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.31.

6 Patrice Ossona de Mendez. First-order transductions of graphs (invited talk). In *STACS*, volume 187 of *LIPIcs*, pages 2:1–2:7. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.STACS.2021.2.

7 Guoli Ding, Bogdan Oporowski, James G. Oxley, and Dirk Vertigan. Unavoidable minors of large 3-connected binary matroids. *J. Comb. Theory, Ser. B*, 66(2):334–360, 1996. doi:10.1006/JCTB.1996.0026.

8 Jan Dreier, Nikolas Mählmann, and Szymon Torunczyk. Flip-breakability: A combinatorial dichotomy for monadically dependent graph classes. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1550–1560. ACM, 2024. doi:10.1145/3618260.3649739.

9 Vida Dujmovic, Louis Esperet, Cyril Gavoille, Gwenaël Joret, Piotr Micek, and Pat Morin. Adjacency labelling for planar graphs (and beyond). *J. ACM*, 68(6):42:1–42:33, 2021. doi:10.1145/3477542.

10 Vida Dujmovic, Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David R. Wood. Planar graphs have bounded nonrepetitive chromatic number. *Advances in Combinatorics*, March 2020. doi:10.19086/aic.12100.

11 Vida Dujmovic, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. doi:10.1145/3385731.

12 Vida Dujmovic, Pat Morin, and David R. Wood. Graph product structure for non-minor-closed classes. *J. Comb. Theory, Ser. B*, 162:34–67, 2023. doi:10.1016/J.JCTB.2023.03.004.

13 Zdenek Dvořák, Tony Huynh, Gwenaël Joret, Chun-Hung Liu, and David R. Wood. Notes on graph product structure theory. In *2019-20 MATRIX Annals*, pages 513–533, Cham, 2021. Springer International Publishing.

14 Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.

15 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

16 Petr Hliněný and Jan Jedelský. Twin-width of planar graphs is at most 8, and at most 6 when bipartite planar. In *ICALP*, volume 261 of *LIPIcs*, pages 75:1–75:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.75.

17 Petr Hliněný and Jan Jedelský. H-clique-width and a hereditary analogue of product structure. *CoRR*, abs/2403.16789, 2024. arXiv:2403.16789.

18 Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

19 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In *WG*, volume 13453 of *Lecture Notes in Computer Science*, pages 287–299. Springer, 2022. doi:10.1007/978-3-031-15914-5_21.

20 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/S00453-011-9554-X.

21 Sang-il Oum and Paul 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.

22 D. Seese. Linear time computable problems and first-order descriptions. *Math. Structures Comput. Sci.*, 6(6):505–526, 1996. doi:10.1016/S1571-0661(05)80203-8.

23 Torsten Ueckerdt, David R. Wood, and Wendy Yi. An improved planar graph product structure theorem. *Electron. J. Comb.*, 29(2), 2022. doi:10.37236/10614.

Randomness Versus Superspeedability

Rupert Hölzl
Universität der Bundeswehr München, Germany

Philip Janicki
Universität der Bundeswehr München, Germany

Wolfgang Merkle
Universität Heidelberg, Germany

Frank Stephan
National University of Singapore, Singapore

Abstract

Speedable numbers are real numbers which are algorithmically approximable from below and whose approximations can be accelerated nonuniformly. We begin this article by answering a question of Barmpalias by separating a strict subclass that we will refer to as *superspeedable* from the speedable numbers; for elements of this subclass, acceleration is possible uniformly and to an even higher degree. This new type of benign left-approximation of numbers then integrates itself into a hierarchy of other such notions studied in a growing body of recent work. We add a new perspective to this study by juxtaposing this hierachy with the well-studied hierachy of algorithmic randomness notions.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases superspeedable numbers, speedable numbers, regainingly approximable numbers, regular numbers, left-computable numbers

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.62

Related Version *arXiv Version*: https://arxiv.org/abs/2404.15811

Funding The investigators acknowledge the following partial support: F. Stephan's research was supported by Singapore Ministry of Education AcRF Tier 2 grant MOE-000538-00 and AcRF Tier 1 grants A-0008454-00-00 and A-0008494-00-00, the grant A-0008494-00-00 also supported the research visits of R. Hölzl to the National University of Singapore for the work on this paper.

Acknowledgements The authors would like to thank the referees for helpful and detailed comments.

1 Introduction

In theoretical computer science, the concepts of randomness and of computation speed, as well as their interactions with each other, play an important role. For example, one can study whether access to sources of random information can enable or simplify the computation of certain mathematical objects, for example by speeding up computations of difficult problems. This is a recurring paradigm of theoretical computer science, and there is more than one way of formalizing it, but the perhaps most well-known instance is the difficult open question in the field of complexity theory of whether P is equal to BPP.

A recent line of research could be seen as investigating the reverse direction: to what degree can objects that are random be quickly approximated? Before answering this question, we need to define what we understand by random. The answer is provided by the research field of information theory that deals with questions of compressibility of information, where randomly generated information will be difficult to compress, due to its lack of internal regularities. The subfield of algorithmic randomness, which set out to unify these and other topics and to understand what actually makes a mathematical object random, has produced a wealth of insights into questions about computational properties that random objects must, may, or may not possess. The general pattern in this area is that we put at our disposal algorithmic tools to detect regularities in sequences. Those sequences that are "random enough" to resist these detection mechanisms are then considered random. Many different "resistance levels" have been identified that give rise to randomness notions of different strengths.

For completeness, we define the two central randomness notions featuring in this article, introduced by Martin-Löf [12] and Schnorr [16] respectively: A *randomness test* is a sequence $(U_n)_{n\in\mathbb{N}}$ of uniformly effectively open subsets of $\mathbb{R}$ and we say that it *covers* an $x \in \mathbb{R}$ if and only if $x \in \bigcap_n U_n$. Write μ for the Lebesgue measure on $\mathbb{R}$; if $\mu(U_n) \leq 2^{-n}$ for all $n \in \mathbb{N}$, then we call $(U_n)_{n\in\mathbb{N}}$ a *Martin-Löf test*, and if an $x \in \mathbb{R}$ is not covered by any such test, then we call it *Martin-Löf random.* If we even require that $\mu(U_n) = 2^{-n}$ for all $n \in \mathbb{N}$, then we call $(U_n)_{n\in\mathbb{N}}$ a *Schnorr test*, and an $x \in \mathbb{R}$ that is not covered by any such test *Schnorr random.* It is well-known that Schnorr randomness is a strictly weaker notion than Martin-Löf randomness; that is, there exist more Schnorr random numbers than Martin-Löf random ones. For the rest of the article, we assume that the reader is familiar with these topics; see Calude [3], Downey and Hirschfeldt [4], Li and Vitányi [10], or Nies [14] for comprehensive overviews.

With those randomness paradigms in place, we can study the interactions between randomness and computation speeds. We will focus on a class of objects that can be approximated by an algorithm, and will be interested in the possible speeds of these approximations.

▶ **Definition 1.** *Let x be a real number. For a fixed sequence $(x_n)_n$ of real numbers converging to it from below, write $\rho(n)$ for the quantity $\frac{x_{n+1}-x_n}{x-x_n}$ for each $n \in \mathbb{N}$, and call $(\rho(n))_n$ the* speed quotients *of $(x_n)_n$.*

▶ **Definition 2.** *If there exists a* left-approximation *of x, that is, a computable increasing sequence of rational numbers converging to x, then we call x* left-computable*. If there even exists a left-approximation $(x_n)_n$ with speed quotients $(\rho(n))_n$ such that $\rho := \limsup_{n\to\infty} \rho(n) > 0$, then we call x* speedable*.*

Left-computable numbers can be Martin-Löf random; the standard example is *Chaitin's* Ω, the measure of the domain of an optimally universal prefix-free Turing machine. It is also known that there exist left-computable Schnorr randoms that are not Martin-Löf random.

The notion of speedability was introduced by Merkle and Titov. They made the interesting observation that speedability is incompatible with Martin-Löf randomness [13, Theorem 10] and asked the question whether the converse holds, that is, whether there exist left-computable numbers which are neither speedable nor Martin-Löf random. A positive answer would have been an interesting new characterization of Martin-Löf randomness; however, Hölzl and Janicki [8, Corollary 62] showed that such a characterization does *not* hold. They also introduced the term *benign approximations* when referring to subclasses of the left-computable numbers that possess special approximations that are in some sense better behaved than general approximations from below. One example is speedability; another that will play a role in this article is the following.

▶ **Definition 3** (Hertling, Hölzl, Janicki [7]). *A real number x is called* regainingly approximable *if there exists a computable increasing sequence of rational numbers $(x_n)_n$ converging to x with $x - x_n \leq 2^{-n}$ for infinitely many $n \in \mathbb{N}$.*

Obviously, every regainingly approximable number is left-computable and every computable number is regainingly approximable, but none of these implications can be reversed by results of Hertling, Hölzl, Janicki [7].

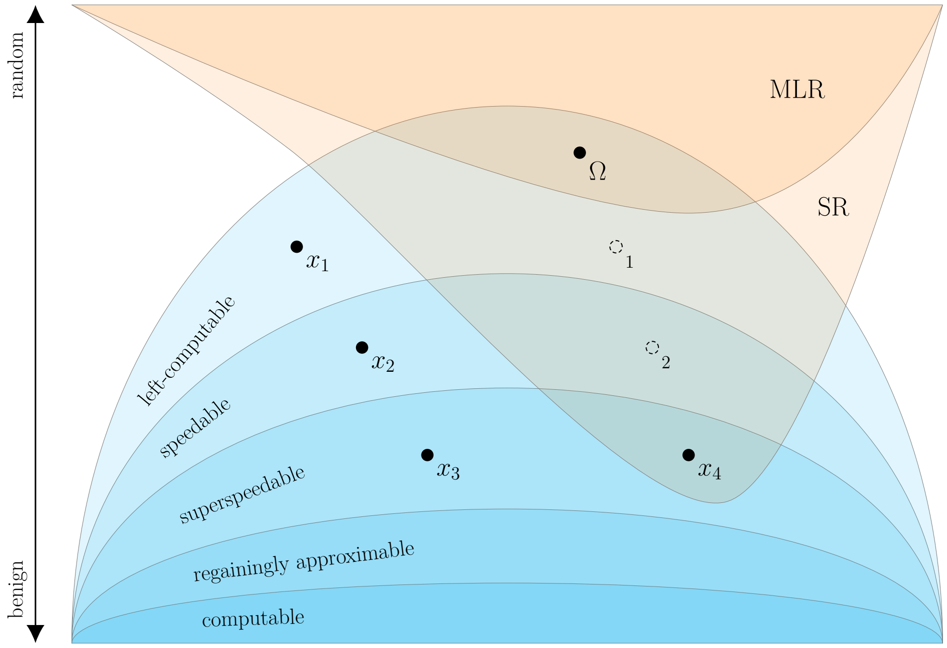


Figure 1 The number x_1 was constructed by Hölzl and Janicki [8, Corollary 62]; by construction it is nearly computable, thus weakly 1-generic by a result of Hoyrup [9], thus not Schnorr random (see, for instance, Downey and Hirschfeldt [4, Proposition 8.11.9]). The number x_2 constructed in Theorem 8 cannot be Schnorr random either, because it is by construction not immune. To see that x_3 exists, consider the example used in the proof of Corollary 16; it is strongly left-computable, thus not immune, thus not Schnorr random. Finally, x_4 is constructed in Theorem 21.
It is an open question whether numbers ◌$_1$ or ◌$_2$ exist; see the discussion in Section 6.

Merkle and Titov also showed that the property of being speedable does not depend on the choice of ρ, that is, if some left-computable number is speedable via some ρ witnessed by some left-approximation, it is also speedable via any other $\rho' < 1$ witnessed by some other left-approximation. However, their proof is highly nonuniform, which led to Barmpalias [1] asking whether every such number also has a *single* left-approximation with $\limsup_{n\to\infty} \rho(n) = 1$.

We answer this question negatively in this article by showing that the requirement to achieve $\limsup_{n\to\infty} \rho(n) = 1$ is a strictly stronger condition that defines a strict subset of the speedable numbers which we will call superspeedable. We then show that the regainingly approximable numbers are strictly contained in this subset. Thus within the speedable numbers there is a strict hierarchy of notions of benign approximations, namely computable implies regainingly approximable implies superspeedable implies speedable.

Wu [18] called finite sums of binary expansions of computably enumerable sets *regular reals*. We show that these reals are a strict subset of the superspeedable numbers as well; in particular every binary expansion of a computably enumerable set is superspeedable. Then we show that finite sums of left-computable numbers can only be superspeedable if at least one of their summands is superspeedable.

With this new approximation notion identified and situated in the hierarchy of benign approximability, it is then natural to wonder how it interacts with randomness. We will show that unlike the Martin-Löf randoms, which by the result of Merkle and Titov [13] cannot even be speedable, the Schnorr randoms may even be superspeedable; however they cannot be regainingly approximable.

2 Speedability versus immunity

Merkle and Titov [13] showed that all non-high and all strongly left-computable numbers are speedable. On the other hand they established that Martin-Löf randoms cannot be speedable. In this section we show that a left-computable number which is not immune has to be speedable.

We first observe that, as an alternative to Definition 2 above, speedability could also be defined using two other forms of speed quotients; this will prove useful in the following.

▶ **Proposition 4.** *Let $x \in \mathbb{R}$. For an increasing sequence of rational numbers $(x_n)_n$ converging to x with speed quotients $(\rho(n))_n$ the following statements are easily seen to be equivalent:*

1. $\limsup_{n\to\infty} \rho(n) > 0$*, that is,* $(x_n)_n$ *witnesses the speedability of* x*;*
2. $\limsup_{n\to\infty} \frac{x_{n+1}-x_n}{x-x_{n+1}} > 0$*;*
3. $\liminf_{n\to\infty} \frac{x-x_{n+1}}{x-x_n} < 1$. ◀

For any set $A \subseteq \mathbb{N}$, we define the real number $x_A := \sum_{n\in A} 2^{-(n+1)} \in [0,1]$. Clearly, A is computable if and only if x_A is computable. If A is only assumed to be computably enumerable, then x_A is a left-computable number; the converse is not true as pointed out by Jockusch (see Soare [17]). If there does exist a computably enumerable set $A \subseteq \mathbb{N}$ with $x_A = x$, then $x \in [0,1]$ is called *strongly left-computable.*

Recall that an infinite set $A \subseteq \mathbb{N}$ is *immune* if it does not have an infinite computably enumerable (or, equivalently, computable) subset. We call it *biimmune* if both A and its complement $\bar{A}$ are immune. A real number $x \in [0,1]$ is called *immune* if there exists an immune set $A \subseteq \mathbb{N}$ with $x_A = x$; analogously for *biimmune.*

▶ **Theorem 5.** *Every left-computable number that is not immune is speedable.*

Before we give the proof we mention that left-computable numbers

- whose binary expansion is not biimmune or
- whose binary expansion is of the form $A \oplus A$

can be shown to be speedable by similar arguments as below.

Proof of Theorem 5. Let x be a left-computable real that is not immune. If it is computable, the theorem holds trivially, thus assume otherwise. Let A be such that $x = x_A$; then there exists a computable increasing function $h\colon \mathbb{N} \to \mathbb{N}$ with $h(\mathbb{N}) \subseteq A$. Let $(B[n])_n$ be a computable sequence of finite sets of natural numbers such that the sequence $(x_{B[n]})_n$ is increasing and converges to x_A. Define the function $r\colon \mathbb{N} \to \mathbb{N}$ recursively via

$$r(0) := 0, \quad r(n+1) := \min\{m > r(n)\colon \{h(0), \dots, h(n)\} \subseteq B[m]\}.$$

It is easy to see that r is well-defined, computable and increasing. Let

$$s_n := \min\{m \in \mathbb{N}\colon\ m \in B[r(n+1)] \setminus B[r(n)]\}$$

for all $n \in \mathbb{N}$. Clearly, the sequence $(s_n)_n$ is well-defined and tends to infinity. Finally, define the sequence $(C[n])_n$ via

$$C[0] := \emptyset, \quad C[n+1] := B[r(n+1)] \restriction (s_n + 1),$$

for all $n \in \mathbb{N}$. Clearly, $(C[n])_n$ is a computable sequence of finite sets of natural numbers, which converge pointwise to A, and the sequence $(x_{C[n]})_n$ is increasing and converges to x.

We claim that there are infinitely many n such that $s_n < h(n)$; this is because otherwise, for almost all n, the sets $B[r(n)], B[r(n+1)], \dots$ would all mutually agree on their first $h(n) - 1$ bits, hence their limit A would be computable, contrary to our assumption. Thus, fix one of the infinitely many n with $s_n < h(n)$ and let $n' \geq n$ be maximal such that $s_{n'} < h(n)$. Then the sets $B[r(n'+1)], B[r(n'+2)], \ldots$ all contain $h(n)$ and, by definition of s and n', agree mutually on their first $(h(n)-1)$ bits; in other words, they even agree on their first $h(n)$ bits. But by choice of n', these sets also agree with the sets $C[n'+1], C[n'+2], \ldots$, respectively, on their first $h(n)$ bits; thus these latter sets must also contain $h(n)$ and agree mutually on their first $h(n)$ bits. By definition, the set $C[n']$ does *not* contain $h(n)$; thus, in summary, we have $x_{C[n'+1]} - x_{C[n']} \geq 2^{-h(n)}$ and $x - x_{C[n'+1]} \leq 2^{-h(n)}$, hence $\frac{x_{C[n'+1]}-x_{C[n']}}{x-x_{C[n'+1]}} \geq 1$.

Since there are infinitely many such n and corresponding n', the left-approximation $(x_{C[n]})_n$ of x witnesses that x is speedable. ◀

3 Superspeedability

We now strengthen the conditions in Proposition 4 to obtain the following new definition.

▶ **Definition 6.** *Let $x \in \mathbb{R}$. For an increasing sequence $(x_n)_n$ converging to x with speed quotients $(\rho(n))_n$ the following statements are easily seen to be equivalent:*

1. $\limsup_{n\to\infty} \rho(n) = 1$*;*
2. $\limsup_{n\to\infty} \frac{x_{n+1}-x_n}{x-x_{n+1}} = \infty$*;*
3. $\liminf_{n\to\infty} \frac{x-x_{n+1}}{x-x_n} = 0$.

We call x superspeedable *if it has a left-approximation $(x_n)_n$ with these properties.*

The following statement, which bears some similarity to Theorem 5, shows that one simple cause for a left-computable number to be superspeedable is for it to have a left-approximation which permanently leaves arbitrarily long blocks of bits fixed to 0.

▶ **Proposition 7.** *Fix a sequence $(I_i)_{i\in\mathbb{N}}$ of disjoint finite intervals in $\mathbb{N}$ with the property that $|I_i| \geq i$ for all n. If $x \in \mathbb{R}$ has a left-approximation $(x_n)_{n\in\mathbb{N}}$ consisting only of dyadic rationals and such that for all $i \in \mathbb{N}$ and $n \in \mathbb{N}$ we have $x_n(j) = 0$ for all $j \in I_i$, then x is superspeedable.*

Proof. Fix any i and let n be maximal such that $x_n(j) \neq x(j)$ for some $j < \min I_i$; that is, let $n+1$ be the last stage at which a bit left of I_i changes during the approximation of x. Then, on the one hand, $x_{n+1} - x_n \geq 2^{-\min(I_i)}$. On the other hand, by the maximality of n and by the assumptions on $(x_n)_{n\in\mathbb{N}}$, we must have $x - x_{n+1} \leq 2^{-\max(I_i)}$. Therefore we have

$$\frac{x-x_{n+1}}{x-x_n} \leq \frac{x-x_{n+1}}{x_{n+1}-x_n} \leq \frac{2^{-\max(I_i)}}{2^{-\min(I_i)}} \leq 2^{-(i-1)},$$

which tends to 0 as i tends to infinity. ◀

Obviously, every superspeedable number is speedable. Barmpalias [1] asked whether the converse is true as well. We give a negative answer with the following theorem and corollary.

▶ **Theorem 8.** *There is a left-computable number that is neither immune nor superspeedable.*

Proof. Let $r\colon \mathbb{N} \to \mathbb{N}$ be the function defined by $r(n) := \max\{\ell \in \mathbb{N}\colon\ 2^\ell \text{ divides } n+1\}$, that is, the characteristic sequence of r is the member of Baire space with the initial segment

0102010301020104010201 03 . . .

We construct a number α by approximating its binary expansion A. We will slightly abuse notation and silently identify real numbers with their infinite binary expansions; similarly we identify finite binary sequences σ with the rational numbers $0.\sigma$.

We approximate A in stages where during stage $s \in \mathbb{N}$ we define a set $A[s]$ such that the sets $A[s]$ converge pointwise to A. For all s and all n with $r(n) = 0$, let $A[s](n) = 0$; and for all s and all n with $r(n) = 1$, let $A[s](n) = 1$. We refer to all remaining bits as the *coding bits*. More precisely, we refer to all bits with $r(n) = e + 2$ as the *e-coding bits* and we let $(c_i^e)_i$ be the sequence of the positions of all e-coding bits in ascending order.

If $b = c_i^e$ for some e and i, then we say that position b is *threatened at stage s* if $A[s](b) = 0$ and $A[s]\restriction c_{i+1}^e = B_e[s]\restriction c_{i+1}^e$, where $(B_0[s])_s, (B_1[s])_s, \ldots$ is some fixed uniformly effective enumeration of all left-computable approximations.

Construction. At every stage s, if there exists a least coding bit $b \leq s$ that is threatened, then set $A[s+1](b) = 1$ and $A[s+1](c) = 0$ for all coding bits $c > b$. For all other $n \in \mathbb{N}$, leave bit n unchanged, that is, $A[s+1](n) = A[s](n)$.

Verification. By construction, once a coding bit has been set to 1, it can only be set back to 0 if a coding bit at an earlier position is set from 0 to 1. From this it follows by an obvious inductive argument that $A[s]$ converges to some limit A in a left-computable fashion; that is, interpreted as real numbers, the values $A[0], A[1], \ldots$ are nondecreasing. Thus α is left-computable.

As it contains all numbers n with $r(n) = 1$, the set A we are constructing is trivially not immune. All that remains to show is that it is not superspeedable. To that end, fix any e and assume that that the left-computable approximation $(B_e[s])_s$ converges to A; as otherwise there is nothing to prove for e. We will show that this left-computable approximation does not witness superspeedability of A.

▷ Claim. There are stages $s_0 < s_1 < \ldots$ such that for all $i \geq 0$, at stage s_i, the coding bit $b_i = c_i^e$ is threatened for the last time and is the least coding bit that is threatened at this stage, and such that A has already converged at this stage up to its $(b_i + 4)$-th bit; that is, for all $\ell \geq 1$ we have $A[s_i]\restriction(b_i + 4) = A[s_i + \ell]\restriction(b_i + 4) = A\restriction(b_i + 4)$.

Proof. Fix some e-coding bit $b_i = c_i^e$. By the discussion preceding the claim, there is some least stage s such that set A has already converged up to its b_i-th bit. By construction, it must hold that $A[s](b_i) = 0$ and since the left-approximation $(B_e[s])_s$ converges to A, there must be a stage $s' > s$ at which b_i becomes threatened and is the least threatened coding bit. Thus, during stage s', the bit $A(b_i)$ is set permanently to 1 and, because the two bits following b_i are no coding bits, A has now converged up to its $(b_i + 4)$-th bit. Therefore, it suffices to set $s_i = s'$ and to observe that we have $s_i < s_{i+1}$ for all i because e-coding bit c_{i+1}^e is set to 0 at stage s_i, hence is threatened again after stage s_i. ◁

Let $s_0, s_1, \ldots$ be the stages from the last claim.

▷ Claim. There is an $\rho_e < 1$ only depending on e such that for all i and $s \in \{s_i, \ldots, s_{i+1} - 1\}$,

$$\frac{B_e[s+1] - B_e[s]}{A - B_e[s]} \leq \rho_e.$$

Proof. Fix some i and some stage s as above. Let $b = c_i^e$ and let $b' = c_{i+1}^e$. Then A has already converged up to its b-th bit at stage s_i and by definition of a threat we must have $B_e[s_i]\restriction b = A[s_i]\restriction b$. But the left-approximation $(B_e[s])_s$ converges to A, hence can never "overshoot" A and thus at stage s_i has already converged up to its b-th bit as well. Consequently, we have $B_e[s+1] - B_e[s] \leq 2^{-(b-1)}$.

By construction and our assumptions,

- at stage s_{i+1}, bit b' is set from 0 to 1,
- bit $b'+1$ permanently maintains value 0 and,
- by definition of a threat, $A[s_{i+1}]$ and $B_e[s_{i+1}]$ agree on their first $b'+2 < c^e_{i+2}$ many bits.

This implies that

$$A - B_e[s+1] \geq A[s_{i+1}+1] - B_e[s_{i+1}] \geq 2^{-(b'+1)} = 2^{-(b+2^{e+3}+1)},$$

and in summary we have

$$\frac{B_e[s+1]-B_e[s]}{A-B_e[s]} = \frac{B_e[s+1]-B_e[s]}{(A-B_e[s+1])+(B_e[s+1]-B_e[s])} \leq \frac{2^{-(b-1)}}{2^{-(b+2^{e+3}+1)}+2^{-(b-1)}} \leq \frac{2^{2+2^{e+3}}}{1+2^{2+2^{e+3}}} < 1,$$

hence there is a constant $\rho_e < 1$ as claimed. ◁

Since e was arbitrary such that $(B_e[s])_s$ converges to A, there cannot be a left-approximation that witnesses superspeedability of A. ◀

By Theorem 5, the number α constructed in Theorem 8 is speedable.

▶ **Corollary 9.** *There exists a speedable number which is not superspeedable.* ◀

Thus, superspeedability is strictly stronger than speedability; we now proceed to identify interesting classes of numbers that have this property. We begin with the class of regainingly approximable numbers. It was shown by Hölzl and Janicki [8] that regainingly approximable numbers are speedable. This result can be strengthened; to show that regainingly approximable numbers are in fact superspeedable, we will use the following characterization.

▶ **Proposition 10** (Hertling, Hölzl, Janicki [7]). *For a left-computable number x the following are equivalent:*

(1) *x is regainingly approximable.*

(2) *For every computable unbounded function $f\colon \mathbb{N} \to \mathbb{N}$ there exists a computable increasing sequence of rational numbers $(x_n)_n$ converging to x with $x - x_n \leq 2^{-f(n)}$ for infinitely many $n \in \mathbb{N}$.*

▶ **Theorem 11.** *Every regainingly approximable number is superspeedable.*

Proof. Let x be a regainingly approximable number, and fix a computable increasing sequence $(x_n)_n$ of rational numbers that converges to x and satisfies $x - x_n \leq \frac{1}{(n+1)!}$ for infinitely many $n \in \mathbb{N}$. Define the sequence $(y_n)_n$ with $y_n := x_n - \frac{1}{n!}$ for all $n \in \mathbb{N}$. Surely, this sequence is computable, increasing and converges to x as well. Considering any of the $n \in \mathbb{N}$ with $x - x_n \leq \frac{1}{(n+1)!}$, we obtain

$$\frac{y_{n+1}-y_n}{x-y_n} = \frac{(x_{n+1}-x_n)+\frac{1}{n!}-\frac{1}{(n+1)!}}{(x-x_n)+\frac{1}{n!}} > \frac{\frac{1}{n!}-\frac{1}{(n+1)!}}{\frac{1}{n!}+\frac{1}{(n+1)!}} = \frac{\frac{n}{(n+1)!}}{\frac{n+2}{(n+1)!}} = \frac{n}{n+2}.$$

Thus we have $\limsup_{n\to\infty} \frac{y_{n+1}-y_n}{x-y_n} = 1$. Therefore, x is superspeedable. ◀

The next class of numbers that we investigate is the following.

▶ **Definition 12** (Wu). *A real number is called* regular *if it can be written as a finite sum of strongly left-computable numbers.*

Note that this includes in particular all binary expansions of computably enumerable sets. Using the following helpful definition and propositions, we now show that every regular number is superspeedable.

▶ **Definition 13.** *Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function which tends to infinity. Then the function $u_f\colon \mathbb{N} \to \mathbb{N}$ is defined by $u_f(n) := |\{k \in \mathbb{N}\colon f(k) = n\}|$ for all $n \in \mathbb{N}$.*

If f is computable, then u_f possesses a computable approximation from below, namely $u_f[0](n) := 0$ and, for all $n, t \in \mathbb{N}$,

$$u_f[t+1](n) := \begin{cases} u_f[t](n) + 1 & \text{if } f(t) = n, \\ u_f[t](n) & \text{otherwise.} \end{cases}$$

▶ **Proposition 14.** *A real number x is regular if and only if there exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$ with $\sum_{k=0}^{\infty} 2^{-f(k)} = x$, a so-called* name *of f, such that u_f is bounded by some constant.* ◀

▶ **Theorem 15.** *Every regular number is superspeedable; in particular this holds for every strongly left-computable number.*

Proof. Let x be a regular number. Fix some computable name $f\colon \mathbb{N} \to \mathbb{N}$ for x and some constant $c \in \mathbb{N}$ with $u_f(n) \leq c$ for all $n \in \mathbb{N}$. We define two sequences of natural numbers $(a_n)_n$ and $(b_n)_n$ and a sequence of sets of natural numbers $(I_n)_n$ by

$$a_n := \frac{n(n+1)}{2}, \quad b_n := a_n + n = a_{n+1} - 1, \quad I_n := \{a_n, \ldots, b_n\},$$

for all $n \in \mathbb{N}$. It is easy to verify that the set $\{I_n\colon n \in \mathbb{N}\}$ is a partition of $\mathbb{N}$ and that we also have $|I_n| = n + 1$ for all $n \in \mathbb{N}$.

We call a stage $s \in \mathbb{N}$ a *true stage* if we have $f(s) < f(t)$ for all $t > s$. Since f tends to infinity, there are infinitely many true stages. For any $n, s \in \mathbb{N}$, we say that the interval I_n is *incomplete* at stage s if there exists some stage $t > s$ with $f(t) \in I_n$. Otherwise, we say that I_n is *complete* at s. Since f tends to infinity, every interval is complete at some stage. Therefore, we can define the sequence $(\theta_n)_n$ with $\theta_n := \min\{t \in \mathbb{N}\colon I_n \text{ is complete at stage } t\}$ for all $n \in \mathbb{N}$. In order to show that x is superspeedable, we consider several cases:

- If there are infinitely many $n \in \mathbb{N}$ with $f(\mathbb{N}) \cap I_n = \emptyset$, then x is superspeedable for the reasons considered in Proposition 7; more explicitly, in this case, consider such an n with $a_n > \min_{k\in\mathbb{N}} f(k)$ and let $s \in \mathbb{N}$ be the latest true stage with $f(s) < a_n$. Then we have $\sum_{k=s+1}^{\infty} 2^{-f(k)} \leq c \cdot 2^{-b_n}$ due to $f(\mathbb{N}) \cap I_n = \emptyset$, and we obtain

$$\frac{2^{-f(s)}}{\sum_{k=s+1}^{\infty} 2^{-f(k)}} > \frac{2^{-a_n}}{c \cdot 2^{-b_n}} = \frac{2^{-a_n}}{c \cdot 2^{-(a_n+n)}} = \frac{2^n}{c}.$$

  As there exist infinitely many n as above, x is superspeedable.
- Otherwise, we can assume w.l.o.g. that $f(\mathbb{N}) \cap I_n \neq \emptyset$ for all $n \in \mathbb{N}$.
  Suppose that there are infinitely many $n \in \mathbb{N}$ with some stage $s \in \mathbb{N}$ at which I_n is incomplete and I_{n+1} is complete. Let $t > s$ be the earliest true stage at which $I_1, \ldots, I_n$ are all complete. On the one hand, we have $f(t) < a_{n+1}$. On the other hand we have $\sum_{k=t+1}^{\infty} 2^{-f(k)} \leq c \cdot 2^{-b_{n+1}}$, since, in particular, I_{n+1} is complete at stage t. We obtain

$$\frac{2^{-f(s)}}{\sum_{k=s+1}^{\infty} 2^{-f(k)}} > \frac{2^{-a_{n+1}}}{c \cdot 2^{-b_{n+1}}} = \frac{2^{-a_{n+1}}}{c \cdot 2^{-(a_{n+1}+n+1)}} = \frac{2^{n+1}}{c}.$$

  As there exist infinitely many n as above, x is superspeedable.

- Otherwise, we can assume w.l.o.g. that $(\theta_n)_n$ is an increasing sequence. This also implies that θ_n is a true stage for every $n \in \mathbb{N}$. Define the sequence $(d_n)_n$ by $d_n := b_n - f(\theta_n)$. Suppose that $(d_n)_n$ is unbounded; then for every $n \in \mathbb{N}$ we have

$$\frac{2^{-f(\theta_n)}}{\sum_{k=\theta_n+1}^{\infty} 2^{-f(k)}} = \frac{2^{-f(\theta_n)}}{c \cdot 2^{-b_n}} = \frac{2^{b_n - f(\theta_n)}}{c} = \frac{2^{d_n}}{c}.$$

  Therefore, x is superspeedable.

- Otherwise, $(d_n)_n$ is bounded. Fix some natural number $D \geq 1$ with $d_n \leq D$ for all $n \in \mathbb{N}$. Define the sequences $(T_n)_n$ and $(T_n[t])_{n,t}$ in $\{0, \ldots, c\}^D$ as follows:

$$T_n := (u_f(b_n - D + 1), \ldots, u_f(b_n)), \quad T_n[t] := (u_f[t](b_n - D + 1), \ldots, u_f[t](b_n))$$

  Clearly, we have $\lim_{t\to\infty} T_n[t] = T_n$ for all $n \in \mathbb{N}$. Fix some D-tuple $T \in \{0, \ldots, c\}^D$ satisfying $T_n = T$ for infinitely many $n \in \mathbb{N}$. Assume w.l.o.g. that there are infinitely many odd numbers n with this property, and recursively define the function $s\colon \mathbb{N} \to \mathbb{N}$ as follows: Let $s(0) := 0$, and define $s(m+1)$ as the smallest stage $t > s(m)$ such that there is some odd number $n \in \mathbb{N}$ with $T_n[t] = T$ and $T_n[t] \neq T_n[s(m)]$. Clearly, s is well-defined, computable and increasing. Finally, define the computable and increasing sequence $(x_n)_n$ by $x_n := \sum_{k=0}^{s(n)} 2^{-f(k)}$ for all $n \in \mathbb{N}$; this clearly converges to x. Let $n \in \mathbb{N}$ be an even number with $T_{n+1} = T$. By the previous assumption that $(\theta_n)_n$ is increasing, there exists a uniquely determined number $m \in \mathbb{N}$ with $s(m+1) = \theta_{n+1}$. Similarly, by the definition of s, we have $s(m) < \theta_n$. On the one hand, we have $x_{m+1} - x_m \geq 2^{-b_n}$. On the other hand, we have $x - x_{m+1} \leq c \cdot 2^{-b_{n+1}}$. So we finally obtain

$$\frac{x_{m+1} - x_m}{x - x_{m+1}} \geq \frac{2^{-b_n}}{c \cdot 2^{-b_{n+1}}} = \frac{2^{b_{n+1} - b_n}}{c} = \frac{2^{a_{n+1} - a_n + 1}}{c} = \frac{2^{n+2}}{c}.$$

  As there exist infinitely many n as above, x is superspeedable. ◀

As a corollary of the theorem, we obtain that the converse of Theorem 11 does not hold.

▶ **Corollary 16.** *Not every superspeedable number is regainingly approximable.*

Proof. By Theorem 15, every strongly left-computable number is superspeedable. However, Hertling, Hölzl, and Janicki [7] constructed a strongly left-computable number that is not regainingly approximable. ◀

Similarly, the converse of Theorem 15 is not true either, as the following observation shows.

▶ **Proposition 17.** *Not every superspeedable number is regular.*

Proof. Hertling, Hölzl, and Janicki [7] constructed a regainingly approximable number α such that for infinitely many n it holds that $K(\alpha\restriction n) > n$. By Theorem 11, this α is superspeedable. However, it cannot be regular, as it is easy to see that the initial segment Kolmogorov complexities of regular numbers must be logarithmic everywhere. ◀

4 Superspeedability and additivity

In this section we study the behaviour of superspeedability with regards to additivity. The following technical statement will be useful.

▶ **Lemma 18.** *Let α and β be left-computable numbers, and let $(c_n)_n$ be a computable increasing sequence of rational numbers converging to $\alpha + \beta$. Then there exist computable increasing sequences of rational numbers $(a_n)_n$ and $(b_n)_n$ converging to α and β, respectively, with $a_n + b_n = c_n$ for all $n \in \mathbb{N}$.*

Proof. Given two computable increasing sequences of rational numbers $(d_n)_n$ and $(e_n)_n$ converging to α and β, respectively, fix $N \in \mathbb{N}$ with

$$-N < \min\{d_0, e_0\} \quad \text{and} \quad -2N < c_0,$$

and define $a_{-1} := b_{-1} := -N$ and $c_{-1} := -2N$. We will inductively define an increasing function $s\colon \mathbb{N} \to \mathbb{N}$ as well as the desired sequences $(a_n)_n$ and $(b_n)_n$.

For every $n \in \mathbb{N}$, define $s(n)$ such that the inequalities

$$d_{s(n)} > a_{n-1}, \quad e_{s(n)} > b_{n-1}, \quad d_{s(n)} + e_{s(n)} \geq c_n$$

are satisfied; this is obviously always possible. Assume by induction that $a_{n-1} + b_{n-1} = c_{n-1}$. Note that the quantities $d_{s(n)} - a_{n-1}$ and $e_{s(n)} - b_{n-1}$ and $c_n - c_{n-1}$ are positive rational numbers. Thus there exist natural numbers $p_a, p_b, q \geq 1$ and $p_c \geq 2$ with

$$d_{s(n)} - a_{n-1} = \frac{p_a}{q}, \quad e_{s(n)} - b_{n-1} = \frac{p_b}{q}, \quad c_n - c_{n-1} = \frac{p_c}{q}.$$

This implies

$$\frac{p_a}{q} + \frac{p_b}{q} = \left(d_{s(n)} + e_{s(n)}\right) - (a_{n-1} + b_{n-1}) \geq c_n - c_{n-1} = \frac{p_c}{q},$$

hence $p_a + p_b \geq p_c$. Let $m := \min\{p_a, p_c - 1\}$, and finally define $a_n := a_{n-1} + \frac{m}{q}$ and $b_n := b_{n-1} + \frac{p_c - m}{q}$. It is easy to verify that $(a_n)_n$ and $(b_n)_n$ are both computable and increasing sequences, converge to α and β, respectively, and satisfy $a_n + b_n = c_n$ for all $n \in \mathbb{N}$. ◀

Using the lemma, we can establish the following result.

▶ **Theorem 19.** *Let α and β be left-computable numbers such that $\alpha + \beta$ is superspeedable. Then at least one of α or β must be superspeedable.*

We point out that it can be shown that the converse does not hold.

Proof. Suppose that neither α nor β are superspeedable. Let $\gamma := \alpha + \beta$, and let $(c_n)_n$ be a computable increasing sequence of rational numbers converging to γ. We show that there is some constant $\rho \in (0, 1)$ with $\frac{c_{n+1} - c_n}{\gamma - c_n} \leq \rho$ for all $n \in \mathbb{N}$. Due to Lemma 18, there exist computable increasing sequences of rational numbers $(a_n)_n$ and $(b_n)_n$ converging to α and β, respectively, with $a_n + b_n = c_n$ for all $n \in \mathbb{N}$. Since α and β are both not superspeedable, there is some constant $\rho \in (0, 1)$ satisfying both $\frac{a_{n+1} - a_n}{\alpha - a_n} \leq \rho$ and $\frac{b_{n+1} - b_n}{\beta - b_n} \leq \rho$ for all $n \in \mathbb{N}$. Considering some arbitrary $n \in \mathbb{N}$, we obtain

$$\begin{aligned}\frac{c_{n+1} - c_n}{\gamma - c_n} &= \frac{(a_{n+1} - a_n) + (b_{n+1} - b_n)}{(\alpha - a_n) + (\beta - b_n)} = \frac{\frac{a_{n+1} - a_n}{\alpha - a_n} \cdot (\alpha - a_n) + \frac{b_{n+1} - b_n}{\beta - b_n} \cdot (\beta - b_n)}{(\alpha - a_n) + (\beta - b_n)} \\ &\leq \frac{\max\left\{\frac{a_{n+1} - a_n}{\alpha - a_n}, \frac{b_{n+1} - b_n}{\beta - b_n}\right\} \cdot ((\alpha - a_n) + (\beta - b_n))}{(\alpha - a_n) + (\beta - b_n)} \leq \rho.\end{aligned}$$

Thus, γ is not superspeedable either. ◀

5 Benignness versus randomness

Recall that Merkle and Titov [13, Theorem 10] made the observation that Martin-Löf randomness is incompatible with speedability. In this section, we study the analogous question for the weaker randomness notion of Schnorr randomness. We begin with the easy observation that Schnorr randomness is incompatible with the rather demanding benignness notion of regaining approximability; the argument is a straight-forward modification of the result of Merkle and Titov.

▶ **Proposition 20.** *No regainingly approximable number can be Schnorr random.*

Proof. Let x be regainingly approximable, witnessed by its left-approximation $(x_n)_n$. Then x is covered by the Solovay test $(S_n)_n$ where $S_n = (x_n - 2^{-(n-1)}, x_n + 2^{-(n-1)})$ for all $n \in \mathbb{N}$. This test is *total*, that is, the sum of the measures of its components is computable; then, by a result of Downey and Griffiths [5, Theorem 2.4], x cannot be Schnorr random. ◀

In light of this result, it is natural to ask how far down into the hierarchy of benignness notions the class of Schnorr randoms reaches. The answer is given by the following theorem.

▶ **Theorem 21.** *There exists a superspeedable number which is Schnorr random.*

We point out that the number we construct will have the additional property that it is not partial computably random.

For an infinite set $A \subseteq \mathbb{N}$ we write p_A for the unique increasing function from $\mathbb{N}$ to $\mathbb{N}$ such that $p_A(\mathbb{N}) = A$. Recall, for instance from Odifreddi [15, Section III.3], that a set A is called *dense simple* if it is computably enumerable and $p_{\bar{A}}$ dominates every computable function. While the complement of a dense simple set must be very thin by definition, we claim that there does exist such a set A whose binary expansion contains arbitrarily long sequences of zeros. To see this, recall that a set B is called *maximal* if it is computably enumerable and coinfinite and for any computably enumerable set C with $B \subsetneq C$, we must have that C is cofinite or that $C \setminus B$ is finite; in other words, if B only has trivial computably enumerable supersets. Such sets exist, for instance see Odifreddi [15, Section III.4]. If we partition the natural numbers into a sequence $(I_n)_n$ of successive intervals of growing length by letting $I_0 := \{0\}$, and $I_{n+1} := \{\max I_n + 1, \ldots, \max I_n + n + 2\}$ for all $n \in \mathbb{N}$ and fix some maximal set B, then it is easy to check that we obtain a set A as required by letting

$$A := \left\{ n \colon \; \exists k, \ell \left[\begin{array}{c} (n \in I_k \wedge k \in B) \vee \\ (n \text{ is the } \ell\text{-th smallest element of } I_k \wedge \#(B{\restriction}k) \geq \ell) \end{array} \right] \right\}.$$

Proof of Theorem 21. Let A be a set that is dense simple and whose binary expansion contains arbitrarily long sequences of zeros. Define Ω_A bitwise via

$$\Omega_A(n) := \begin{cases} \Omega(m) & \text{if } n = p_A(m), \\ 0 & \text{else;} \end{cases}$$

for all $n \in \mathbb{N}$; that is, Ω_A contains all bits of Ω, but at those places that are elements of A; all other bits of Ω_A are zeros. Let $(A[t])_t$ be a computable enumeration of A and for all $n \in \mathbb{N}$ write $p_A(n)[t]$ for the n-th smallest element in $A[t]$, if it already exists. Then it is easy to see that $(\Omega_A[t])_t$ defined bitwise via

$$\Omega_A(n)[t] := \begin{cases} \Omega(m)[t] & \text{if } n = p_A(m)[t]\downarrow, \\ 0 & \text{else,} \end{cases}$$

for all $n \in \mathbb{N}$, is a left-approximation of Ω_A; namely, the individual bits of Ω are approximated in a left-computable fashion, and as more and more bits appear in the computable enumeration of A, the positions where the bits of Ω get stored in Ω_A may move to the left.

We claim that Ω_A is as required by the theorem.

- That Ω_A is superspeedable follows from Proposition 7. Namely, recall that A contains arbitrarily long blocks of bits that will permanently maintain value 0, that is, for all n in such a block and all t, we have $\Omega_A(n)[t] = 0$. As a consequence, the left-approximation of Ω_A described above witnesses superspeedability at the infinitely many stages where for the last time a bit left of one of these blocks changes.
- Note that the fact that $\bar{A}$ is dense immune implies that for every computable increasing function $f\colon \mathbb{N} \to \mathbb{N}$, there exists a number $m_f \in \mathbb{N}$ with $\left|\bar{A}\restriction f(n)\right| \le \frac{n}{4}$ for all $n \ge m_f$.
- Let $\alpha, \omega\colon \mathbb{N} \to \mathbb{N}$ be defined via

  $$\begin{aligned}\alpha(n) &:= \min\{t \in \mathbb{N}\colon\ \ A[t]\restriction(n+1) = A\restriction(n+1)\},\\ \omega(n) &:= \min\{t \in \mathbb{N}\colon\ \ \Omega[t]\restriction(n/4) \ \ = \Omega\restriction(n/4) \ \ \}\end{aligned}$$

  for all $n \in \mathbb{N}$. We claim that there exists a number $m_1 \in \mathbb{N}$ such that for every $n \ge m_1$ we have $\alpha(n) \le \omega(n)$. This is clear since the Kolmogorov complexity of computably enumerable sets grows logarithmically in the length of its initial segments, while that of Ω grows linearly.
- Let $\theta\colon \mathbb{N} \to \mathbb{N}$ be defined via

  $$\theta(n) := \min\{t \in \mathbb{N}\colon\ \Omega_A[t]\restriction n = \Omega_A\restriction n\}$$

  for all $n \in \mathbb{N}$. We claim that there exists a number $m_2 \in \mathbb{N}$ such that for every $n \ge m_2$ we have $\omega(n) \le \theta(n)$. To see this, fix some $n \ge m_{\text{id}}$ and such that $\Omega_A\restriction n$ contains at least one 1; write t_n for $\theta(n)$.

  First, it is easy to see that if we let ℓ_n denote the maximal ℓ such that $(\Omega_A[t_n]\restriction n)(\ell) = 1$, then we must already have $A[t_n]\restriction \ell_n = A\restriction \ell_n$. But then, since we chose $n \ge m_{\text{id}}$, the sets A and $A[t_n]$ contain exactly the same at least $\ell_n - {}^{n}\!/_{4}$ elements less than ℓ_n. Consequently, we must have $\Omega[t_n]\restriction(\ell_n - {}^{n}\!/_{4}) = \Omega\restriction(\ell_n - {}^{n}\!/_{4})$. If we choose $m_2 \ge m_{\text{id}}$ such that for all $n \ge m_2$ we have $\ell_n \ge {}^{n}\!/_{2}$, we are done. To see that such an m_2 exists, we argue as follows: For a given $n \ge m_{\text{id}}$, $\Omega_A\restriction n$ contains at least the first $k \ge {}^{3}\!/_{4} \cdot n$ bits of Ω; and thus if the last 1 in $\Omega_A\restriction n$ occurs before position ${}^{n}\!/_{2}$, then the last ${}^{k}\!/_{3}$ bits of Ω must be 0. But this can occur only for finitely many k as Ω is Martin-Löf random.
- Now we show that Ω_A is not partial computably random. We recursively define the partial martingale $d\colon \subseteq\Sigma^* \to \mathbb{Q}_{\ge 0}$ as follows: Let $d(\lambda) := 1$, and suppose that $d(\sigma)$ is defined for some $\sigma \in \Sigma^*$. Let $t_\sigma \in \mathbb{N}$ be the earliest stage with $\Omega_A[t_\sigma]\restriction|\sigma| = \sigma$, if it exists. Then define $d(\sigma 0)$ and $d(\sigma 1)$ via

  $$d(\sigma 0) := \begin{cases}\frac{3}{2}\cdot d(\sigma) & \text{if } |\sigma| \notin A[t_\sigma],\\ d(\sigma) & \text{otherwise,}\end{cases} \qquad d(\sigma 1) := \begin{cases}\frac{1}{2}\cdot d(\sigma) & \text{if } |\sigma| \notin A[t_\sigma],\\ d(\sigma) & \text{otherwise.}\end{cases}$$

  It is clear that d is a partial computable martingale. We claim that d succeeds on Ω_A. To see this, let $\ell \ge \max(m_{\text{id}}, m_1, m_2)$ and assume that d receives input $\sigma := \Omega_A\restriction\ell$. Let $t_\sigma \in \mathbb{N}$ be the first stage such that $\Omega_A[t_\sigma]\restriction\ell = \sigma$. Then, by choice of ℓ and σ, we have $\alpha(\ell) \le \omega(\ell) \le \theta(\ell) = t_\sigma$, thus $A[t_\sigma]\restriction(\ell+1) = A\restriction(\ell+1)$. Thus, $\ell \notin A[t_\sigma]$ if and only if $\ell \notin A$, and by construction, for each of the infinitely many $\ell \notin A$, we have $d(\Omega_A\restriction(\ell+1)) = \frac{3}{2}\cdot d(\sigma)$. On the other hand, for all other ℓ, we have $d(\Omega_A\restriction(\ell+1)) = d(\sigma)$. Thus, $\lim_{n\to\infty} d(\Omega_A\restriction n) = \infty$, and Ω_A is not partial computably random.

- It remains to show that Ω_A is Schnorr random. For the sake of a contradiction, assume this is not the case; then, according to a result of Franklin and Stephan [6], there exists a computable martingale $d\colon \Sigma^* \to \mathbb{Q}_{\geq 0}$ and a computable increasing function $f\colon \mathbb{N} \to \mathbb{N}$ with $d(\Omega_A \restriction f(n)) \geq 2^n$ for infinitely many $n \in \mathbb{N}$. W.l.o.g. we can assume $d(\lambda) = 1$.
  Let $m := \max(m_f, m_1)$ and define the partial computable martingale $d'\colon \subseteq\Sigma^* \to \mathbb{Q}_{\geq 0}$ recursively as follows: Let $d'(\sigma) := 1$ for each σ with $|\sigma| \leq m$. Now suppose that $d'(\sigma)$ is defined for some $\sigma \in \Sigma^*$, write $\ell := |\sigma|$, and let $t_\sigma \in \mathbb{N}$ be the earliest stage with $\Omega[t_\sigma]\restriction\ell = \sigma$, if it exists. To define how d' bets on the next bit, imitate the betting of d on the $p_A[t_\sigma](\ell+1)$-th bit of $\Omega_A[t_\sigma]$. It is clear that d' is computable.
  We claim that d' succeeds on Ω. To see this, fix any $\ell \geq m$ and assume that d' receives input $\sigma := \Omega\restriction\ell$. Let $t_\sigma \in \mathbb{N}$ be the first stage such that $\Omega[t_\sigma]\restriction\ell = \sigma$. By choice of m, this implies $p_A[t_\sigma](\ell+1) = p_A(\ell+1)$. By construction, d' bets in the same way on $\Omega\restriction(\ell+1)$ as d does on $\Omega_A\restriction p_A(\ell+1)$.
  Define the function $g\colon \mathbb{N} \to \mathbb{N}$ by $g(n) := f(n) - \left|\bar{A}\restriction f(n)\right|$, and pick any of the, by assumption, infinitely many n with $d(\Omega_A\restriction f(n)) \geq 2^n$. Recall that $\Omega_A\restriction f(n)$ contains at most $n/4$ bits which belong to $\bar{A}$ and each of them can at most double the starting capital. This implies that d', which omits exactly these bets but imitates all the others, must still at least achieve capital $2^{3/4\cdot n}$ on the initial segment $\Omega\restriction g(n)$. This contradicts the well-known fact that Ω is partial computably random. ◀

6 Future research

We finish the article by highlighting possible future research directions:

The first two open questions concern the positions marked ◌$_1$ and ◌$_2$ in Figure 1; in both cases it is unknown whether any such numbers can exist. Note that proving the existence of ◌$_1$ would give a negative answer to the open question posed by Hölzl and Janicki [8] whether the left-computable numbers are covered by the union of the Martin-Löf randoms with the speedable and the nearly computable numbers; in particular, it would provide an alternative way to obtain a counterexample to the question of Merkle and Titov discussed in the introduction. Concerning ◌$_2$, one might be tempted to believe that such a number could be constructed by an argument similar to that used to prove Theorem 21 but using a maximal computably enumerable set A; such a set would still be dense simple by a result of Martin [11], but it would only contain isolated zeros. However, it can be shown that an Ω_A constructed in this way would still end up being superspeedable.

Next, besides Schnorr randomness, there are numerous other randomness notions weaker than Martin-Löf randomness, such as computable randomness or weak s-randomness (see, for instance, Downey and Hirschfeldt [4, Definitions 13.5.6 and 13.5.8]). It is natural to ask which of them are compatible with which of the notions of benign approximability discussed in this article.

Finally, in this field, many relevant properties of the involved objects are not computable; one might ask *how far* from computable they are. One framework in which questions of this type can be studied is the Weihrauch degrees, a tool to gauge the computational difficulty of mathematical tasks by thinking of them as black boxes that are given *instances* of a problem and that have to find one of its admissible *solutions*. This model then allows comparing the "computational power" of such black boxes with each other (for more details see, for instance, the survey by Brattka, Gherardi, and Pauly [2]). In the context of this article, we could for example ask for the Weihrauch degrees of the following non-computable tasks, with many variants imaginable:

- Given an approximation that witnesses speedability of some number as well as a desired constant, determine infinitely many stages at which the speed quotients of the given approximation beat the constant.
- For a speedable number, given an approximation to it as well as a desired speed constant ρ, determine another approximation of that number which achieves speed constant ρ in the limit superior.
- For an approximation witnessing regaining approximability of some number, determine the sequence of n's at which the approximation "catches up."

References

1 George Barmpalias. Personal communication, 2023.

2 Vasco Brattka, Guido Gherardi, and Arno Pauly. Weihrauch complexity in computable analysis. In Vasco Brattka and Peter Hertling, editors, *Handbook of Computability and Complexity in Analysis*, pages 367–417. Springer, 2021. doi:10.1007/978-3-030-59234-9_11.

3 Cristian S. Calude. *Information and Randomness: An Algorithmic Perspective*. Springer, 2nd edition, 2002. doi:10.1007/978-3-662-04978-5.

4 Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer, 2010. doi:10.1007/978-0-387-68441-3.

5 Rodney G. Downey and Evan J. Griffiths. Schnorr randomness. *Journal of Symbolic Logic*, 69(2):533–554, 2004. doi:10.2178/jsl/1082418542.

6 Johanna N. Y. Franklin and Frank Stephan. Schnorr trivial sets and truth-table reducibility. *Journal of Symbolic Logic*, 75(2):501–521, 2010. doi:10.2178/jsl/1268917492.

7 Peter Hertling, Rupert Hölzl, and Philip Janicki. Regainingly approximable numbers and sets. *Journal of Symbolic Logic*, 2024. Appeared online. doi:10.1017/jsl.2024.5.

8 Rupert Hölzl and Philip Janicki. Benign approximations and non-speedability, 2023. Preprint. doi:10.48550/arXiv.2303.11986.

9 Mathieu Hoyrup. Genericity of weakly computable objects. *Theory of Computing Systems*, 60(3):396–420, 2017. doi:10.1007/s00224-016-9737-6.

10 Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 3rd edition, 2008. doi:10.1007/978-0-387-49820-1.

11 Donald A. Martin. A theorem on hyperhypersimple sets. *Journal of Symbolic Logic*, 28(4):273–278, 1963. doi:10.2307/2271305.

12 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966. doi:10.1016/S0019-9958(66)80018-9.

13 Wolfgang Merkle and Ivan Titov. Speedable left-c.e. numbers. In Henning Fernau, editor, *Proceedings of the 15th International Computer Science Symposium in Russia*, Lecture Notes in Computer Science 12159, pages 303–313. Springer, 2020. doi:10.1007/978-3-030-50026-9_22.

14 André Nies. *Computability and Randomness*. Oxford University Press, 2009. doi:10.1093/acprof:oso/9780199230761.001.0001.

15 Piergiorgio Odifreddi. *Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers*. Elsevier, 1992.

16 Claus Peter Schnorr. *Zufälligkeit und Wahrscheinlichkeit*. Springer, 1971. doi:10.1007/BFb0112458.

17 Robert I. Soare. Cohesive sets and recursively enumerable Dedekind cuts. *Pacific Journal of Mathematics*, 31:215–231, 1969. doi:10.2140/pjm.1969.31.215.

18 Guohua Wu. Regular reals. *Mathematical Logic Quarterly*, 51(2):111–119, 2005. doi:10.1002/malq.200310129.

Fully-Adaptive Dynamic Connectivity of Square Intersection Graphs

Ivor van der Hoog
Technical University of Denmark, Lyngby, Denmark

André Nusser
CNRS, Inria, I3S, Université Côte d'Azur, France

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Frank Staals
Utrecht University, The Netherlands

Abstract

A classical problem in computational geometry and graph algorithms is: given a dynamic set $\mathcal{S}$ of geometric shapes in the plane, efficiently maintain the connectivity of the intersection graph of $\mathcal{S}$. Previous papers studied the setting where, before the updates, the data structure receives some parameter P. Then, updates could insert and delete disks as long as at all times the disks have a diameter that lies in a fixed range $[\frac{1}{P}, 1]$. As a consequence of that prerequisite, the aspect ratio ψ (i.e. the ratio between the largest and smallest diameter) of the disks would at all times satisfy $\psi \leq P$. The state-of-the-art for storing disks in a dynamic connectivity data structure is a data structure that uses $O(Pn)$ space and that has amortized $O(P \log^4 n)$ expected amortized update time. Connectivity queries between disks are supported in $O(\log n / \log \log n)$ time.

In the dynamic setting, one wishes for a more flexible data structure in which disks of any diameter may arrive and leave, independent of their diameter, changing the aspect ratio freely. Ideally, the aspect ratio should merely be part of the analysis. We restrict our attention to axis-aligned squares, and study fully-dynamic square intersection graph connectivity. Our result is fully-adaptive to the aspect ratio, spending time proportional to the current aspect ratio ψ, as opposed to some previously given maximum P. Our focus on squares allows us to simplify and streamline the connectivity pipeline from previous work. When n is the number of squares and ψ is the aspect ratio after insertion (or before deletion), our data structure answers connectivity queries in $O(\log n / \log \log n)$ time. We can update connectivity information in $O(\psi \log^4 n + \log^6 n)$ amortized time. We also improve space usage from $O(P \cdot n \log n)$ to $O(n \log^3 n \log \psi)$ – while generalizing to a fully-adaptive aspect ratio – which yields a space usage that is near-linear in n for any polynomially bounded ψ.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Computational geometry

Keywords and phrases Computational geometry, planar geometry, data structures, geometric intersection graphs, fully-dynamic algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.63

Funding *Ivor van der Hoog*: Received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.
André Nusser: Supported by the French government through the France 2030 investment plan managed by the National Research Agency (ANR), as part of the Initiative of Excellence of Université Côte d'Azur under reference number ANR-15-IDEX-01.
Eva Rotenberg: Supported by the Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis" and the Carlsberg Foundation Young Researcher Fellowship CF21-0302 "Graph Algorithms with Geometric Applications".

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 63; pp. 63:1–63:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Geometric intersection graphs are one of the most well-studied geometrically-flavoured graph classes: Their nodes are geometric shapes, and an edge between two such shapes exists if and only if they intersect. This makes the description complexity of a geometric intersection graph very compact; it is linear in the number of objects, while the underlying graph potentially has a quadratic number of edges. In this work, we consider square intersection graphs in the dynamic setting. Intersection graphs are one of the few examples of dynamic graphs where fully-dynamic insertion and deletion of vertices is motivated and interesting. Since a vertex can come or leave with $\Theta(n)$ edges, applying any existing edge-updatable dynamic graph algorithm in a blackbox manner would lead to $\Omega(n)$ update time. Yet, the geometric nature of these graphs often allows for sublinear or even polylogarithmic update times.

A classical problem in dynamic graph algorithms is dynamic connectivity. In this problem, we want to maintain a data structure under edge or node insertions and deletions that allows for fast queries that return whether a given pair of vertices is connected. Connectivity was one of the first problems to be studied in the dynamic setting dating back to the 80s [11, 25], and has received ample attention ever since. Dynamic connectivity in general graphs has been studied in many settings; randomised or deterministic, amortised or worst-case [11, 13, 29, 18, 24], and the partially dynamic incremental or decremental settings [26, 27, 28, 1]. Due to its fundamental nature, and its many applications, dynamic connectivity has also received much attention for simpler graph classes. Examples of such graph classes include trees [25, 3], planar graphs [10, 21], and graphs of bounded genus [9, 15].

Naturally, the dynamic connectivity problem also drew attention for the class of geometric intersection graphs. This setting is particularly interesting as a single node insertion can drastically change the number of connected components, as it can introduce a linear number of new edges. On the other hand, the geometric structure can be exploited in the data structures. The first result for geometric connectivity in geometric intersection graphs with update and query time independent of the object diameters is by Chan et al. [7] who presents a dynamic (Euclidean) disk intersection data structure with an update time of $O(n^{20/21+\varepsilon})$ and a query time of $O(n^{1/7+\varepsilon})$. This has recently been improved to $O(n^{7/8})$ amortized update time with constant query time [6]. As progress seemed difficult in this setting, the setting in which the disks in the data structure have restricted diameters was considered. For a fixed diameter range, where there is some value P given in advance and diameters have to be contained in an interval $[\frac{1}{P}, 1]$, Kaplan et al. [17] showed that there is a data structure with expected amortized $O(P^2 \log^{10} n)$ update time, query time $O(\log n / \log \log n)$, using $O(nP \log n)$ space. Recently, Kaplan et al. [16] improved this to $O(P \log^7 n)$ expected amortized update time with the same update time and $O(nP)$ space.

From disks to squares. While the above works are stated for Euclidean disks, we note that the approach in [16] can be combined with [30] to obtain a data structure that works for the simpler setting of connectivity between axis-aligned squares. The obtained update time is amortized $O(P \log^4 n)$. Disk intersection graphs are often motivated by communication networks where the disks are interpreted as some sort of transmission diameter. This is an idealization of a complicated physical process and actual ad-hoc communication networks do not correspond to perfectly circular disks [20]. Thus, it is reasonable to switch to a different metric for computational reasons while maintaining the core idea of the underlying problem. Recently, similar progress was made for computing a single-source shortest path tree in an intersection graph by assuming square regions instead of disks [19].

Table 1 Complexities are asymptotic. All update times are amortized. The query time for all approaches is $O(\log n/\log\log n)$. $\lambda_s(n)$ denotes the maximum length of a Davenport-Schinzel sequence of order s on n symbols.

Object	Aspect ratio	Update time	Space	Ref.
disks	fixed $[\frac{1}{P}, 1]$	$P \cdot \log^7 n \cdot \lambda_6(\log n)$ exp.	nP	[16]
disks	fixed $[\frac{1}{P}, 1]$	$P \cdot \log^4$ exp.	nP	[16]+[22]
squares	fixed $[\frac{1}{P}, 1]$	$P \cdot \log^4 n$	nP	[16]+[30]
squares	adaptive ψ	$\psi \log^4 n + \log^6 n$	$n \log^3 n \log \psi$	Thm 10

We study the dynamic connectivity problem where the input S is a set of (axis-aligned) squares, while being fully adaptive to their aspect ratio. We let ψ denote the adaptive aspect ratio. Formally, if S is the input before an update and S' is the input after an update we define $\psi = \max\{\frac{\max_{\sigma \in S} |\sigma|}{\min_{\sigma' \in S} |\sigma'|}, \frac{\max_{\sigma \in S'} |\sigma|}{\min_{\sigma' \in S'} |\sigma'|}\}$. Our date structure maintains connectivity between squares with $O(\psi \log^4 n + \log^6 n)$ update time, $O(\log n/\log\log n)$ query time, and using $O(n \log^3 n \log \psi)$ space, see Table 1 for a comparison. our approach only requires near-linear space while maintaining near-linear update time and polylogarithmic query time.

Implications of adaptivity and our reduced space usage. To understand the implications of the adaptivity of our new solution, consider the scenario where the set of squares starts with a square A with diameter 1 and B with diameter $\frac{1}{n}$. Now, suppose the sequence of updates first removes B, then inserts a sequence of n squares of diameter 1, and finally reinserts B. Previous work [16] would require as input the interval $[\frac{1}{n}, 1]$ and the promise that all updates only insert squares in this interval. The space usage and the total update time of [16] is quadratic. In our case, since for almost all updates, the aspect ratio is constant. Moreover, the space usage and total update time is near-linear.

2 Problem statement and technical overview

Let $\mathcal{S} \subset \mathbb{R}^2$ be a set of axis-aligned squares. The intersection graph $G[\mathcal{S}]$ is the graph with vertex set $\mathcal{S}$ and with an edge between squares $\sigma, \sigma' \in \mathcal{S}$ whenever they intersect. We say that two squares σ, σ' are *connected* if there exists a path between their corresponding vertices in $G[\mathcal{S}]$. The set $\mathcal{S}$ is a fully dynamic set subject to (adversarial) insertions and deletions of squares. We wish to maintain $\mathcal{S}$ in a data structure supporting *connectivity queries* between squares in $\mathcal{S}$. We denote the diameter of square $\sigma \in \mathcal{S}$ by $|\sigma|$. We consider three settings with different restrictions on the square diameters in $\mathcal{S}$:

- The *fixed diameter range* setting. Here, the input specifies some P and each $\sigma \in \mathcal{S}$ has a diameter in $[\frac{1}{P}, 1]$ for some fixed P.
- The *bounded aspect ratio* setting. Here, the input specifies some P and at all times, for all $\sigma, \sigma' \in \mathcal{S}$ we have $\frac{|\sigma|}{|\sigma'|} \leq P$.
- The *adaptive aspect ratio* ψ setting in which arbitrary insertions and deletions may occur. Let S be the set of squares before an update and S' be the set of squares after an update. We define $\psi = \max\{\frac{\max_{\sigma \in S} |\sigma|}{\min_{\sigma' \in S} |\sigma'|}, \frac{\max_{\sigma \in S'} |\sigma|}{\min_{\sigma' \in S'} |\sigma'|}\}$ the aspect ratio relevant for the update.

We measure the algorithmic complexity in $n := |\mathcal{S}|$ and ψ, where n is the present size of the dynamic set $\mathcal{S}$. At all times, we maintain some minimal axis-aligned square F that contains $\mathcal{S}$, and the coordinates of F are powers of 2.

Results. The data structure of Kaplan et al. [16], when adapted to axis-aligned squares by applying Range Trees [30], can store a dynamic set of axis-aligned squares as follows: The input specifies some value P and all squares have fixed diameter range with ratio P. Their solution uses $O(nP)$ space and supports updates to $\mathcal{S}$ in $O(P \log^4 n)$ amortized time (Table 1). They answer connectivity queries in $O(\log n / \log \log n)$ worst-case time.

There are several reasons why [17] uses $O(nP)$ space and why their update bound cannot depend on the adaptive ψ instead of P (which we discuss further down). We present an adaption of their work that relies on the fact that $\mathcal{S}$ is a set of axis-aligned squares. Under this assumption, we adapt their data structure to work for adaptive ψ. We improve the space usage to near linear in n and $\log \psi$. In full generality, we also improve the performance: allowing for update times proportional to the density around the update σ. More concretely, we parameterize the runtime by the size of two sets $\mathcal{C}(\sigma)$ and $\mathcal{P}(\sigma)$. Intuitively, these sets contain squares in σ or squares around σ. These sets have size at most $O(\min\{\psi, n\})$.

Our result is a technical contribution, that examines and refines the data structure in [16] in the special case where $\mathcal{S}$ is a set of axis-aligned squares. To detail our contribution, we now present a technical overview, where we reference several concepts whose formal definitions are presented in their respective sections. The core component is a quadtree that stores $\mathcal{S}$.

The existing pipeline. We can describe the data structure of [16] (adapted to squares) on a high-level: They construct a quadtree $H(\mathcal{S})$ in which quadtree cells may store squares in $\mathcal{S}$. For any quadtree cell C, we denote by $\pi(C)$ the set of squares stored in C. A crucial definition is the concept of a *perimeter*. For a square σ, its perimeter $\mathbb{P}^*(\sigma, P)$ is intuitively a ring of $\Theta(P)$ quadtree cells of size at least $\frac{1}{4P}$ that are sufficiently close to the boundary of σ. Using this concept, their data structure is a pipeline of five components (Fig 1):

1. The set $\mathcal{S}$ gets stored in a compressed quadtree $H(\mathcal{S})$, such that for every $\sigma \in \mathcal{S}$, the quadtree contains $\mathbb{P}^*(\sigma, P)$.[1] This quadtree has $\Theta(P \cdot n)$ cells. For each $\sigma \in \mathcal{S}$, its storing cell is the maximal quadtree cell contained in σ.
2. They store all (maximal) quadtree cells contained in some $\sigma \in \mathcal{S}$ in a special ancestor data structure. We leave out the details of this structure, as we show that it suffices to use the well-studied marked-ancestor data structure by Alstrup, and Husfeldt, Rauhe [2].
3. For each square $\sigma \in \mathcal{S}$ with storing cell C_σ, for each quadtree cell $C_2 \in \mathbb{P}^*(\sigma, P)$, they store a square intersection data structure (a range tree). This data structure stores the squares $R \subset \pi(C_\sigma)$ that have C_2 in their perimeter (i.e. $R = \{\gamma \in \pi(C_\sigma) \mid C_2 \in \mathbb{P}^*(\sigma, P)\}$).
4. For each square σ with storing cell C_σ, for each quadtree cell $C_2 \in \mathbb{P}^*(\sigma, P)$, they store a maximal bichromatic matching (MBM) in the graph $G[R \cup B]$ with $B = \pi(C_2)$ with $R = \{\gamma \in \pi(C_\sigma) \mid C_2 \in \mathbb{P}^*(\sigma, P)\}$.
5. They store a *proxy graph* over the quadtree in the dynamic connectivity data structure by Holm, Lichtenberg, and Thorup (HLT) [14]. This graph contains an edge between two cells C_1, C_2 if and only if their maximal bichromatic matching is not empty.

They subsequently support connectivity queries for a query (σ, ρ) as follows. Given σ, obtain a pointer to its storing cell C_σ. Using their ancestor data structure, they obtain the largest ancestor C_a that is contained in a square $\sigma^* \in \mathcal{S}$. Let C^* be its storing cell. Doing the same procedure for ρ gives a cell R^*. They show that (σ, ρ) are connected in $G[\mathcal{S}]$ if and only if (C^*, R^*) are connected in the proxy graph; which they test in $O(\log n / \log \log n)$ time.

[1] Whilst originally their approach is a forest of quadtrees, we note that since each root of the forest is disjoint, the whole solution can be stored as a quadtree.

Why the space usage is high, and update times are not adaptive. For every $\sigma \in \mathcal{S}$, this pipeline creates a set of $\Theta(P)$ quadtree cells with sizes in $[\frac{1}{4P}, 1]$ which we denote by $\mathbb{P}^*(\sigma, P)$. For each quadtree cell $C_2 \in \mathbb{P}^*(\sigma, P)$, σ gets stored in a square intersection data structure, even if the quadtree cell C_2 is empty and/or has no other squares nearby. This approach makes it require a lot of space. Moreover, this approach fails when the aspect ratio becomes adaptive: suppose that $\mathcal{S}$ has $n-1$ squares with diameter $\frac{1}{P}$ and one unit square. We replace the unit square with a square of diameter $\frac{1}{P^2}$. The aspect ratio remains bounded by P. However (after rescaling the plane by a factor $\frac{1}{P}$) the quadtree no longer contains for every $\sigma \in \mathcal{S}$ the set $\mathbb{P}^*(\sigma, P)$ as (after rescaling the plane) it only contains quadtree cells of size 1. Reconstructing these requires $\Omega(Pn)$ time.

Our adaption. We improve this pipeline in several ways based on a few key insights: First, we revisit the definition of *perimeter*; presenting a new definition $\mathcal{P}(\sigma)$ which intuitively contains only cells in $\mathbb{P}^*(\sigma, \psi)$ that store at least one square. Since we only store data structures on cells that store at least one square, we save space and allow ψ to become fully adaptive. This introduces a new challenge, as we need to work with significantly fewer precomputed inormation. Concretely, we do the following (Figure 2):

1. We define a new type of quadtree $T(\mathcal{S})$ that uses only $O(n \log \psi)$ quadtree cells.
2. We replace their custom ancestor data structure by the well-studied Marked Ancestor Tree (MAT), simplifying the data structure.
3. For squares metric we create a new data structure that has deterministic guarantees and that avoids storing many copies.
4. For each square $\sigma \in \mathcal{S}$ with storing cell C_σ, for each quadtree cell C_2 in our new perimeter $\mathcal{P}(\sigma)$, we store a Maximal Bichromatic Matching (MBM*) in a graph $G[R \cup B]$. Using our new definition of perimeter, we define $R \leftarrow \{\gamma \in \pi(C_\sigma) \mid C_2 \in \mathcal{P}(\sigma)\}$ and $B \leftarrow \pi(C_2)$. We present a new algorithm to maintain this Maximal Bichromatic Matching.
5. Finally, we use HLT [14] on a proxy graph with an edge for every non empty MBM*.

We go through our data structures one by one in the order indicated in Figure 2.

3 Storing disks in quadtrees

Recall that F is a (dynamic) square bounding box of $\mathcal{S}$. By construction, the side length of F is 2^ω for some integer ω. We define the square F to be a *quadtree cell* and we recursively define quadtree cells to be any square obtained by *splitting* a cell into four equally sized closed squares. A *quadtree* T on F is any hierarchical decomposition obtained by recursively splitting cells. This hierarchical decomposition has a natural representation as a tree: the root is the cell F and every cell has either 0 or 4 children depending on whether it was split. We denote by $\mathbb{F}$ the (infinite) set of cells that are obtained by recursively splitting all cells, starting from F. We say that a cell $C \in \mathbb{F}$ is at *level* ℓ whenever its side length is 2^ℓ. We treat any quadtree T as a set of cells, i.e., $T \subset \mathbb{F}$.

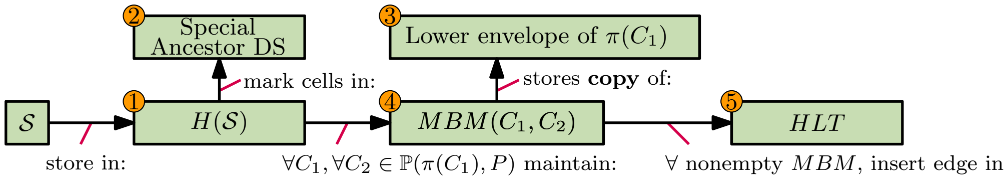

2
Special Ancestor DS
3
Lower envelope of $\pi(C_1)$
mark cells in:
stores **copy** of:
$\mathcal{S}$
1
$H(\mathcal{S})$
4
$MBM(C_1, C_2)$
5
HLT
store in:
$\forall C_1, \forall C_2 \in \mathbb{P}(\pi(C_1), P)$ maintain:
$\forall$ nonempty MBM, insert edge in

Figure 1 The five-component pipeline by Kaplan et al. where the arrows indicate dependencies.

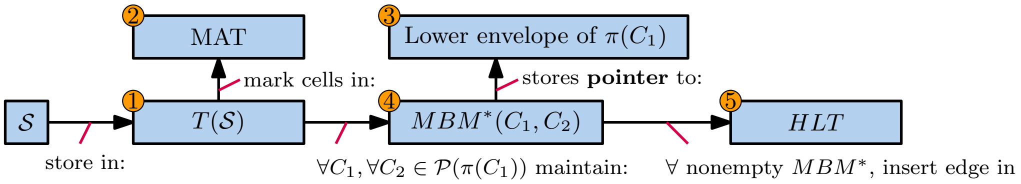

2
MAT
3
Lower envelope of $\pi(C_1)$
mark cells in:
stores **pointer** to:
$\mathcal{S}$
1
$T(\mathcal{S})$
4
$MBM^*(C_1, C_2)$
5
HLT
store in:
$\forall C_1, \forall C_2 \in \mathcal{P}(\pi(C_1))$ maintain:
$\forall$ nonempty MBM^*, insert edge in

Figure 2 Our five-component pipeline where the arrows indicate dependencies.

Löffler, Simons, and Strash [23] use quadtrees to store arbitrary squares (or disks). Let F be fixed and σ be some square with center s. The unique *storing cell* $C_\sigma \in \mathbb{F}$ is a largest cell in $\mathbb{F}$ that contains the center s and that itself is contained in σ (if s lies at the intersection of multiple largest contained cells, we define the bottom left cell to be C_σ). Löffler, Simons and Strash subsequently say that C_σ *stores* σ. For a cell $C \in \mathbb{F}$, we denote by $\pi(C)$ all square of $\mathcal{S}$ stored in C. We will define five different cell sets to define our quadtree storing $\mathcal{S}$.

Quadtree cell sets. We assume that we have some bounding box F (which induces the set $\mathbb{F}$) and some set of storing cells in $\mathbb{F}$. We subsequently define two types of subsets of $\mathbb{F}$:

- definitions that originate from [16] and depend on some $P \in \mathbb{R}$ (blackboard font),
- and new definitions depending on only the storing cells (calligraphic font).

Quadtree cells C, C' are *neighboring* whenever they are not descendants of one another and intersect in their boundary. For any property, a quadtree cell C in $\mathbb{F}$ is *maximal* if there does not exist an ancestor of C in $\mathbb{F}$ with the same property.

Let $\sigma \in \mathcal{S}$ have a storing cell C_σ. We define (Figure 3):

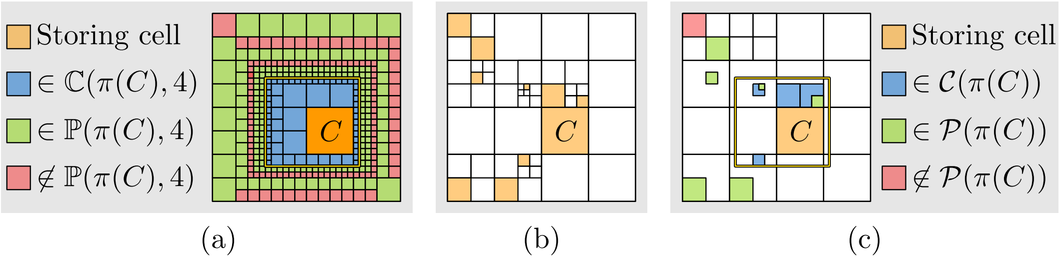
- $\mathcal{N}(\sigma) \subset \mathbb{F}$ as the cells of size $|C_\sigma|$ neighboring C_σ or a neighbor of C_σ.
- $\mathbb{C}^*(\sigma, P) \subset \mathbb{F}$ as the maximal cells $C' \in \mathbb{F}$ with $C' \subset \sigma$ and $|C| \in [\frac{1}{4P}, 1]$.
- $\mathcal{C}(\sigma) \subset \mathbb{F}$ as the maximal cells $C' \in \mathbb{F}$ with $C' \subset \sigma$ that contain at least one storing cell.
- $\mathbb{P}^*(\sigma, P) \subset \mathbb{F}$ as the *perimeter* of σ. These are all $C' \in \mathbb{F}$ contained in a cell in $\mathcal{N}(\sigma)$ with $|C'| \in [\frac{1}{4P}, 1]$ with the additional property that there exists a square $\rho \subset \mathbb{R}^2$ where C' would be the storing cell of ρ if $\rho \in S$, and, ρ intersects the boundary of σ.
- $\mathcal{P}(\sigma) \subset \mathbb{F}$ as all *storing cells* with diameter at most $|\sigma|$ that, when scaled around their center by a factor 5, intersect *the boundary* σ. Note that these may be contained in σ.

For $R \subseteq \mathcal{S}$, we define $\mathcal{N}(R)$ to be the union of all $\mathcal{N}(\sigma)$ with $\sigma \in R$. All other sets (e.g., $\mathcal{P}(R)$) are defined analogously. Let $\ell_{\min}$ (resp. $\ell_{\max}$) be the smallest (resp. largest) level that contains any cell in any of the five sets. Per definition, $\ell_{\max} - \ell_{\min} \in O(\log \psi)$.

▶ **Lemma 1** (Lemma 4.2 in [16]). *For any $\sigma \in \mathcal{S}$, if regions are disks under an L_p metric with a diameter in $[\frac{1}{4P}, 1]$ then:* $|\mathbb{C}^*(\sigma, P)| \in O(P)$ and $|\mathbb{P}^*(\sigma, P)| \in O(P)$.

Compressed quadtrees. Denote by $X \subset \mathbb{F}$ some set of cells. Denote by T_X the minimal quadtree over some bounding box F that contains all cells in X. The size of T_X can be arbitrarily large, even when $|X|$ is constant. To reduce quadtree space complexity, a quadtree may be *compressed* [12]. An α-compressed quadtree (for some variable $\alpha \geq 1$) is defined as follows: let C be a quadtree cell in T_X and C_α be the smallest descendant of C such that (1) $|C| \geq 2^\alpha |C_\alpha|$ and (2) all cells in X that are contained in C are also contained in C_α. Then C has not 4 children, but only C_α as its child. Given some constant α, every quadtree has a unique maximally compressed equivalent that has size linear in $|X|$ [12]. Given the above definitions, we want to mention two different quadtrees that store $\mathcal{S}$:

- [23] defines $L(\mathcal{S})$ as the compressed quadtree storing $\mathcal{N}(\mathcal{S})$.
- [16] defines $H(\mathcal{S})$ as the compressed quadtree storing $\mathcal{N}(\mathcal{S})$, $\mathbb{C}^*(\mathcal{S})$ and $\mathbb{P}^*(\mathcal{S})$.


Storing cell
$\in \mathbb{C}(\pi(C), 4)$
$\in \mathbb{P}(\pi(C), 4)$
$\notin \mathbb{P}(\pi(C), 4)$
C
C
C
Storing cell
$\in \mathcal{C}(\pi(C))$
$\in \mathcal{P}(\pi(C))$
$\notin \mathcal{P}(\pi(C))$

(a) (b) (c)

Figure 3 (a) We show for a square σ its storing cell in orange. We set $P = \psi = 2$ and show our sets. Many cells in $\mathbb{C}^*(\pi(C), 8)$ are also in $\mathbb{P}^*(\pi(C), 8)$. (b) The minimal quadtree that contains a set of storing cells. (c) Given the quadtree with storing cells, we illustrate our sets. Red cells are storing cells that occur in neither $\mathcal{C}(\pi(C))$ nor $\mathcal{P}(\pi(C))$.

Our quadtree. We define our quadtree $T(\mathcal{S})$ as the compressed quadtree storing $\mathcal{N}(\mathcal{S})$, where cells in $\mathcal{C}(\mathcal{S})$ are uncompressed. Note that any quadtree that contains $\mathcal{N}(\mathcal{S})$, also contains the cells in $\mathcal{C}(\mathcal{S})$. The key difference between $L(\mathcal{S})$ and $T(\mathcal{S})$ is that we decompress the cells in $\mathcal{C}(\mathcal{S})$, adding them to memory (i.e., we treat these cells as storing cells in the quadtree). Since these cells are uncompressed, this structure uses more space than the $O(n)$ cells in $\mathcal{N}(\mathcal{S})$. If we view quadtrees as a collection of (uncompressed) cells, then $L(\mathcal{S}) \subset T(\mathcal{S}) \subset H(\mathcal{S})$. By Lemma 1, Kaplan et al. [16] prove that $|\mathbb{C}^*(\mathcal{S}, P)|, |\mathbb{P}^*(\mathcal{S}, P)| \in O(Pn)$. It would be easy to show that $|\mathcal{C}(\mathcal{S})|, |\mathcal{P}(\mathcal{S})| \in O(n\psi)$. But through clever counting, we prove that storing $T(\mathcal{S})$ uses only $O(n \log \psi)$ space instead (Theorem 4).

3.1 Space complexity of the quadtree

We upper bound the size of $\mathcal{N}(\mathcal{S})$ and $\mathcal{C}(\mathcal{S})$ (and thus the size of $T(\mathcal{S})$).

▶ **Observation 2.** *There are $O(n)$ cells in $\mathcal{N}(\mathcal{S})$.*

▶ **Lemma 3.** *Let C be a storing cell. Denote by Z any cell, such that there could exist a σ stored in Z where an ancestor of C lies in $\mathcal{C}(\sigma)$. There are at most $O(\log \psi)$ such cells and we can report them in $O(\log \psi \log n)$ time.*

Proof. Let $|C| = 2^\ell$ (i.e., C is at level ℓ). By definition, Z is in a level $j \geq \ell$. Fix a level j. For any cell Z at level j, $\mathcal{C}(\pi(Z))$ contains an ancestor of C only if a square in $\pi(Z)$ intersects (or contains) C_j (the ancestor of C at level j). As the diameter of squares in $\pi(Z)$ is at most a factor 5 larger than the diameter of Z, there are at most $O(1)$ cells at level j that *could* store a square ρ that intersects C_j (the neighbors of C_j, their neighbors and possibly their neighbors). We can find these cells in $O(\log n)$ time by doing a point location in each cell for the level j. The fact that per definition of ψ, all storing cells and all cells in $\mathcal{C}(\mathcal{S})$ lie in a range of $O(\log \psi)$ levels concludes the proof. ◀

▶ **Theorem 4.** *At all times, the compressed quadtree $T(\mathcal{S})$ uses $O(n \log \psi)$ space.*

Proof. Since $T(\mathcal{S})$ is the quadtree that stores $\mathcal{N}(S)$ and $\mathcal{C}(S)$, and compressed quadtrees have linear space in the number of uncompressed cells, Observation 2 and Lemmas 3 immediately imply the theorem. ◀

We additionally upper bound the size of two more quadtree cell types:

▶ **Lemma 5.** *Let $\mathcal{S}$ be a set of squares with aspect ratio ψ. For all $\sigma \in \mathcal{S}$ with storing cell C_σ there are at most $O(\psi)$ cells in $\mathcal{C}(\sigma)$ and $\mathcal{P}(\sigma)$ and $O(\psi^2)$ cells in $\mathcal{P}(\pi(C_\sigma))$.*

Proof. Consider any set of squares S. We rescale the plane such that the diameter of squares in $\mathcal{S}$ lies in $[\frac{1}{\psi}, 1]$. We apply Lemma 1 to conclude that $|\mathbb{C}(\sigma, \psi)|, |\mathbb{P}(\sigma, \psi)| \in O(\psi)$. Note that the smallest storing cell in $T(\mathcal{S})$ then has size $\frac{1}{4\psi}$. Having rescaled, $\mathcal{C}(\sigma) \subseteq \mathbb{C}^*(\sigma, \phi)$.

Suppose that after rescaling there is a cell $C \in \mathcal{P}(\sigma)$ that is not in $\mathbb{P}^*(\sigma, \psi)$. Then C, scaled by a factor 5, intersects the boundary of σ. Yet if $C \notin \mathbb{P}^*(\sigma, \psi)$ then C cannot store any square ρ that intersects σ. Denote by C' a neighbor of C of size $2|C|$ that lies closer to the center of σ. It must be that $C' \in \mathbb{P}^*(\sigma, \psi)$ (indeed, we can construct a square with diameter $4|C|$ stored in C' that intersects σ). This way, each cell in $\mathbb{P}^*(\sigma, \psi)$ can get charged by at most $O(1)$ cells $C' \in \mathcal{P}(\sigma)$ where $C' \notin \mathbb{P}^*(\sigma, \psi)$. Thus, $|\mathcal{P}(\sigma)| \in O(\psi)$. The upper bound on $|\mathcal{P}(\pi(C_\sigma))|$ follows from the standard packing argument. ◀

▶ **Lemma 6.** *Let C be a storing cell. Denote by Z any cell, such that there could exist a σ stored in Z with $C \in \mathcal{P}(\sigma)$. We can report all $O(\log \psi)$ such cells in $O(\log \psi \log n)$ time.*

Proof. Let $|C| = 2^\ell$ (i.e., C is at level ℓ in the quadtree). By definition, every quadtree cell Z of the lemma statement is stored at a level $j \geq \ell$. Fix a level $j \geq \ell$ and let C_j be the ancestor of C at level j. If for any cell Z at level j, $C \in \mathcal{P}(\pi(Z))$ then it must be that the cells Z and C_j (when both are scaled around their center by a factor 5) intersect. There are at most $O(1)$ such cells Z at level j for which this can be true. We can find these cells at level j in $O(\log n)$ time by performing $O(1)$ point locations in the quadtree (querying a neighborhood of 25×25 around C_j). The fact that all cells in $\mathcal{P}(\mathcal{S})$ lie in a range of $O(\log \psi)$ levels concludes the proof. ◀

4 Maintaining and navigating quadtrees

A compressed quadtree T_X that stores a set X of quadtree cells can be dynamically maintained in $O(\log |X|)$ time per insertion and deletion [12]. Moreover, leaf location queries are supported in $O(\log |X|)$ time, which take as input some point $q \in \mathbb{R}^2$ and output the leaf of T_X that contains q. By Theorem 4, $|X| = O(n \log \psi)$ in our setting. Since we assume that $\psi \in O(n^c)$, see Section 2, we can say that our insertion, deletion and point location operations in the compressed quadtree take $O(\log n)$ time. Compressed quadtrees additionally support level locations where for any query point $q \in \mathbb{R}^2$ and level ℓ, the output is the quadtree cell at level ℓ that contains q; this can be used to dynamically maintain for all $\sigma \in \mathcal{S}$ the set $\mathcal{N}(\sigma)$ in our quadtree in $O(\log n)$ time per update in $\mathcal{S}$ [5].

We maintain $L(\mathcal{S})$ in $O(\log n)$ time per update to $\mathcal{S}$, while supporting point location queries. We maintain in $O(\log n)$ time per update in $\mathcal{S}$ the values $d_{\max}$ and $d_{\min}$ that denote the maximal and minimal diameter in $\mathcal{S}$ respectively. We apply 3 more data structures:

Marked Ancestor Trees (MAT). Alstrup, Husfeldt, and Rauhe [2] introduce marked-ancestor trees. Let T be a dynamic tree. Each node in T is either marked or unmarked. Given a node $v \in T$, the MAT supports changing the mark of v or updating T in $O(\log \log n)$ time. Additionally, given a node $v \in T$, one can find the lowest/highest marked node on the path from v to the root in $O(\log n / \log \log n)$ time. We augment our quadtree with a MAT.

Orthogonal range trees. Willard and Lueker [30] show a data structure to store a set of n squares using $O(n \log n)$ space. Given a query rectangle ρ, it can report an input square contained in ρ in $O(\log^4 n)$ time, if such a square exists. Given a query square ρ it can report the number of squares that contain ρ in $O(\log^4 n)$ time. We implement the range tree using general balanced trees [4], so that we can support updates in amortized $O(\log^4 n)$ time.

Segment trees. Segment trees [8] store a set of n horizontal segments using $O(n \log n)$ space, so that for a vertical query segment Q we obtain all k input segments that intersect Q in $O(\log^2 n + k)$ time. The data structure can again be made dynamic, supporting updates in $O(\log^2 n)$ amortized time. For any σ, we store the horizontal sides of C_σ (scaled by a factor 5) in such an orthogonal intersection data structure. Furthermore, we create a second such a data structure storing all vertical sides.

▶ **Theorem 7.** *Let $\mathcal{S}$ be a set of axis-aligned squares. We augment $T(\mathcal{S})$ with an $O(n \log n)$-size data structure using $O(n \log n)$ space, supporting inserting/deleting a square σ in $O(|\mathcal{C}(\sigma)| \cdot \log^4 n + \log^6 n)$ time, and*

- *all cells in $\mathcal{C}(\mathcal{S})$ are marked in our marked-ancestor tree;*
- *for any query square γ, we can obtain $\mathcal{P}(\gamma)$ in $O(\log^2 n + |\mathcal{P}(\gamma)|)$ time.*
- *for any query cell C, we obtain the set $\mathcal{Z}(C) := \{Z \mid C \in \mathcal{P}(\pi(Z))\}$ in $O(\log^5 n)$ time.*

Proof. By Theorem 4, our quadtree requires $O(n \log \psi)$ space. Using the standard operations on compressed quadtrees, we can maintain $\mathcal{N}(\mathcal{S})$ in $O(\log n)$ time per update. What remains for quadtree maintenance is to identify, decompress and mark all cells in $\mathcal{C}(\mathcal{S})$.

Maintaining $\mathcal{C}(\mathcal{S})$. Every cell $C \in \mathbb{F}$ has a counter that counts for how many $\sigma \in \mathcal{S}$, $C \in \mathcal{C}(\sigma)$. Whenever the counter is zero, we do not store it explicitly. Otherwise, $C \in \mathcal{C}(\mathcal{S})$ and we need to make sure that C is decompressed and marked. For each update of a square σ with storing cell C_σ, there are two types of counter updates:

1. updating counters of $C \in \mathcal{C}(\sigma)$, and
2. updating the counters of C_σ and its ancestors.

We start with the first case. Instead of increasing counters, we do something slightly stronger as we can report all of $\mathcal{C}(\sigma)$. By definition, $C_\sigma \in \mathcal{C}(\sigma)$ and we add it to our output. We split σ into eight rectangles that are bounded by σ and the boundary of C_σ, see Figure 4. We process each rectangle separately. Consider such a rectangle R. We query our orthogonal range tree to report a storing cell C_1 in R in $O(\log^4 n)$ time. Given C_1, we walk in $O(\log \psi)$ time up the quadtree to find its largest ancestor C_1' that is still contained in σ. By definition, $C_1' \in \mathcal{C}(\sigma)$ and we add it to our output. Subsequently, we partition R into nine rectangles that are bounded by the boundaries of C_1' and recurse. For each cell in $\mathcal{C}(\sigma)$ we perform eight orthogonal range queries. For each range query, we either conclude that the range contains no cells in $\mathcal{C}(\sigma)$, or we identify at least one cell in $\mathcal{C}(\sigma)$. As we recurse on rectangles that are bounded by cell boundaries and we explore all of R, we find all cells of $\mathcal{C}(\sigma)$ in $O(|\mathcal{C}(\sigma)| \cdot \log^2 n)$ time. As we find them, we may adjust their counters.

Now onto the second case, where we simply recompute all counters from scratch. There are at most $O(\log \psi)$ ancestors $C_\sigma, C_1, \ldots C_k$ of C_σ that may be contained in a square in $\mathcal{S}$. For each of these, we recompute their counters from scratch. Fix an ancestor C_i with parent C_{i+1}. By Lemma 3, there are at most $O(\log \psi)$ cells Z such that Z could store a square γ with $C_i \in \mathcal{C}(\gamma)$. We obtain all such Z in $O(\log \psi \log n)$ time and iterate over each of them. For a fixed Z, we use the range tree to count how many $\gamma \in \pi(Z)$ contain C_i in $O(\log^4 n)$ time. We then count how many $\gamma \in \pi(Z)$ contain C_{i+1}. The difference between these counts is the number of squares $\gamma^* \in \pi(Z)$ for which $C_i \in \mathcal{C}(\gamma^*)$. We compute and sum all these numbers to recompute the count of C_i. It follows from the fact that $O(\log \psi) \subset O(\log n)$ that we can maintain $\mathcal{C}(\mathcal{S})$ in $O(|\mathcal{C}(\sigma)| \cdot \log^2 n + \log^6 n)$ time.

Querying for $\mathcal{P}(\gamma)$. We show how to obtain for any query squares γ the set $\mathcal{P}(\gamma)$ in $O(\log^2 n + |\mathcal{P}(\gamma)|)$ time. For each storing cell C, we consider the cell C^* that is C scaled by a factor 5 around its center. We store each of the boundary segments of C^* in our Segment Tree. There is exactly one storing cell per square in $\mathcal{S}$, so maintaining the Segment Tree intersection data structure takes $O(\log^2 n)$ time per update and uses $O(n \log n)$ space. For any query square γ, we have $C \in \mathcal{P}(\gamma)$ if and only if one of the boundary segments of γ intersects one of the boundary segments of C^*. Thus, we can immediately use the intersection data structure to compute $\mathcal{P}(\gamma)$ in $O(\log^2 n + |\mathcal{P}(\gamma)|)$ time, as each cell in $\mathcal{P}(\gamma)$ is reported at most a constant number of times.

Querying for $\mathcal{Z}(C)$. Denote by Z any cell, such that there could exist a γ stored in Z with $C \in \mathcal{P}(\gamma)$. By Lemma 6, we can report all at $O(\log \psi)$ such cells in $O(\log \psi \log n)$ time. For every such Z, we conceptually rotate the plane such that Z lies above C. Let C^* be the cell C increased by a factor 5 around its center. Any square ρ in $\pi(Z)$ intersects C^* in its boundary if and only if one of two conditions hold: ρ contains the top left endpoint of C^* but not the bottom left endpoint, or ρ contains the top right endpoint of C^* but not the bottom right endpoint. We select the top right endpoint of C^* and we count how many squares in $\pi(Z)$ contain the top right endpoint in $O(\log^4 n)$ time. We do the same for the bottom right endpoint. If the counts differ, there is at least one square in $\pi(Z)$ that intersects C^* in its boundary and thus $C \in \mathcal{P}(\pi(Z))$. Doing this for all $O(\log \psi)$ levels takes $O(\log^5 n)$ time. ◀

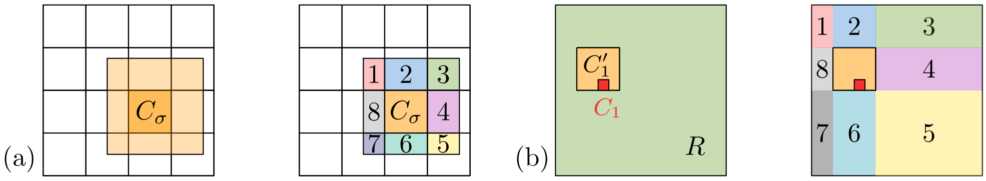


Figure 4 (a) Given a storing cell C_σ, we partition σ into nine rectangles (one being C_σ). (b) For each rectangle R, we do a range query to find a storing cell C_1 (if it exists). For the largest ancestor $C_1' \subset \sigma$ of C_1, we partition R into nine rectangles once again and recurse.

5 Specific square intersection data structures

In this section we develop a solution for the following data structure problem: Let $\mathcal{R}$ be a set of m squares. Let $R_1, \ldots, R_k$, be k subsets of $\mathcal{R}$ that we refer to as *conflict sets* and let $\ell \leq k$ be the maximum number of conflict sets that any square from $\mathcal{R}$ appears in. We want to store $\mathcal{R}$ and all the conflict sets $R_1, \ldots, R_k$ in a data structure that has size near linear in m and $z = \sum_i |R_i|$, and support the following operations in the following time:

Insert(ρ), ($O(l \cdot \log^3 m)$ time): Insert a square ρ into $\mathcal{R}$.

Delete(ρ), ($O(l \cdot \log^3 m)$ time): Delete a square ρ from $\mathcal{R}$ and every R_i that it occurs in.

Insert(ρ, R_i), ($O(\log^3 m)$ time): Insert a square ρ in the conflict set R_i. If $R_i = \emptyset$, create a new conflict set.

Delete(ρ, R_i), ($O(\log^3 m)$ time): Delete a square ρ from the conflict set R_i.

Query(σ, R_i, C), ($O(\log^3 m)$ time): Given a query σ whose center lies below all centers of all squares in $\mathcal{R}$, and a horizontal line segment C below σ, return (if it exists) a square $\rho \in \mathcal{R}$ that intersects σ, but is not in the conflict set R_i, and that does not contain C.

In Section 6 we solve this problem as follows: we map every square $\rho = [\ell_\rho, r_\rho] \times [b_\rho, t_\rho] \in \mathcal{R}$ to a point $p_\rho = (b_\rho, \ell_\rho, r_\rho)$ in $\mathbb{R}^3$, and store these points in a 3D-range tree T augmented for range counting queries [8]. Hence, every third-level subtree T_ν stores the number of points m_ν in T_ν. Furthermore, for each such subtree, and each conflict set R_i, consider the subset of points stored in the leaves of T_ν for which the corresponding square appears in R_i. If this set is non-empty then node ν also stores the size $m_{\nu,i} = |\{p_\rho \mid p_\rho \in T_\nu \wedge \rho \in R_i\}|$ of this set. Furthermore, we maintain a bipartite graph between the squares in $\mathcal{R}$ and the conflict sets R_i, so that given a square $\rho \in \mathcal{R}$ we can find the ℓ conflict sets it appears in in $O(\ell)$ time. We implement all trees using general balanced trees [4] so that we can perform updates efficiently. A 3D-range tree uses $O(m \log^2 m)$ space. Each square in the multiset $\bigcup_i R_i$ contributes to $O(\log^3 m)$ nodes of T, and hence the entire structure uses at most $O((m+z)\log^3 m)$ space (Lemma 8). In Section 7 we use this structure for connectivity queries.

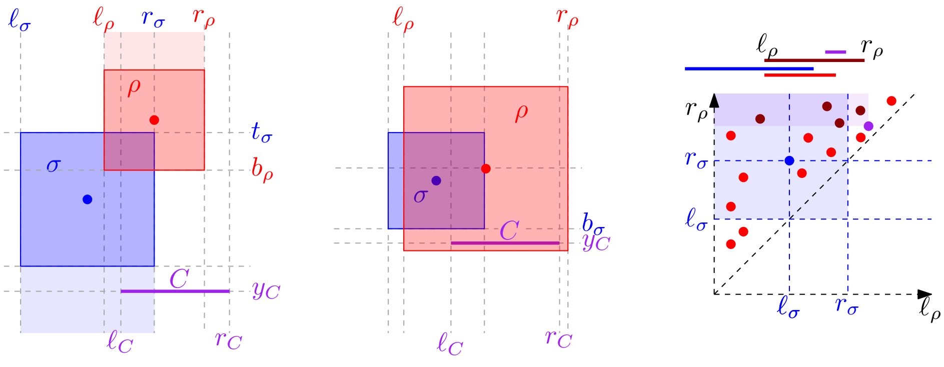
6 Square intersection data structure

Let $\mathcal{R}$ be a set of m squares. Let $R_1, \ldots, R_k$, be k subsets of $\mathcal{R}$ that we refer to as *conflict sets* and let $\ell \leq k$ be the maximum number of conflict sets that any square from $\mathcal{R}$ appears in. We want to store $\mathcal{R}$ and all the conflict sets $R_1, \ldots, R_k$ in a data structure that has size near linear in m and $z = \sum_i |R_i|$, and support the following operations:

Insert(ρ): Insert a square ρ into $\mathcal{R}$.
Delete(ρ): Delete a square ρ from $\mathcal{R}$ and every R_i that it occurs in.
Insert(ρ, R_i): Insert a square ρ in the conflict set R_i. If $R_i = \emptyset$, create a new conflict set.
Delete(ρ, R_i): Delete a square ρ from the conflict set R_i. If R_i becomes empty, delete R_i.
Query(σ, R_i, C): Given a query square σ whose center lies below all centers of all squares in $\mathcal{R}$, and a horizontal line segment C below σ, return (if it exists) a square $\rho \in \mathcal{R}$ that intersects σ, but is not in the conflict set R_i, and that also does not contain C.

We map every square $\rho = [\ell_\rho, r_\rho] \times [b_\rho, t_\rho] \in \mathcal{R}$ to a point $p_\rho = (b_\rho, \ell_\rho, r_\rho)$ in $\mathbb{R}^3$, and store these points in a 3D-range tree T augmented for range counting queries [8]. Hence, every third-level subtree T_ν stores the number of points m_ν in T_ν. Furthermore, for each such subtree, and each conflict set R_i, consider the subset of points stored in the leaves of T_ν for which the corresponding square appears in R_i. If this set is non-empty then node ν also stores the size $m_{\nu,i} = |\{p_\rho \mid p_\rho \in T_\nu \wedge \rho \in R_i\}|$ of this set. Furthermore, we maintain a bipartite graph between the squares in $\mathcal{R}$ and the conflict sets R_i, so that given a square $\rho \in \mathcal{R}$ we can find the ℓ conflict sets it appears in in $O(\ell)$ time. We implement all trees using general balanced trees [4] so that we can perform updates efficiently. A 3D-range tree uses $O(m \log^2 m)$ space. Each square in the multiset $\bigcup_i R_i$ contributes to $O(\log^3 m)$ nodes of T, and hence the entire structure uses at most $O((m+z)\log^3 m)$ space. We show how to answer our queries and how to update the data structure:

▶ **Lemma 8.** *Let $\mathcal{R}$ be a set of m squares, let $z = \sum |R_i|$, and let there be at most $k \leq m$ conflict sets. Let each $\rho \in \mathcal{R}$ appear in at most l conflict sets. There is a data structure $\mathcal{D}^*(\mathcal{R})$ of size $O((m+z)\log^3 m)$ that supports:*
Insert(ρ) *in $O(l \cdot \log^3 m)$ amortized deterministic time,*
Delete(ρ) *in $O(l \cdot \log^3 m)$ amortized deterministic time,*
Insert(ρ, R_i) *in $O(\log^3 m)$ amortized deterministic time,*
Delete(ρ, R_i) *in $O(\log^3 m)$ amortized deterministic time, and*
Query(σ, R_i, C) *in $O(\log^3 m)$ amortized deterministic time.*



(a) (b) (c)

Figure 5 (a) Since the center of ρ lies above the center of σ, we can essentially treat ρ as a rectangle unbounded from the top, and σ as a rectangle unbounded from the bottom. (b) A square ρ may intersect σ but is not allowed to contain C. (c) The x-extents of the objects map to points in $\mathbb{R}^2$. Squares (whose x-extent) intersects (the x-extent) of σ lie in the blue region, and are not allowed to lie in the purple region.

Proof. To insert a square $\rho \in \mathcal{R}$ we use the standard insertion procedure for (dynamic) 3D-range trees; we insert the point p_ρ into $O(\log^3 m)$ subtrees. If, one of our subtrees becomes too unbalanced, we rebuild it from scratch. Rebuilding a dD-range tree on a set P of n points can be done in $O(n \log^{d-1} n)$ time. However, we also still have to update the $m_{\nu,i}$ counts for each ternary subtree T_ν and each conflict list. We can do this in $O(ln \log^d n)$ time as follows. For each point $p_\rho \in P$ we obtain the at most l conflict sets R_i it appears in, and for each leaf corresponding to p_ρ we simply walk upward updating the $m_{\nu,i}$ counts appropriately. It follows that the amortized insertion time is $O(l \log^3 m)$. Deletions are handled similarly in $O(l \log^3 m)$ amortized time.

To insert or delete a square ρ in one of the conflict sets R_i we update the $m_{\nu,i}$ counts in the $O(\log^3 m)$ affected nodes (and we insert or delete the appropriate edge in the bipartite graph). If one of the $m_{\nu,i}$ counts reaches zero after a deletion, we stop storing it. Any counts that we do not store explicitly are considered to be zero.

Consider a query with square $\sigma = [\ell_\sigma, r_\sigma] \times [b_\sigma, t_\sigma]$, horizontal segment $C = [\ell_C, r_C] \times \{y_C\}$, and conflict set R_i (see also Figure 5). We will argue that there are $O(1)$ axis parallel boxes $Q_1, .., Q_{O(1)}$ such that the subset of squares from $\mathcal{R}$ that intersect σ but do not contain C is the subset of points that lies in $\bigcup_j Q_j$. Our range tree allows us to obtain $O(\log^3 m)$ ternary subtrees T_ν that together represent the points in this region (in $O(\log^3 m)$ time). For each such subtree we then consider the counts m_ν and $m_{\nu,i}$: if they are equal all points (squares) in T_ν also appear in R_i, and hence there are no candidate points (squares) to be found in T_ν. Otherwise, we have $m_\nu > m_{\nu,i}$, and hence T_ν does contain a point p_ρ for which ρ intersects σ, does not contain C, and for which $\rho \notin R_i$. Moreover, one of the two children of ν, say node μ, must then also have $m_\mu > m_{\mu,i}$. This way we can find p_ρ in time proportional to the height of T_ν. It follows that the total query time is $O(\log^3 m)$. All that remains is to describe the regions $Q_1, .., Q_{O(1)}$.

Since the center of σ is guaranteed to lie below all centers of squares in $\mathcal{R}$, and $y_C \leq b_\sigma$, we can essentially treat all squares as three-sided rectangles. In particular, a square $\rho \in \mathcal{R}$ intersects σ if and only if $p_\rho = (b_\rho, \ell_\rho, r_\rho)$ lies in the query range $Q = (-\infty, t_\sigma] \times (-\infty, r_\sigma] \times [\ell_\sigma, \infty)$ (see Figure 5). Using that $y_c \leq b_\sigma \leq (b_\rho + t_\rho)/2$, we find that $C \subset \rho$ if and only if p_ρ lies in the range $Q' = (\infty, y_C] \times (-\infty, \ell_C] \times [r_C, \infty)$. Hence, ρ intersects σ, but does not contain C if and only if $p_\rho \in Q \setminus Q'$. Since both Q and Q' are orthogonal boxes this region can be expressed as the union of $O(1)$ orthogonal ranges. ◀

7 Maximal Bichromatic Matchings

The Maximal Bichromatic Matching data structure (MBM) in [16] relies upon a square intersection data structure $\mathcal{D}$. Consider a pair of disjoint quadtree cells C_1, C_2 and two sets $R \subseteq \pi(C_1)$ and $B \subseteq \pi(C_2)$. The MBM stores two square intersection data structures: $\mathcal{D}(R)$ and $\mathcal{D}(B)$, plus a maximal bichromatic matching M_{RB} of the graph $G[R \cup B]$.

Given two such cells C_1, C_2, they dynamically maintain the matching as follows: For all edges in M_{RB}, dynamically remove the endpoints from the square intersection structures storing $\mathcal{D}(R \backslash M_{RB})$ and $\mathcal{D}(B \backslash M_{RB})$. When a new square σ gets inserted into B, query $\mathcal{D}(R \backslash M_{RB})$ to find a square in $R \backslash M_{RB}$ that intersects σ. If such a square ρ exists, add the edge (ρ, σ) to the matching M_{RB}. Subsequently delete ρ from $\mathcal{D}(R \backslash M_{RB})$. With a similar procedure for deletions, one can dynamically maintain M_{RB} in time proportional to the update and query time of the intersection data structure $\mathcal{D}$.

Defining the sets R and B. The sets R and B must be carefully chosen if we want to avoid spending quadratic space in n or ψ. Recall the pipeline of Kaplan et al. [16]. For each storing cell C_1, for each $C_2 \in \mathbb{P}^*(\pi(C_1))$, they store an MBM between the pair (C_1, C_2). We know that there may be $\Theta(\psi^2)$ cells in the set $\mathbb{P}^*(\pi(C_1))$. Suppose that for each pair C_1, C_2 they set $R \leftarrow \pi(C_1)$ and $B \leftarrow \pi(C_2)$. Every square in $\pi(C_1)$ may get stored $O(\psi^2)$ times and the total space usage is $O(\psi^2 n)$. To improve space and time usage, the authors of [16] instead set $R \leftarrow \{\sigma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\sigma)\}$ and $B \leftarrow \pi(C_2)$, see Figure 6(a). Since each square $\sigma \in \pi(C_1)$ has $O(\psi)$ cells in its perimeter $\mathbb{P}^*(\sigma)$, each square σ is stored $O(\psi)$ times and the total space is $O(\psi \cdot |\pi(C_1)|)$. A charging argument then shows that the total space required is $O(\psi n)$. The data structures can be updated in $O(\psi)$ times: the update time of the intersection data structures $\mathcal{D}(R \backslash M_{RB})$ and $\mathcal{D}(B \backslash M_{RB})$.

Defining an MBM for adaptive ψ. When the aspect ratio is adaptive (or, even when it is bounded), the approach by Kaplan et al. [16] requires an update time linear in ψn, since, after increasing ψ, the perimeter $\mathbb{P}^*(\pi(\sigma))$ increases in size by $O(\psi)$ for every storing cell C_1. We could try to avoid this issue by replacing their definition of perimeter with ours. That is, for every cell C_1, we would consider the $O(\psi^2)$ cells C_2 in $\mathcal{P}(\pi(C_1))$. For the pair C_1, C_2, we want to maintain an MBM between sets $R \leftarrow \{\sigma \in \pi(C_1) \mid C_2 \in \mathcal{P}(\sigma)\}$ and $B \leftarrow \pi(C_2)$; by storing R and B each in their separate data structure for square intersection queries. However, such a structure can also not be efficiently dynamically maintained, see Figure 6(b). Thus, we must avoid storing the sets $R \backslash M_{RB}$ and $B \backslash M_{RB}$ explicitly in a data structure.

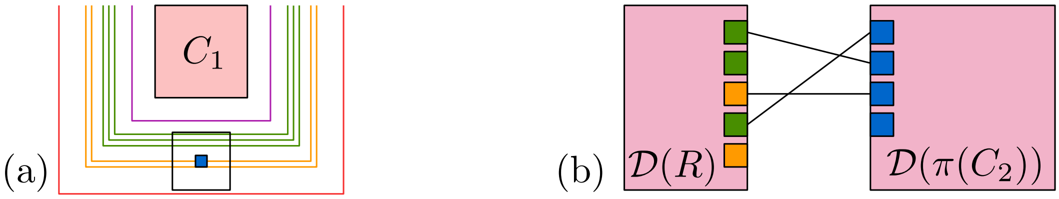


Figure 6 (a) C_1 with a blue $C_2 \in \mathbb{P}^*(\pi(C_1))$. The elements of the set $R = \{\sigma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\sigma)\}$ are the green and yellow squares. (b) If we split C_2 to create a cell C', then the corresponding R' would consist only of the orange squares. Since there exists no efficient way to split intersection data structures, constructing the new data structure on R' takes linear time.

Applying our data structure problem. We now apply our previous data structure problem. Let C_1 be a storing cell in our quadtree $T(\mathcal{S})$. We maintain the data structure $\mathcal{D}^*(\pi(C_1))$ (i.e. we set $\mathcal{R} \leftarrow \pi(C_1)$). Let there be k cells in the perimeter $\mathcal{P}(\pi(C_1))$. Then by Lemma 5, we have $k \in O(\psi^2)$. Let C_i be some storing cell in $\mathcal{P}(\pi(C_1))$, we denote by M_i some maximal matching in the graph $G[R \cup B]$ for $R \leftarrow \{\gamma \in \pi(C_1) \mid C_i \in \mathcal{P}(\gamma)\}$ and $B \leftarrow \pi(C_i)$. Denote by R_i the squares in $\pi(C_1)$ that are part of M_i; we say that R_i is a conflict set. The result of this transformation are k conflict sets of $\pi(C_1)$. Moreover, each square ρ in $\pi(C_1)$ may appear in at most $l \in O(|\mathcal{P}(\rho)|) \subset O(\psi)$ conflict sets. We now apply Lemma 8 four times (one for each direction). We maintain for all C_1 this data structure $\mathcal{D}^*(\pi(C_1))$ and show:

▶ **Theorem 9.** *Let $\mathcal{S}$ be a set of n squares with adaptive aspect ratio ψ. We can maintain for pair of storing cells (C_1, C_2) a maximal bichromatic matching in $G[R \cup \pi(C_2)]$ with $R \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathcal{P}(\gamma)\}$. Our solution uses $O(n \log \psi \log^3 n)$ space. Inserting/deleting a square σ requires $O(|\mathcal{P}(\sigma)| \cdot \log^3 n + \log^5 n)$ amortized time.*

Proof. We first analyse our space usage and then show how to maintain each matching.

Upper bounding size. For C_1, denote by z_1 the sum of all C_i, over all edges in the maximal matching of $G[R \cup \pi(C_i)]$ with $R \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\gamma)\}$. Lemma 8 presents a data structure with size $O((|\pi(C_1)| + z_1) \log^3 |\pi(C_1)|)$. Denote by $\mathcal{M}(\mathcal{S})$ the set of all edges, across all maximal bichromatic matchings, for all pairs of storing cells (C_1, C_2). Let $\mathcal{M}(\mathcal{S})$ contain z^* elements. It follows that all these data structures use at most $O((n + z^*) \cdot \log^3 n)$ total space. We upper bound the number of edges in z^* by charging each edge to one of their endpoints. Intuitively, we charge each matched edge to the squares of the smallest quadtree cell. Every square $\sigma \in \mathcal{S}$ receives at most $O(\log \psi)$ charges and z^* is upper bound by $n \log \psi$.

More formally, we over-estimate the edges in $\mathcal{M}(\mathcal{S})$. Fix for *every* pair (C_1, C_2) with $C_2 \in \mathcal{P}(\pi(C_1)))$ an arbitrary maximal bichromatic matching in the graph $G[\pi(C_1) \cup \pi(C_2)]$ (i.e., we ignore the fact that we match between sets $R \subseteq \pi(C_1)$ and $B \subseteq \pi(C_2)$, and fix some potentially larger matching in the bigger graph $G[\pi(C_1) \cup \pi(C_2)]$). Denote for C_1 by $\mathcal{M}_{\prec}(C_1)$ the set of all matchings between C_1 and C_2 where $|C_1| \prec |C_2|$ for $\prec \in \{<, =, >\}$. Any edge $e \in \mathcal{M}(\mathcal{S})$ is in $\mathcal{M}_{=}(C_1) \cup \mathcal{M}_{<}(C_1)$ for some storing cell C_1. First, we upper bound $|\mathcal{M}_{=}(C_1)|$. There are $O(1)$ cells C_2 with $|C_1| = |C_2|$ and $C_1 \in \mathcal{P}(\pi(C_2))$ or vice versa. For every such C_2, there can be at most $O(|\pi(C_1)|)$ edges in a MBM in $G[\pi(C_1) \cup \pi(C_2)]$. Thus, there are at most $O(|\pi(C_1)|)$ edges in $\mathcal{M}_{=}(C_1)$. Second, by Lemma 6, there are at most $O(\log \psi)$ cells C_2 with $C_1 \in \mathcal{P}(\pi(C_2))$ and $|C_1| < |C_2|$. Again, every matching in $G[\pi(C_1) \cup \pi(C_2)]$ has at most $O(|\pi(C_1)|)$ edges, thus $\mathcal{M}_{<}(C_1)$ contains at most $O(|\pi(C_1)| \cdot \log \psi)$ edges. Now:

$$z^* = |\mathcal{M}(\mathcal{S})| \leq \sum_{\text{storing cell } C_1} |\mathbb{M}_{=}(C_1)| + |\mathbb{M}_{<}(C_1)| \leq \sum_{\text{storing cell } C_1} |\pi(C_1)| \cdot \log \psi \leq n \log \psi$$

It follows we use at most $O((n + z^*) \cdot \log^3 n) \subset O(n \log \psi \log^3 n))$ space.

Maintaining the MBM. Suppose that we delete a square σ from $\mathcal{S}$ (this is the more difficult case). We can find its storing cell C_σ in $O(\log n)$ time using standard quadtree navigation. We obtain $\mathcal{D}^*(\pi(C_\sigma))$ with its $k \in O(\psi^2)$ conflict sets. Recall that σ appears in at most $l \in O(|\mathcal{P}(\sigma)|)$ conflict sets. By Lemma 8, we may remove σ from the data structure $\mathcal{D}^*(\pi(C_\sigma))$ in $O(l \log^2 n) \subseteq O(|\mathcal{P}(\sigma)| \cdot \log^2 n)$ amortized time. What remains is to update all the matchings. We recall that we maintain a matching between (C_σ, C_2) in two cases: either the cell $C_2 \in \mathcal{P}(\pi(C_\sigma))$ or $C_\sigma \in \mathcal{P}(\pi(C_2))$. There are at most $O(|\mathcal{P}(\pi(C_\sigma))|)$ cells of the first case, and $O(\log \psi)$ cells of the second case. By Theorem 7, we may obtain all such cells in $O(\log^5 n + |\mathcal{P}(\sigma)|)$ time.

Processing a cell C_2. Fix a cell C_2 with a corresponding conflict set R_2 in $\mathcal{D}^*(\pi(C_\sigma))$. We test if σ was an endpoint of the matching (σ, ρ) by searching over the conflict set R_2. If so, then we delete σ from the conflict set R_2. What remains is to try and rematch ρ.

Thus, we want to find a square in $R = \{\gamma \in \pi(\sigma) \mid C_2 \in \mathcal{P}(\gamma)\}$, that is not already in the conflict set R_2 (i.e. not already in the matching between $G[R \cup B]$). Denote by K the cell C_σ scaled by a factor 5 around its center and by $\underline{K}$ the bottom facet of K. We claim that ρ can be matched to a square in R if and only if Query$(\rho, R_2, \underline{K})$ from Lemma 8 is not empty.

Indeed, for any $\gamma \in \pi(C_\sigma)$ that intersects ρ and contains $\underline{K}$, must contain K. By definition, $C_2 \notin \mathcal{P}(\gamma)$ and thus $\gamma \notin R$. For any $\gamma \in \pi(C_\sigma)$ that intersects ρ where $\gamma \in R_2$, by definition $\gamma \notin R \backslash M_{RB}$. For any $\gamma \in \pi(C_\sigma)$ that does not intersect ρ, there is no edge between γ and ρ in $G[R \cup B]$. It follows that with one query we may rematch ρ in $O(\log^3 n)$ amortized time.

Since there are at most $O(|\mathcal{P}(\sigma)| + \log \psi)$ cells C_2 to consider, we can maintain every MBM in $O(|\mathcal{P}(\sigma)| \cdot \log^3 n + \log^5 n)$ time. ◀

8 Dynamic connectivity in square intersection graphs

Having formally introduced and analysed every component, we can now fully state what our data structure maintains. For an illustration, we refer back to Figure 2. We store a data structure that uses at most $O(n \log^3 n \log \psi)$ space:

(1) We store $\mathcal{S}$ in a quadtree $T(\mathcal{S})$.
- This quadtree contains for each cell $\sigma \in \mathcal{S}$ the neighborhood $\mathcal{N}(\sigma)$. Additionally, we
- maintain all $C \in \mathcal{C}(\mathcal{S})$ with $O(|\mathcal{C}(\sigma)| \cdot \log^4 n + \log^6 n)$ amortized time (Thm 7).
- This quadtree requires $O(n \log \psi)$ space (Thm 7).

(2) We augment our quadtree with a Marked-Ancestor Tree (MAT).
- We mark each cell $C \in \mathcal{C}(\mathcal{S})$ in the MAT (Thm 7).

(3) For any storing cell C, we define a conflict set R_i for all cells $C_i \in \mathcal{P}(\pi(C))$. We store $\pi(C)$ with the conflict sets in our square intersection data structure $\mathcal{D}^*(\pi(C))$.
- Let $z^* = \sum_C \sum_i |R_i|$, the total space required is $O((n + z^*) \log^3 n)$ (Lem 8).
- By the proof of Theorem 9, $z^* \in O(n \log \psi)$ so we use $O(n \log \psi \log^3 n)$ total space.

(4) For each storing cell C_1 and each $C_2 \in \mathcal{P}(\pi(C_1))$, we store a Maximal Bichromatic Matching (MBM) in $G[R \cup B]$.
- We set R as the set of squares in C_1 that have C_2 in their perimeter ($R \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathcal{P}(\gamma)\}$ and $B \leftarrow \pi(C_2)$).
- Updates in $\mathcal{S}$ require $O(|\mathcal{P}(\sigma)| \cdot \log^3 n + \log^5 n)$ amortized time (Thm 9).

(5) Finally, for any pair (C_1, C_2), if their MBM is not empty, we store an edge between them.
- We maintain the resulting "proxy graph" in the HLT data structure [14].
- Inserting or deleting a square σ introduces at most $O(|\mathcal{P}(\sigma)| + \log \psi)$ new edges.

We finally show that this data structure implies the following:

▶ **Theorem 10.** *Let $\mathcal{S}$ be a set of squares with adaptive aspect ratio ψ. We can store $\mathcal{S}$ in a dynamic data structure of size $O(n \log^3 n \log \psi)$ with $O((|\mathcal{C}(\sigma)| + |\mathcal{P}(\sigma)|) \log^4 n + \log^6 n)$ amortized deterministic update time such that for any pair of squares (σ, ρ) we can query for the connectivity between σ and ρ in $O(\log n / \log \log n)$ time.*

Proof. Our pipeline functions identical to the pipeline of [16]. Given σ, we obtain a pointer to its storing cell C_σ in $O(1)$ time. We then query the marked-ancestor tree in $O(\log n / \log \log n)$ time to find the largest ancestor C_α of C_σ that is marked. The cell C_α is marked by at least one squares γ that contains C_α in its interior. We obtain a pointer to γ and its storing cell C^* in $O(1)$ time. We note that if there is also some squares γ' that marked C_α, we may arbitrarily get a pointer to either γ or γ'. Doing the same procedure for ρ gives a cell R^*. We test whether C^* and R^* are connected in the proxy graph $O(\log n/\log\log n)$ time. We now claim that these two cells are connected in the proxy graph if and only if (ρ, σ) are connected. The key observation to prove this claim is that, if we were to rescale the plane, our graph contains the proxy graph maintained by Kaplan et al. [16] as a subgraph. Indeed at the time of a query, ψ is fixed. Thus, we may rescale the plane such that every square has a diameter in $[\frac{1}{\psi}, 1]$. Let $H(\mathcal{S})$ be the quadtree of [16], then $T(\mathcal{S}) \subset H(\mathcal{S})$. Kaplan et al. maintain for every pair (C_1, C_2) with $C_2 \in \mathbb{P}^*(\pi(C_1))$ an Maximal Bichromatic Matching in the graph $G[R' \cup B']$ for $R' \leftarrow \{\gamma \in \pi(C_1) \mid C_2 \in \mathbb{P}^*(\gamma)\}$ and $B' \leftarrow \pi(C_2)$. For each nonempty MBM between a pair (C_1, C_2), they store an edge in the proxy graph.

Note that if the MBM is nonempty, then both C_1 and C_2 are storing cells. It follows that $C_2 \in \mathcal{P}(\pi(C_1))$; and that $R' = R$. Thus, we store for each non-empty MBM a maximal bichromatic matching in the graph $G[R \cup B] = G[R' \cup B']$ as in [16]. This implies that after rescaling, whenever there exists an edge in the proxy graph of [16], there exists an edge in our data structure. Thus, we may immediately apply the proof of Theorem 4.3 in [16] to conclude that (σ, ρ) are connected if and only if (C^*, R^*) are. ◀

References

1 Anders Aamand, Adam Karczmarz, Jakub Lacki, Nikos Parotsidis, Peter M. R. Rasmussen, and Mikkel Thorup. Optimal decremental connectivity in non-sparse graphs. *CoRR*, abs/2111.09376, 2021. arXiv:2111.09376.

2 Stephen Alstrup, Thore Husfeldt, and Theis Rauhe. Marked ancestor problems. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. no. 98CB36280)*, pages 534–543. IEEE, 1998.

3 Stephen Alstrup, Jens P. Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *Inf. Process. Lett.*, 64(4):161–164, 1997. doi:10.1016/S0020-0190(97)00170-1.

4 Arne Andersson. General balanced trees. *J. Algorithms*, 30(1):1–18, 1999. doi:10.1006/jagm.1998.0967.

5 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

6 Timothy M Chan and Zhengcheng Huang. Dynamic geometric connectivity in the plane with constant query time. *arXiv preprint arXiv:2402.05357*, 2024.

7 Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011. doi:10.1137/090751670.

8 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: Algorithms and applications, 3rd Edition.* Springer, 2008. URL: https://www.worldcat.org/oclc/227584184.

9 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 599–608. ACM/SIAM, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644208.

10 David Eppstein, Zvi Galil, Giuseppe F Italiano, and Thomas H Spencer. Separator based sparsification for dynamic planar graph algorithms. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of Computing*, pages 208–217, 1993.

11 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. doi:10.1137/0214055.

12 Sariel Har-Peled. Quadtrees-hierarchical grids. *Lecture notes*, 2010.

13 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.

14 Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

15 Jacob Holm and Eva Rotenberg. Good r-divisions imply optimal amortized decremental biconnectivity. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.42.

16 Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity in disk graphs. In *38th International Symposium on Computational Geometry (SoCG 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

17 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete & Computational Geometry*, 64(3):838–904, 2020.

18 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142. SIAM, 2013. doi:10.1137/1.9781611973105.81.

19 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Computational Geometry*, 111:101979, 2023.

20 Fabian Kuhn, Rogert Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the 2003 Joint Workshop on Foundations of Mobile Computing*, DIALM-POMC '03, pages 69–78, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/941079.941089.

21 Jakub Lacki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. 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 608–621. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.608.

22 Chih-Hung Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM Journal on Computing*, 51(3):723–765, 2022.

23 Maarten Löffler, Joseph A Simons, and Darren Strash. Dynamic planar point location with sub-logarithmic local updates. In *Algorithms and Data Structures: 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings 13*, pages 499–511. Springer, 2013.

24 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.92.

25 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

26 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979. doi:10.1016/0022-0000(79)90042-4.

27 Robert Endre Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *J. ACM*, 31(2):245–281, 1984. doi:10.1145/62.2160.

28 Mikkel Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, 1999. doi:10.1006/jagm.1999.1033.

29 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350. ACM, 2000. doi:10.1145/335305.335345.

30 Dan E Willard and George S Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM (JACM)*, 32(3):597–617, 1985.

Pebble Games and Algebraic Proof Systems

Lisa-Marie Jaser
Institut für Theoretische Informatik, Universität Ulm, Germany

Jacobo Torán
Institut für Theoretische Informatik, Universität Ulm, Germany

Abstract

Analyzing refutations of the well known pebbling formulas $\mathrm{Peb}(G)$ we prove some new strong connections between pebble games and algebraic proof system, showing that there is a parallelism between the reversible, black and black-white pebbling games on one side, and the three algebraic proof systems Nullstellensatz, Monomial Calculus and Polynomial Calculus on the other side. In particular we prove that for any DAG G with a single sink, if there is a Monomial Calculus refutation for $\mathrm{Peb}(G)$ having simultaneously degree s and size t then there is a black pebbling strategy on G with space s and time $t+s$. Also if there is a black pebbling strategy for G with space s and time t it is possible to extract from it a MC refutation for $\mathrm{Peb}(G)$ having simultaneously degree s and size ts. These results are analogous to those proven in [14] for the case of reversible pebbling and Nullstellensatz. Using them we prove degree separations between NS, MC and PC, as well as strong degree-size tradeoffs for MC.

We also notice that for any directed acyclic graph G the space needed in a pebbling strategy on G, for the three versions of the game, reversible, black and black-white, exactly matches the variable space complexity of a refutation of the corresponding pebbling formula $\mathrm{Peb}(G)$ in each of the algebraic proof systems NS, MC and PC. Using known pebbling bounds on graphs, this connection implies separations between the corresponding variable space measures.

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Complexity theory and logic; Mathematics of computing → Graph theory

Keywords and phrases Proof Complexity, Algebraic Proof Systems, Pebble Games

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.64

1 Introduction

The use of pebble games in complexity theory goes back many decades. They offer a very clean tool to analyze certain complexity measures, mainly space and time, in an isolated way on a graph, which can then be translated to specific computational models. Very good overviews of these results can be found in [26, 28, 23].

We consider several versions of the game, defined formally in the preliminaries. Intuitively, the goal of these games is to measure the minimum number of pebbles needed by a single player in order to place a pebble on the sink of a directed acyclic graph (DAG) following certain rules (this is called the pebbling price). A black pebble can only be placed on a vertex if it is a source or if all its direct predecessors already have a pebble on them, but these pebbles can be removed at any time. A white pebble (modelling non-determinism) can be placed on any vertex at any time but can only be removed if all its direct predecessors contain a pebble. In the reversible pebble game, pebbles can only be placed or removed from a vertex if all the direct predecessors of the vertex contain a pebble. These three games define a short hierarchy being reversible pebbling weaker than black pebbling and this in turn weaker than the black-white pebble game.

In proof complexity one tries to understand the resources needed for a proof of a mathematical statement in a formalized system. Pebbling games have also become one of the most useful tools for proving results in this area. The reason for this is that one can often

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 64; pp. 64:1–64:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

translate a certain measure for the pebbling game, mainly number of pebbles or pebbling time, into a suitable complexity measure for a concrete proof system. Very often the bounds for this measure in a graph translate accurately to bounds in the different proof systems for a certain kind of contradictory formulas mimicking the game, called pebbling formulas. These formulas were introduced in [6] and have been extremely useful for proving separations, upper and lower bounds as well as tradeoff results in basically all studied proof systems. See e.g. [22].

In the present paper we will concentrate on algebraic proof system. In these systems formulas are encoded as sets of polynomials over a field and the question of whether a formula is unsatisfiable is translated to the question of whether the polynomials have a common root. Powerful algebraic tools like the Gröbner Basis Algorithm can be used for this purpose. Several algebraic proof systems have been introduced in the literature (defined formally bellow). Well known are Nullstellensatz (NS) introduced in [3] and the more powerful Polynomial Calculus (PC) defined in [11]. The first one is usually considered as a static system in which a "one-shot" proof has to be produced, while in PC there are certain derivation rules like in a more standard proof system.

The best studied complexity measures for refutations in these systems are the degree (maximum degree of a polynomial) and size (number of monomials counted with repetitions). For studying the connections with the pebble games it is very useful to consider also space measures and the configurational refutations associated with space. We will use the variable space measure (number of variables that are simultaneously active in a refutation).

In [8] the Monomial Calculus system (MC) was identified. This system is defined by limiting the multiplication rule in PC to monomials and its power lies between NS and PC. Building on results from [2] for the Sherali-Adams proof system, the authors proved that for any pair of non-isomorphic graphs, the MC degree for the refutation of the corresponding isomorphism formulas exactly corresponds to the Weisfeiler-Leman bound for separating the graphs, a very important tool in graph theory and descriptive complexity. This equivalence (as well as the relations to pebbling shown here) motivates the study of Monomial Calculus as a natural proof system between NC and PC.

As mentioned above, connections between pebbling games and algebraic systems have been known. Already in [9] it was proved that for any directed acyclic graph (DAG) G the corresponding pebbling formula $\mathrm{Peb}(G)$ can be refuted with constant degree in PC but in NS it requires degree $\Omega(s)$, where s is the black pebbling price of $G, \mathsf{Black}(G)$. Using pebbling results, this automatically proves a strong degree separation between NS and PC. As a more recent example, the authors in [14] proved a very tight connection between NS and the reversible pebbling game. They showed that space and time in the game played on a DAG exactly correspond to the degree and size measures in a NS refutation of the corresponding pebbling formula. From this connection strong degree-size tradeoffs for NS follow. This result also improves degree separation from [9] since it is known that there are graphs for which the reversible pebbling price is a logarithmic factor larger than the black pebbling price.

We show in this paper that besides these results, there are further parallelisms between the reversible, black and black-white game hierarchy on one side, and the NS, MC and PC proof systems on the other side.

1.1 Our Results

In Section 3 we prove that very similar results to those given in [14] for NS and reversible pebbling are also true for the case of MC and black pebbling. More concretely we show in Theorem 13 that for any DAG G with a single sink, if there is a MC refutation for $\mathrm{Peb}(G)$

having simultaneously degree s and size t then there is a black pebbling strategy on G with space s and time $t+s$. This is done by proving that any Horn formula has a very especial kind of MC refutation, which we call input monomial refutation since it is the same concept as an input refutation in Resolution. Horn formulas constitute an important class with applications in many areas like program verification or logic programming. It is well known that input Resolution is complete for Horn formulas.

For the other direction, we show in Theorem 8 that from a black pebbling strategy for G with space s and time t it is possible to extract a MC refutation for $\mathrm{Peb}(G)$ having simultaneously degree s and size ts. The small loss in the time parameter compared to the results in [14] comes from the fact that size complexity is measured in slightly different ways in NS and MC. Using these results we are able to show degree separations between NS and MC as well as the first strong degree separations between MC and PC. We also use the simultaneous relation with time and space in the black pebbling game to obtain strong degree-size tradeoffs for MC in the same spirit as those in [14]. The results also show that strong degree lower bounds for MC refutations do not imply exponential size lower bounds as it happens in the PC proof system [19].

The degrees of the refutation for pebbling formulas in NS and MS correspond exactly to the space in reversible and black games respectively. It would be very nice if the same could be said about PC degree and space in the black-white game. Unfortunately this is not the case since as mentioned above, it was proven in [9] that for any DAG the corresponding pebbling formula can be refuted within constant PC degree. We notice however that if instead of the degree we consider the complexity measure of variable space, then the connection still holds. We notice that for for any single sink DAG G the variable space complexity of refuting $\mathrm{Peb}(G)$ in each of the algebraic proof systems NS, MC and PC is exactly the space needed in a strategy for pebbling G in each of the three versions Reversible, Black and Black-White of the pebble game. These results allow us to apply known separations between the pebbling space needed in the different versions of the the game, in order to obtain separations in the variable space measure between the different proof systems.

2 Preliminaries

2.1 Pebble Games

Black pebbling was first mentioned implicitly in [24], while black-white pebbling was introduced in [12]. Note, that there exist several variants of the (black-white) pebble game in the literature. For differences between these variants, we refer to [23]. For the following definitions, let $G=(V,E)$ be a DAG with a unique sink vertex z.

▶ **Definition 1** (Black and black-white pebble games). *The* black-white pebble game *on G is the following one-player game: At any time i of the game, there is a* pebble configuration *$\mathbb{P}_i := (B_i, W_i)$, where $B_i \cap W_i = \varnothing$ and $B_i \subseteq V$ is the set of black pebbles and $W_i \subseteq V$ is the set of white pebbles, respectively. A pebble configuration $\mathbb{P}_{i-1} = (B_{i-1}, W_{i-1})$ can be changed to $\mathbb{P}_i = (B_i, W_i)$ by applying exactly one of the following rules:*

Black pebble placement on v: *If all direct predecessors of an empty vertex v have pebbles on them, a black pebble may be placed on v. More formally, letting $B_i = B_{i-1} \cup \{v\}$ and $W_i = W_{i-1}$ is allowed if $v \notin B_{i-1} \cup W_{i-1}$ and $\mathrm{pred}_G(v) \subseteq B_{i-1} \cup W_{i-1}$. In particular, a black pebble can always be placed on an empty source vertex s, since $\mathrm{pred}_G(s) = \varnothing$.*

Black pebble removal from v: *A black pebble may be removed from any vertex at any time. Formally, if $v \in B_{i-1}$, then we can set $B_i = B_{i-1} \setminus \{v\}$ and $W_i = W_{i-1}$.*

- **White pebble placement on v:** *A white pebble may be placed on any empty vertex at any time. Formally, if $v \notin B_{i-1} \cup W_{i-1}$, then we can set $B_i = B_{i-1}$ and $W_i = W_{i-1} \cup \{v\}$.*
- **White pebble removal from v:** *If all direct predecessors of a white-pebbled vertex v have pebbles on them, the white pebble on v may be removed. Formally, letting $B_i = B_{i-1}$ and $W_i = W_{i-1} \setminus \{v\}$ is allowed if $v \in W_{i-1}$ and $\mathrm{pred}_G(v) \subseteq B_{i-1} \cup W_{i-1}$. In particular, a white pebble can always be removed from a source vertex.*

A black-white pebbling *of G is a sequence of pebble configurations $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_t)$ such that $\mathbb{P}_0 = \mathbb{P}_t = (\emptyset, \emptyset)$, for some $i \le t$, $z \in B_i \cup W_i$, and for all $i \in [t]$ it holds that $\mathbb{P}_i$ can be obtained from $\mathbb{P}_{i-1}$ by applying exactly one of the above-stated rules.*

A black pebbling *is a pebbling where $W_i = \varnothing$ for all $i \in [t]$. Observe that w.l.o.g. we can always assume that $B_{t-1} = \{z\}$. For convenience we will also use the dual notion of* white pebbling *game. A white (only) pebbling is a pebbling where $B_i = \varnothing$ for all $i \in [t]$. Notice that $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_t)$ is a black pebbling of G if and only if $\mathcal{P}' = (\mathbb{P}'_t, \ldots, \mathbb{P}'_0)$ is a white pebbling of G, where each configuration $\mathbb{P}'_i$ contains the same set of pebbled vertices as in $\mathbb{P}_i$, but with white pebbles instead of black pebbles. In a white pebbling we can always suppose that $W_1 = \{z\}$.*

▶ **Definition 2** (Pebbling time, space, and price). *The* time *of a pebbling $\mathcal{P} = (\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_t)$ is $\mathsf{time}(\mathcal{P}) := t$ and the* space *of it is $\mathsf{space}(\mathcal{P}) := \max_{i \in [t]} |B_i \cup W_i|$. The* black-white pebbling price *(also known as the* pebbling measure *or* pebbling number*) of G, which we will denote by $\mathsf{BW}(G)$, is the minimum space of any black-white pebbling of G. The* black pebbling price *of G, denoted by $\mathsf{Black}(G)$, is the minimum space of any black pebbling of G. By the observation above, the white pebbling price $\mathsf{White}(G)$ coincides with $\mathsf{Black}(G)$*

Finally, we mention the reversible pebble game introduced in [7]. In the reversible pebble game, the moves performed in reverse order should also constitute a legal black pebbling, which means that the rules for pebble placements and removals have to become symmetric. This implies that reversible pebbling is a restricted version of black pebbling. The notions of reversible pebbling time, space, and price are defined as in the other pebbling variants.

2.2 Formulas and Polynomials

We will only consider propositional formulas in conjunctive normal form (CNF). Such a formula is a conjunction of clauses and a clause is a disjunction of literals. A literal is a variable or its negation. For a formula F, $\mathrm{Var}(F)$ denotes the set of its variables.

A Horn formula in a special type of CNF formula in which each clause has at most one positive literal. For a more detailed treatment of formulas as well as the well known Resolution proof system we refer the interested reader to some of the introductory texts in the area like [29]. We will basically only deal with pebbling formulas. These provide the connection between pebbling games and proof complexity.

▶ **Definition 3** (Pebbling formulas). *Let $G = (V, E)$ be a DAG with a set of sources $S \subseteq V$ and a unique sink z. We identify every vertex $v \in V$ with a Boolean variable x_v. For a vertex $v \in V$ we denote by $\mathrm{pred}(v)$ the set of its direct predecessors. In particular, for a source vertex v, $\mathrm{pred}(v) = \emptyset$. The* pebbling contradiction *over G, denoted $\mathrm{Peb}(G)$, is the conjunction of the following clauses:*

- *for all vertices v, the clause $\bigvee_{u \in \mathrm{pred}(v)} \bar{x}_u \vee x_v$,* (pebbling axioms)
- *for the unique sink z, the unit clause $\bar{x}_z$.* (sink axiom)

Observe that every clause in a pebbling formulas has at most one positive literal. These formulas are therefore Horn formulas.

A way to prove that a CNF formula is unsatisfiable is by translating it into a set of polynomials over a field $\mathbb{F}$ and then show that these polynomials do not have any common $\{0,1\}$-valued root. A clause $C = \bigvee_{x\in P} x \vee \bigvee_{y\in N} \bar{y}$ can be encoded as the polynomial $p(C) = \prod_{x\in P}(1-x)\prod_{y\in N} y$. A set of clauses $C_1,\ldots,C_m$ is translated as set of polynomials $p(C_1),\ldots,p(C_m)$. Adding the polynomials $x_i^2 - x_i$ (as axioms) for each variable x_i, there is no common $\{0,1\}$-valued root for all these polynomials if and only if the original set of clauses is unsatisfiable. The intuition here is to identify false with 1 and true with 0. A monomial is falsified by a Boolean assignment if all its variables get value 1, while it is satisfied if one of its variables gets value 0. In this context we will consider a monomial m as a set of variables and a polynomial p as a linear combination of monomials. A monomial with its coefficient in $\mathbb{F}$ is called a monomial term.

When encoding the pebbling formulas as polynomials, for a set $U \subseteq V$, we denote by m_U the monomial $\prod_{u\in U} x_u$. For $U = \emptyset$, $m_U = 1$. For every vertex $v \in V$ the axiom $\bigvee_{u\in \mathrm{pred}(v)} \bar{x}_u \vee x_v$ becomes the polynomial $A_v := m_{\mathrm{pred}(v)}(1-x_v)$, and the sink axiom $\bar{x}_z$ is transformed into the polynomial $A_{sink} := x_z$. Observe that every polynomial in the encoding of a pebbling formula has one or two monomials.

To avoid confusion we will denote the polynomial encoding of a CNF formula F by P_F.

2.3 Algebraic Proof Systems

Several proof systems that work with polynomials have been defined in the literature. The simplest one is *Nullstellensatz*, NS.

▶ **Definition 4.** *A Nullstellensatz refutation of the set of polynomials $p_1,\ldots,p_m$ in $\mathbb{F}[x_1,\ldots,x_n]$ consists of a set of polynomials $g_1,...,g_m,h_1,\ldots,h_n$ such that*

$$\sum_{j=1,\ldots,m} p_j g_j + \sum_{i=1,\ldots,n} h_i(x_i^2 - x_i) = 1.$$

As a consequence of Hilbert's Nullstellensatz, the NS proof systems is sound and complete for the set of encodings of unsatisfiable CNF formulas.

A stronger more dynamic algebraic refutational calculus also dealing with polynomials is the Polynomial Calculus (PC). As in the case of Nullstellensatz, PC is intended to prove the unsolvability of a set of polynomial equations.

▶ **Definition 5.** *The* PC *proof system uses the following rules:*

1. Linear combination

$$\frac{p \qquad q}{\alpha p + \beta q} \qquad \alpha,\beta \in \mathbb{F}.$$

2. Multiplication

$$\frac{p}{x_i p} \qquad i \in [n].$$

A refutation in PC *of an initial unsolvable set of polynomials $\mathcal{P}$ is a sequence of polynomials $\{q_1,\ldots,q_m\}$ such that each q_i is either a polynomial in $\mathcal{P}$, a Boolean axiom $x_i^2 - x_i$ or it is obtained by previous polynomials in the sequence applying one of the rules of the calculus.*

A less known algebraic proof system between NS and PC is Monomial Calculus, MC. This system was introduced in [8] identifying exactly the complexity of refuting graph isomorphism formulas. This proof system is defined like PC but the multiplication rule is only allowed to be applied to a monomial, or to a monomial times an axiom.

▶ **Definition 6.** *The* MC *proof system uses the following rules:*

1. Linear combination

$$\frac{p \qquad q}{\alpha p + \beta q} \qquad \alpha, \beta \in \mathbb{F}.$$

2. Multiplication

$$\frac{p}{x_i p} \qquad i \in [n], \quad p \quad \textit{is a monomial or the product of a monomial and an axiom.}$$

As is the case of PC, *a refutation in* MC *of an initial unsolvable set of polynomials $\mathcal{P}$ is a sequence of polynomials $\{q_1, \ldots, q_m\}$ where each one of them is either in $\mathcal{P}$, an axiom or is obtained by applying one of the rules of the calculus.*

As pointed out in [8], an equivalent definition of the Nullstellensatz system, but a dynamic one, would be to restrict the multiplication rule in the above definition even more, and only allow to apply it to polynomials that are a monomial multiplied by an axiom. In this way, the difference in the definition of the three systems NS, MC and PC is just a variation on how the multiplication rule can be applied. This alternative view of the definition also allows to consider configurational proofs in the NS system. In order to analyze and compare refutations we will consider several complexity measures on them.

▶ **Definition 7** (Complexity measures). *Let $\mathcal{C}$ be one of the mentioned systems $\mathcal{C} \in \{\text{NS}, \text{MC}, \text{PC}\}$ Let $\pi = \{q_1, \ldots, q_m\}$ be a $\mathcal{C}$ refutation. The degree of a polynomial q_i, $\deg(q_i)$ is the maximum degree of its monomials and the* degree *of π, $\deg_{\mathcal{C}}(\pi) = \max_{i=1,\ldots,n}(\deg(q_i))$. The size of π, denoted by $\text{Size}_{\mathcal{C}}(\pi)$ is the total number of monomials in π (counted with repetitions), when all polynomials p_i are fully expanded as linear combinations of monomials*[1].

For the space measures we need to define configurational proofs. Such a proof π in the system $\mathcal{C}$ is a sequence of configurations $\pi = C_0, \ldots C_t$ in which each C_i is a set of polynomials with $C_0 = \emptyset$ and $C_t = 1$. Each configuration C_i represents a set of polynomials that are kept simultaneously in memory at time i in the refutation, and for each $i, 0 < i \leq t$, C_i is either

- *$C_{i-1} \cup \{p\}$ for some axiom p (axiom download),*
- *$C_{i-1} \setminus \{p\}$ (erasure) or*
- *$C_{i-1} \cup \{p\}$ for some p inferred by the rules of $\mathcal{C}$ by some rule of the system (inference).*

The variable space of the proof π, $\text{VSpace}_{\mathcal{C}}(\pi)$ is defined as the maximum number of different variables appearing in any configuration of the proof.

For any of the defined complexity measures Comp and proof systems $\mathcal{C}$, and for every unsatisfiable set of polynomials P_F we denote by $Comp_{\mathcal{C}}(P_F \vdash)$ the minimum over all $\mathcal{C}$ refutations of P_F of $Comp_{\mathcal{C}}(\pi)$.

It is often convenient to consider a multilinear setting in which the multiplications in the mentioned algebraic systems are implicitly multilinearized. Clearly the degree and size measures can only decrease in this setting.

[1] Usually the size in the NS proof system is defined in a different way, for simplicity we keep this unifying definition although in some of the referenced results the size of NS refutations corresponds to the size definition given in the reference.

3 Monomial Calculus and pebbling formulas

In [14] it was shown that for any DAG G with a single sink, the reversible pebbling space and time of G, exactly coincides with the degree and the size of a NS refutation of Peb_G. We show that a very similar relation holds for the case of black pebbling and Monomial Calculus.

▶ **Theorem 8.** *Let G be a directed acyclic graph with a single sink z. If there is a black pebbling strategy of G with time t and space s then there is a MC refutation of* Peb_G *with degree s and size ts. The variable space of this refutation coincides with its degree.*

Proof. It is convenient to consider here the equivalent notion of white pebbling. Let $\mathcal{P} = (\mathbb{P}_0, \ldots, \mathbb{P}_t)$ be a white pebbling strategy for G with $\mathbb{P}_1 = \{z\}$ and $\mathbb{P}_t = \emptyset$ using s pebbles. We show that for each pebbling configuration $\mathbb{P}_i$, $i \in [t]$, $\mathbb{P}_i = \{v_{i_1}, \ldots, v_{i_{k_i}}\}$ the monomial $m_i = \prod_{v \in \mathbb{P}_i} x_v$ can be derived from Peb_G and m_{i-1} in degree s and size 1 if $\mathbb{P}_i$ adds a pebble, or size $2s-1$ if $\mathbb{P}_i$ removes a pebble. This proves the result since in the t steps of the pebbling strategy half of the steps add a pebble and the other half of the steps remove a pebble (each added pebble has to be removed). The total number of steps is therefore $\frac{t}{2} + \frac{t}{2}(2s-1) = ts$.

Pebble placement. If the configuration at pebbling step $i+1$ is reached after placing a white pebble on vertex v and $\mathbb{P}_i = \{u_{i_1}, \ldots, u_{i_{k_i}}\}$ with $k_i \leq s-1$ then $\mathbb{P}_{i+1} = \{v, u_{i_1}, \ldots, u_{i_{k_i}}\}$. Multiplying the monomial $m_i = \prod_{u \in \mathbb{P}_i} x_u$ by the variable x_v we obtain m_{i+1}. We have just added one more monomial of degree at most s to the proof.

Pebble removal. If the configuration at pebbling step $i+1$ is reached after removing a white pebble from vertex v and $\mathbb{P}_i = \{v, u_{i_1}, \ldots, u_{i_{k_i}}\}$ with $k_i \leq s-1$ then all predecessors of v are in the set $\{u_{i_1}, \ldots, u_{i_{k_i}}\}$. For the derivation of m_{i+1} we can multiply the axiom $(1 - x_v)\prod_{u \in \mathrm{pred}(v)} x_u$ by the variables in $\mathrm{Var}(m_i) \setminus (\bigcup_{u \in \mathrm{pred}(v)} x_u \cup \{x_v\})$, and add this polynomial to m_i obtaining m_{i+1}. Since there are at most $s-1$ variables in $\mathrm{Var}(m_i) \setminus (\bigcup_{u \in \mathrm{pred}(v)} x_u \cup \{x_v\})$, the number of intermediate monomials added to the proof (counting also monomial m_{i+1}) is at most $2(s-1)+1 = 2s-1$.

Observe that in all the steps in the refutation, at most two different monomials are active and the number of different variables in these monomials coincides with the largest of their degrees. This shows that the variable space of the MC refutation is also bounded by s. ◀

▶ **Observation 9.** *The size bound ts in the above proof comes from the way the MC rules are defined. As is the case of PC, in the multiplication rule only one variable at at time is multiplied, even when multiplying the axiom polynomials. When an axiom is multiplied by a monomial with several variables, all the intermediate polynomials contribute to the size of the MC refutation. This is different from the the usual way to measure the size in the NS case, where intermediate monomials are not counted. If we would define the MC rules as those in NS, that is, if a whole monomial could be multiplied by an axiom in one step, the size of the MC proof would be would avoid the s factor in the monomial size and obtain size $2t$ instead.*

In order to prove a result in the other direction we consider a very restricted kind of refutation in MC, similar to what is known as an input refutation in Resolution. It this kind of refutation in every Resolution step one of the parent clauses must be an axiom. Input Resolution is not complete, but it is complete for Horn formulas. We will show that the same is true for MC input refutations.

▶ **Definition 10.** *A MC refutation π of a contradictory set of polynomials F is called an* input refutation *if there is a sequence of monomials $M_0, \ldots, M_t$ such that M_0 is the product of a monomial and an axiom, $M_t = 1$ and for each i M_i is obtained by multiplying M_{i-1}*

times a variable, or by the linear combination rule from M_{i-1} and a monomial multiplied by an axiom polynomial. We will call the sequence of monomials $M_0, \dots, M_t$ the backbone of the proof.

▶ **Lemma 11.** *Let F be an unsatisfiable Horn formula and let P_F be the encoding of F as a set of polynomials. Let π be any MC refutation of P_F. There is an input MC refutation π' of P_F with at most the same size and degree as π.*

Proof. Let d and t be the degree and size of π. We can suppose that π is multilinear. We prove the result by induction on k, the number of times the multiplication rule is applied to a monomial derived in π. In the base case $k = 0$, π is just a NS refutation of P_F. This means that there is a linear combination of a set of polynomials S that adds up to 1. Each of these polynomials has the form of a polynomial axiom multiplied by a monomial and since F is a Horn formula, each polynomial in S has either one or two monomials. We will represent such a polynomial $p = \alpha_m m + \alpha_{m'} m'$ by the pair of monomials (m, m'). In all these polynomials the monomial terms have some coefficients α_m and $\alpha_{m'}$. Clauses without positive literals are encoded as single monomials. Some polynomial in S has a single monomial otherwise the whole set S would have a common root by setting all variables to 1. Moreover, there has to be a sequence of polynomials $p_1, \dots, p_\ell$ represented by the monomials $(\emptyset, m_1), (m_1, m_2), (m_2, m_3) \dots, (m_{\ell-1}, m)$ [2]. This is because the linear combination adds up to 1 and for this to happen, there has to be a polynomial $(\emptyset, m_1)$ in the linear combination since otherwise all monomials would have variables. Also the monomial m_1 in $(\emptyset, m_1)$ has to be cancelled and there has to be some other polynomial of the form (m_1, m_2) and so on. It must also hold that some polynomial in the sequence must have the form $(m_{\ell-1}, m)$ that can cancel with one of the polynomials with a single monomial m in S. We suppose that $p_1, \dots, p_\ell$ is a minimal sequence with these properties. Now we can define the input monomial refutation π starting at $M_0 = m$ and applying then ℓ linear combinations with axioms multiplied by monomials and deriving all the monomials $m_\ell, \dots, m_1$ until 1 is derived. Observe that the monomials $M_0, \dots M_t$ are exactly those appearing in $p_1, \dots, p_\ell$. By the minimality of the sequence we also know that the monomials in the backbone are all different.

All the monomials in π' belong also to π, therefore the degree of the new refutation is not larger than that in π. In fact all the polynomials in $p_1, \dots, p_\ell$ are already in π. Besides these polynomials π' contains also the ℓ new monomials in the backbone. Since the $p_1, \dots, p_\ell$ and m belong to π and in each linear combination of two polynomials at most one monomial vanishes, there are at least ℓ intermediate polynomials in π until 1 is reached. This means that the size of π' is bounded by t.

For the case $k > 0$ let m' be the first monomial in the proof that is the result of a multiplication from a derived monomial m and a variable x, $m' = xm$ in π. The same argument as above shows that there is a sequence of polynomials $p_1, \dots, p_\ell, \hat{m}$ in π from which we can extract an input monomial refutation that starts at $M_0 = \hat{m}$ and derives at some point $M_i = m$. In the next step the multiplication rule is applied to obtain $M_{i+1} = m'$. Observe that the set of polynomials $m' \cup P_F$ still has the Horn property and that there is sub-proof of π that refutes this set to the monomial 1 applying the multiplication rule at most $k - 1$ times. By induction hypothesis we know that there is a sequence of polynomials $p'_1, \dots, p'_{\ell'}$ in π represented by the monomials $(\emptyset, m'_1), (m'_1, m'_2), \dots, (m'_r, m')$ from which an input refutation of $P_F \cup m'$ can be extracted. We can put together both input MC refutations $M_0 \dots M_i$ and $M_{i+1}, \dots, 1$. Again we can assume that all the monomials in the backbone

[2] Since we are representing monomials by their set of variables, the monomial 1 is represented by $\emptyset$

are different since if $M_i = M_j$ for $i < j$, we could shorten π' by connecting M_i with M_{j+1}. By the same argument as in the base case the size and degree of the input MC refutation cannot be larger than that of π. ◀

Since pebbling formulas are Horn formulas we immediately obtain:

▶ **Corollary 12.** *Let G be a directed acyclic graph with a single sink vertex z and let π be a MC refutation of* $\mathrm{Peb}(G)$*. There is an input MC refutation π' of* $\mathrm{Peb}(G)$ *with at most the same size and degree as π.*

We consider next a result in the other direction.

▶ **Theorem 13.** *Let G be a directed acyclic graph with a single sink. Let π be a MC refutation of* $\mathrm{Peb}(G)$ *with degree s and size t. There is a black pebbling strategy with s pebbles and time $t + s$.*

Proof. Because of Corollary 12 we can suppose that there exits an input MC refutation with monomials $M_0, \dots M_t$ starting with $M_1 = mx_{\mathrm{sink}}$ for some monomial m and with $M_t = 1$. We describe a strategy for a white pebbling of G following π. At each step i only the vertices corresponding to variables in M_i have a pebble on them. In a multiplication step a new pebble is added, which is always possible in a white pebbling strategy. We only have to show that in case variables disappear when going from M_i to M_{i+1}, this is a correct pebbling move. But in this case, the step from i to $i + 1$ is a linear combination of M_i with the axiom for some vertex v, $m_{\mathrm{pred}(v)}(1 - x_v)$ multiplied by some monomial m. The only variable that can disappear in M_{i+1} is x_v and in this case $M_i = m_{\mathrm{pred}(v)}x_v$. Therefore all the vertices in $\mathrm{pred}(v)$ have pebbles on them and the pebble in x_v can be removed. At the end of the refutation, when the 1 monomial is reached there are no pebbles left on G. The number of pebbles present at any moment is the number of variables in any of the monomials and this is the degree of π. The number of pebbling steps needed is at most d steps to place a pebble in each variable of $M_1 = mx_{\mathrm{sink}}$ and then t more pebbling steps. ◀

▶ **Observation 14.** *For the case of Polynomial Calculus it is known that strong degree lower bounds imply size lower bounds. If a set of unsatisfiable polynomials P_F with n variables and constant degree requires PC refutations of degree s, then any PC refutation of P_F requires size at least $2^{\Omega(\frac{d^2}{n})}$ [19]. The previous results show that this does not hold for Monomial Calculus. This follows from the fact that there are graph families $\{G_n\}_{n=0}^{\infty}$ with n vertices and constant in-degree that require black pebbling space $\Omega(\frac{n}{\log n})$ [25]. Theorem 13 implies that the pebbling formulas for this graph family needs degree $\Omega(\frac{n}{\log n})$. On the other hand, for every single-sink DAG with n vertices there is a trivial black pebbling strategy using space n and pebbling time $2n$. By Theorem 8 this implies that the pebbling formulas corresponding to the graphs in $\{G_n\}_{n=0}^{\infty}$ have MC refutations of quadratic size in n. This is a family of formulas with MC refutation degree $\Omega(\frac{n}{\log n})$ but having quadratic size refutations, a very different situation from the PC case.*

3.1 Degree separations

The given relationships between MC and the black pebbling game allow for the immediate translation of pebbling results to Monomial Calculus. We start with some degree separations between MC and PC. The original motivation for introducing MC was the close connection between the degree complexity of the refutation of the graph isomorphism formulas in this proof system, and the Weisfeiler-Leman hierarchy [8]. Formulas corresponding to non-isomorphic graphs pairs that can only be distinguished using a large level of the WL algorithm, require a MC refutation with large degree. It was proven later in [1, 15], that the degree of a PC refutation of the isomorphism formulas cannot be much smaller than in the MC case, in fact the degrees of a MC and a PC refutation can only be a constant factor apart. We improve this separation and obtain an almost optimal degree separations by considering the pebbling formulas. In [9] it was shown that pebbling formulas have constant PC degree and that for any directed acyclic graph G with black pebbling price $B(G)$, the formula $\mathrm{Peb}(G)$ requires NS refutations with degree $\Omega(B(G))$. Since it is known that there are graph families $\{G_n\}_{n=0}^{\infty}$ with $\Theta(n)$ vertices and $B(G_n) = \Omega(\frac{n}{\log n})$ [25], this implies a degree separation of $\Omega(\frac{n}{\log n})$ between PC and NS. From Theorem 13 follows that this is in fact a degree separation between MC and PC.

▶ **Theorem 15.** *There is an unsatisfiable family of formulas $\{F_n\}_{n=0}^{\infty}$ with $\Theta(n)$ variables each, that have PC refutations of constant degree but require MC refutations of degree $\Omega(\frac{n}{\log n})$.*

For the case of NS, from Theorem 8 and the equivalence between reversible pebbling price and NS degree from [13], [14], follows that a separation between reversible and black pebbling price for a graph family implies a degree separation between NS and MC for the corresponding pebbling formulas. For example it is known that a directed path graphs with n vertices can be black pebbled with 2 pebbles but requires reversible pebbling number $\lceil \log n \rceil$ [7]. Translated to pebbling formulas this means:

▶ **Theorem 16.** *There is a family of unsatisfiable formulas $\{F_n\}_{n=0}^{\infty}$ with $\Theta(n)$ variables each, that have MC refutations of degree 2 but require NS refutations of degree $\lceil \log n \rceil$.*

There are other graph families for which a separation between the black and reversible pebbling prices by a logarithmic factor in the number of vertices is known, [10],[30]. The separations in pebbling for these graphs is translated into the next result.

▶ **Theorem 17.** *For any function $s(n) = O(n^{1/2-\epsilon})$ for constant $0 < \epsilon < \frac{1}{2}$ there is a family of unsatisfiable formulas $\{F_n\}_{n=0}^{\infty}$ with $\Theta(n)$ variables each, that have MC refutations of degree $O(s(n))$ but require NS refutations of degree $\Omega(s(n) \log n)$.*

The question of whether the separation between reversible and black pebbling space can be larger than a logarithmic factor in the number of nodes is open. The best known degree separation between NS and and MC is slightly better. This was obtained in [16] with very different methods. Using a classic result from descriptive complexity [18], the authors show that for for every constant $c \geq 1$ there are families of formulas F_n with O(n) variables that have a degree 3 MC refutation but require NS degree at least $\log^c(n)$. It is also open whether this degree separation between NS and MC is optimal.

3.2 Size-degree tradeoffs for MC

The close connections between black pebbling space and monomial calculus expressed in Theorems 8,13 make it possible to translate space-time tradeoffs for pebbling into degree-size tradeoffs for MC. There is a slight loss of the time parameter that comes from the extra space factor in the the MC refutation from Theorem 8. We present two such results as examples. The first one is an extreme tradeoff result that shows how decreasing the degree by one can make the size increase exponentially.

▶ **Theorem 18** ([28]). *There is a family of directed graphs $\{G_n\}_{n=0}^{\infty}$ having $\Theta(n^2)$ vertices each and with $\mathsf{Black}(G_n) = \Theta(n)$ for which any black pebbling strategy with $\mathsf{Black}(G_n)$ pebbles requires at least $2^{\Omega(n \log n)}$ steps while there is a pebbling strategy with $\mathsf{Black}(G_n) + 1$ pebbles and $O(n^2)$ steps.*

▶ **Corollary 19.** *There is a family of unsatisfiable formulas $\{F_n\}_{n=0}^{\infty}$ with F_n having $O(n^2)$ variables and $d_n \in O(n)$ such that F_n has a MC refutation of degree d_n but any MC refutation with this degree requires size $2^{\Omega(n \log n)}$. On the other side there is a MC refutation of F_n with degree $d_n + 1$ and size $O(n^3)$.*

As a second example we present a robust time-space result from [23].

▶ **Theorem 20.** *There is a family of directed graphs $\{G_n\}_{n=0}^{\infty}$ having $\Theta(n)$ vertices each and with $\mathsf{Black}(G_n) = O(\log^2 n)$, with a black pebbling strategy in space $O(n/\log n)$ and time $O(n)$. There is also a constant $c > 0$ for which any pebbling strategy using less than $cn/\log n$ pebbles requires at least $n^{\Omega(\log \log n)}$ steps.*

▶ **Corollary 21.** *There is a family of unsatisfiable formulas $\{F_n\}_{n=0}^{\infty}$ with F_n having $O(n)$ variables, and a constant $c > 0$ such that F_n has a MC refutation of degree $O(n/\log n)$ and size $O(n^2/\log n)$ but for which any MC refutation with degree smaller than $cn/\log n$ requires size at least $n^{\Omega(\log \log n)}$.*

4 Pebble Games and Variable Space

The equality between degree and pebbling price for the cases of Monomial Calculus and black pebbling from the previous section, as well as for Nullstellensatz and reversible pebbling from [14] cannot be extended to the case of Polynomial Calculus and black-white pebbling price since as already mentioned, it was proven in [9] that for any DAG G, $\deg_{\mathrm{PC}}(\mathrm{Peb}(G)) = O(1)$. We show in this section that the correspondence between the three pebbling variations and the proof systems holds if we consider the variables space measure instead.

It can be seen in the proof of Theorem 8, that not only the minimum degree of a monomial calculus refutation of $\mathrm{Peb}(G)$, but also the minimum variable space is bounded by the black pebbling price of G. The same can be observed in the proof of Theorem 3.1 in [14] for the case of Nullstellensatz and reversible pebbling. Considering the trivial fact that variable space measure is always greater or equal that the degree needed for the refutation of a formula in all three proof systems NS, MC and PC, and considering Theorems 13 as well as Theorem 3.4 in [14] this implies:

▶ **Observation 22.** *For every DAG G with a single sink,* $\mathrm{VSpace}_{\mathrm{NS}}(\mathrm{Peb}(G) \vdash) = \mathsf{Rev}(G)$ *and* $\mathrm{VSpace}_{\mathrm{MC}}(\mathrm{Peb}(G) \vdash) = \mathsf{Black}(G)$*.*

For the case of black-white pebbling it is known that for the Resolution proof system, the variable space needed in a refutation of $\mathrm{Peb}(G)$ equals $\mathsf{BW}(G)$. The inclusion from left to right is from [5] while the other inclusion appeared in [17]. This results can be extended to other proof systems using the following result:

▶ **Lemma 23** ([4], [27]). *Let S be a proof system that can simulate Resolution step by step without including new variables. For every unsatisfiable formula F,* $\mathrm{VSpace}_S(F \vdash) = \mathrm{VSpace}_{\mathrm{Res}}(F \vdash)$*.*

This implies:

▶ **Observation 24.** *For every DAG G with a single sink,* $\mathrm{VSpace}_{\mathrm{PC}}(\mathrm{Peb}(G) \vdash) = \mathsf{BW}(G)$*.*

Which together with Observation 22 shows the equivalence between variable space in the proof systems and the pebbling price in the three variations of the game.

4.1 Variable Space Separations

These observations allow us to use pebbling results to obtain separation in the variable space complexity in the algebraic proof systems. The reason why these results do not contradict Lemma 23, is that MC (or NS) cannot simulate Resolution step by step since the intermediate polynomials are not necessarily monomials.

For the variable space separations between NS and MC on pebbling formulas, the same degree separations given in Subsection 3.1 hold, since as we have seen, for this kind of formulas the variable space and the degree coincide in both proof systems. For the case of MC versus PC, it is known that for any DAG G, the separation between the black and the black-white pebbling prices can be at most quadratic [21]. This limits the variable space gap between MC and PC that can be obtained using pebbling formulas. In [31] a family of graphs is given that shows an asymptotic separation between the black-white and black pebbling prices. Translating this to our context we obtain:

▶ **Theorem 25.** *There is a family of unsatisfiable formulas* $\{F_n\}_{n=0}^{\infty}$ *with polynomially many variables (in* n*) such that* $\mathrm{VSpace}_{\mathrm{PC}}(F_n \vdash) = O(n)$ *and* $\mathrm{VSpace}_{\mathrm{MC}}(F_n \vdash) = \Omega(\frac{n \log n}{\log \log n})$.

An optimal quadratic separation between the black-white and black pebbling price was given in [20] but for a family of graphs having exponentially many vertices respect to their pebbling price. This implies:

▶ **Theorem 26.** *There is a family of unsatisfiable formulas* $\{F_n\}_{n=0}^{\infty}$ *with* $\exp(\Theta(n \log n))$ *many variables such that* $\mathrm{VSpace}_{\mathrm{PC}}(F_n \vdash) = O(n)$ *and* $\mathrm{VSpace}_{\mathrm{MC}}(F_n \vdash) = \Omega(n^2)$.

5 Conclusions and Open Questions

We have proven a strong connection between the black pebble game and the Monomial Calculus proof system by showing that the degree and size bounds required simultaneously in a MC refutation of the pebbling formula for a DAG G closely correspond to the number of pebbles and the time in a pebbling strategy for G. This improves the known relations between the complexities of pebble games and algebraic proof systems and implies strong degree-size tradeoffs for the MC system as well as degree separations between NS, MC and PC.

We have also shown that the variable space measure for the refutation of pebbling formulas in the three systems PC, MC and NS exactly corresponds to the number of pebbles in the black, black-white and reversible games. From this equivalence we obtain variable space separations between the proof systems.

It is open whether these separations are optimal or can be improved using other techniques. Finding out what is the optimal degree separation between the NS and MC proof systems is another interesting open question.

References

1 Albert Atserias and Joanna Fijalkow. Definable ellipsoid method, sums-of-squares proofs, and the graph isomorphism problem. *SIAM J. Comput.*, 52(5):1193–1229, 2023. doi:10.1137/20m1338435.

2 Albert Atserias and Elitza N. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. *SIAM Journal on Computing*, 42(1):112–137, January 2013. Preliminary version in *ITCS '12*. doi:10.1137/120867834.

3 Paul Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science (FOCS '94)*, pages 794–806, 1994. doi:10.1109/SFCS.1994.365714.

4 Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 813–822, May 2013. doi:10.1145/2488608.2488711.

5 Eli Ben-Sasson. Size space tradeoffs for resolution. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 457–464, May 2002. doi:10.1145/509907.509975.

6 Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow – Resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*. doi:10.1145/375827.375835.

7 Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989. doi:10.1137/0218053.

8 Christoph Berkholz and Martin Grohe. Limitations of algebraic approaches to graph isomorphism testing. In *ICALP 2015*, volume 9134 of *Lecture Notes in Computer Science*, pages 155–166. Springer, 2015. doi:10.1007/978-3-662-47672-7_13.

9 Joshua Buresh-Oppenheim, Matthew Clegg, Russell Impagliazzo, and Toniann Pitassi. Homogenization and the polynomial calculus. *Computational Complexity*, 11(3-4):91–108, 2002. Preliminary version in *ICALP '00*. doi:10.1007/s00037-002-0171-6.

10 Siu Man Chan, Massimo Lauria, Jakob Nordström, and Marc Vinyals. Hardness of approximation in PSPACE and separation results for pebble games (Extended abstract). In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15)*, pages 466–485, 2015. doi:10.1109/FOCS.2015.36.

11 Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 174–183, May 1996. doi:10.1145/237814.237860.

12 Stephen A. Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13(1):25–37, 1976. Preliminary version in *STOC '74*. doi:10.1016/S0022-0000(76)80048-7.

13 Susanna F. de Rezende, Or Meir, Jakob Nordström, Toniann Pitassi, Robert Robere, and Marc Vinyals. Lifting with simple gadgets and applications to circuit and proof complexity. *CoRR*, abs/2001.02144, 2020. doi:10.48550/arXiv.2001.02144.

14 Susanna F. de Rezende, Or Meir, Jakob Nordström, and Robert Robere. Nullstellensatz size-degree trade-offs from reversible pebbling. *Comput. Complex.*, 30(1):4, 2021. doi:10.1007/s00037-020-00201-y.

15 Erich Grädel, Martin Grohe, Benedikt Pago, and Wied Pakusa. A finite-model-theoretic view on propositional proof complexity. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:4)2019.

16 Martin Grohe and Wied Pakusa. Descriptive complexity of linear equation systems and applications to propositional proof complexity. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005081.

17 Alexander Hertel. *Applications of Games to Propositional Proof Complexity*. PhD thesis, University of Toronto, May 2008. Available at http://www.cs.utoronto.ca/~ahertel/. URL: http://hdl.handle.net/1807/16735.

18 Neil Immerman. Number of quantifiers is better than number of tape cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981. doi:10.1016/0022-0000(81)90039-8.

19 Russell Impagliazzo, Pavel Pudlák, and Jiří Sgall. Lower bounds for the polynomial calculus and the Gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999. doi:10.1007/s000370050024.

20 Balasubramanian Kalyanasundaram and George Schnitger. On the power of white pebbles. *Combinatorica*, 11(2):157–171, June 1991. Preliminary version in *STOC '88*. doi:10.1007/BF01206359.

21 Friedhelm Meyer auf der Heide. A comparison of two variations of a pebble game on graphs. *Theoretical Computer Science*, 13(3):315–322, 1981. doi:10.1016/S0304-3975(81)80004-7.

22 Jakob Nordström. Pebble games, proof complexity and time-space trade-offs. *Logical Methods in Computer Science*, 9:15:1–15:63, September 2013. doi:10.2168/LMCS-9(3:15)2013.

23 Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. Manuscript in preparation. Current draft version available at http://www.csc.kth.se/~jakobn/research/PebblingSurveyTMP.pdf, 2015.

24 Michael S. Paterson and Carl E. Hewitt. Comparative schematology. In *Record of the Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127, 1970. doi:10.1145/1344551.1344563.

25 Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game on graphs. *Mathematical Systems Theory*, 10:239–251, 1977. doi:10.1007/BF01683275.

26 Nicholas Pippenger. Pebbling. Technical Report RC8258, IBM Watson Research Center, 1980.

27 Alexander A. Razborov. On space and depth in resolution. *Computational Complexity*, 27(3):511–559, 2018. doi:10.1007/s00037-017-0163-1.

28 John E. Savage. *Models of computation - exploring the power of computing*. Addison-Wesley, 1998.

29 Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*, volume 3 of *Mathematics for Applications (Mathematik für Anwendungen)*. Lehmanns Media, 2013.

30 Jacobo Torán and Florian Wörz. Reversible pebble games and the relation between tree-like and general resolution space. *Comput. Complex.*, 30(1):7, 2021. doi:10.1007/s00037-021-00206-1.

31 Robert E. Wilber. White pebbles help. *Journal of Computer and System Sciences*, 36(2):108–124, 1988. Preliminary version in *STOC '85*. doi:10.1016/0022-0000(88)90023-2.

Punctual Presentability in Certain Classes of Algebraic Structures

Dariusz Kalociński
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Luca San Mauro
Department of Philosophy, University of Bari, Italy

Michał Wrocławski
Faculty of Philosophy, University of Warsaw, Poland

Abstract

Punctual structure theory is a rapidly emerging subfield of computable structure theory which aims at understanding the primitive recursive content of algebraic structures. A structure with domain ℕ is punctual if its relations and functions are (uniformly) primitive recursive. One of the fundamental problems of this area is to understand which computable members of a given class of structures admit a punctual presentation. We investigate such a problem for a number of familiar classes of algebraic structures, paying special attention to the case of trees, presented both in a relational and functional signature.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases fully primitive recursive structures, punctual presentability, trees, injection structures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.65

Funding *Dariusz Kalociński*: supported by the National Science Centre Poland grant under the agreement no. 2023/49/B/HS1/03930.
Luca San Mauro: is a member of INDAM-GNSAGA.
Michał Wrocławski: supported by the National Science Centre Poland grant under the agreement no. 2023/49/B/HS1/03930.

Acknowledgements We would like to thank the reviewers as well as Nikolay Bazhenov, Ivan Georgiev and Stefan Vatev for helpful discussions.

1 Introduction

Computable structure theory is a vast research program which aims at analyzing the algorithmic content of algebraic structures through the tools of computability theory (for an excellent introduction to this field, see [15]). The fundamental concept of when a structure is computably presented dates back to the seminal work of Mal'cev [14] and Rabin [16] in the 1960s: A structure with domain the set ℕ of the natural numbers is computable if its relations and functions are uniformly Turing computable. Then, a countably infinite structure is computably presentable (or, it has computable copy) if it is isomorphic to some computable structure.

A classic problem in computable structure theory is to understand, for familiar classes of structures $\mathfrak{K}$, which members of $\mathfrak{K}$ are computably presentable. Sometimes the answer is that *every* member of $\mathfrak{K}$ has a computable copy: this is the case for, e.g., vector spaces over ℚ and algebraically closed fields of a given characteristic. On the other hand, it is an immediate consequence of Tennebaum's theorem that, among models of Peano Arithmetic, only one is computably presentable: the standard model. Yet, in most cases, characterizing the computable members of $\mathfrak{K}$ is a delicate task, which often requires to individuate nice invariants for $\mathfrak{K}$ (if any) and to determine how hard is to compute such invariants.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 65; pp. 65:1–65:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A parallel endeavour is to explore when a computable algebraic structure has a feasible presentation. The problem can be formalized in a number of different ways leading, e.g., to the study of algebraic structures presented by finite state automata [13], or to polynomial-time algebra [7, 1]. Kalimullin, Melnikov, and Ng [12] initiated the systematic study of punctual presentations, that lies somewhere in the between of computability and complexity theory:

▶ **Definition 1.** *A structure with domain* $\mathbb{N}$ *is* punctual *(or, fully primitive recursive) if its relations and functions are uniformly primitive recursive.*

Then, clearly, a structure is punctually presentable, if it is isomorphic to some punctual structure. Intuitively – and by relying on a sort of restricted Church-Turing thesis – a structure is punctual, if there is algorithmic with no unbounded search which is able to decide any quantifier-free question about the structure (that is, for any such question, one knows in advance how much time is needed to compute an answer).

Punctual structure theory rapidly emerged as an intriguing subfield of computable structure theory [10, 11, 4, 3] and it also serves as a theoretical underpinning for the study of online algorithms [2] (i.e., algorithms in which the input is received and processed piece by piece without having access from the start to the complete problem data).

One of notable results is that the index set of computable structures that are punctually presentable is Σ^1_1-complete [5]. However, if we restrict the class of structures is some natural way, it may be the case that every computable member from that class is punctually presentable (and thus the corresponding index set is trivial).

▶ **Definition 2.** *We say that a class of structures* $\mathfrak{K}$ *is* punctually robust*, if every computable member of* $\mathfrak{K}$ *admits a punctual presentation.*

In [12], it is proven that the following classes of structures are punctually robust: equivalence structures, linear orders, torsion-free abelian groups, Boolean algebras, and abelian p-groups; on the other hand, there are computable undirected graphs, computable torsion abelian groups, and computable Archimedean ordered abelian groups with no punctual copy. In this work, we contribute to this line of research, exhibiting new examples of both classes that are punctually robust and classes that are not.

Kalimullin, Melnikov, and Ng [12] showed that there exists a computable undirected graph with no punctual presentation. However, many natural classes of graphs admit such presentations. In Section 3, using a technique introduced in [12] for showing that equivalence structures are punctually presentable, we isolate one such class.

▶ **Proposition 3.** *Every computable digraph with an infinite semi-transversal is punctually presentable.*

For structures with a functional signature, the situation may change drastically. One of the simplest types of purely functional structures are the so-called injection structures. They are of the form $\mathcal{A} = (\mathbb{N}, f)$, where f is 1-1. Injection structures have already been considered in the computable setting (see, e.g., [6]) and in punctual structure theory with an emphasis on punctual categoricity in [10]. In Section 4 we prove the following:

▶ **Theorem 4.** *The class of injection structures is not punctually robust.*

The structure witnessing that injections structures are not punctually robust will consist only of cycles. One can observe that the relational counterpart of this structure is punctually presentable (this follows from Proposition 3). We supplement Theorem 4 by a list of sufficient conditions for an injection structure to be punctually presentable, as well as by a list of conditions equivalent to their punctual presentability. These supplementary results are omitted due to space constraints and will appear in the extended version.[1]

A similar contrast emerges for trees. We show that the punctual robustness (or, lack thereof) of the class of trees depends on how they are presented. More precisely, in Section 5 we consider the notion of a tree represented as a function mapping each child to its parent and looping back at the root. We then prove the main result of the paper:

▶ **Theorem 5.** *The class of functional trees is not punctually robust.*

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N}$. We let $[a, b) = \{x \in \mathbb{N} : a \le x < b\}$. We assume a fixed computable enumeration $p_0, p_1, \dots$ of all primitive recursive unary functions. We abbreviate *primitive recursive* as *p.r.* By f^n be denote the n-fold composition of f with itself. If F is a partial function or a relation, χ_F denotes a graph of F. A structure is any tuple $\mathcal{A} = (A, (R_i)_{i\in I}, (f_j)_{j\in J}, (c_k)_{k\in K})$, where $A \neq \emptyset$ is the domain of $\mathcal{A}$, in symbols $dom(\mathcal{A})$, while R_i, f_j, c_k are relations, functions, constants on A. By $\mathcal{A} \cong \mathcal{B}$ we mean that $\mathcal{A}$ and $\mathcal{B}$ are isomorphic, and by $\mathcal{A} \hookrightarrow \mathcal{B}$ that $\mathcal{A}$ embeds into $\mathcal{B}$. All structures in this article are countably infinite, except finite structures that appear in constructions.

▶ **Definition 6** (computable structure)**.** *A structure is a computable if its domain, as well as all relations, functions, and constants from its signature are uniformly computable.*

Punctual structures were defined in the introduction (Definition 1).

Notice that $(\mathbb{N}, p_0), (\mathbb{N}, p_1), \dots$ is a computable enumeration of all punctual structures with just one unary functional symbol.

▶ **Definition 7** (punctual presentability)**.** *A structure $\mathcal{A}$ is a copy (presentation) of $\mathcal{B}$ if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. We say that $\mathcal{A}$ is punctually presentable if $\mathcal{A}$ has a punctual presentation.*

3 Directed graphs

In this section, we individuate a class of computable directed graphs (from now on, digraphs) which are punctually presentable. For a digraph $G = (\mathbb{N}, E_G)$, we denote by L_G the collection of G-nodes with loops, i.e., $L_G := \{x : (x, x) \in E_G\}$. We denote by $G \restriction_c$ the restriction of G to the initial segment $\{x : 0 \le x \le c\}$.

▶ **Definition 8** (semi-transversal)**.** *Let $G = (\mathbb{N}, E_G)$ be a computable digraph. The* semi-transversal *of G, denoted as τ_G, is the collection of numbers that are not adjacent to any smaller number, i.e.,*

$$\tau_G := \{x : (\forall y < x)(\{(y, x), (x, y)\} \cap E_G = \emptyset)\}.$$

It is immediate to observe that τ_G is computable, whenever G is computable. As aforementioned, the following result elaborates on ideas presented in [12] when dealing with equivalence structures. The proof may serve as a gentle introduction to the way of thinking about punctual copies and as a warm-up to the more complex constructions of this paper.

[1] Following a suggestion from one of the reviewers, the authors have realized that some results on injection structures were obtained earlier [8]. Essentially, Theorem 4 should be attributed to Cenzer and Remmel (see, Lemma 3.12 and Theorem 3.13 in [8]). For completeness of the presentation, we include our proof of this theorem in the punctual setting.

▶ **Proposition 3.** *Every computable digraph with an infinite semi-transversal is punctually presentable.*

Proof. Let $G = (\mathbb{N}, E_G)$ be a computable digraph so that τ_G is an infinite set. For the sake of exposition, we assume that both $L_G \cap \tau_G$ and $(\mathbb{N} \setminus L_G) \cap \tau_G$ are infinite (the other cases are treated similarly and are somehow simpler). We will build by stages a punctual copy P_G of G and a bijection $f : \mathbb{N} \to \mathbb{N}$ witnessing that $G \cong P_G$.

The underlying idea of the proof is rather straightforward: At any given stage of the construction, we monitor a finite fragment of G. In particular, we aim at determining if a given number c belongs to τ_G. This information is computable but, in general, not primitive recursive. So to ensure that P_G will be punctual, while waiting to know if $c \in \tau_G$, we let several numbers x to be inactive (meaning that such numbers will belong to the semi-transversal of P_G). If we see that $c \notin \tau_G$, we extend the desired isomorphism by letting $f(c)$ be a fresh number and ensuring that $G \restriction_c$ embeds into the active part of P_G. On the other hand, if $c \in \tau_G$, we will let $f(c)$ be a suitable inactive number z (which returns active right after this action). Let us now be formal.

Construction

To record which fragment of G is currently monitored, we use a parameter c that will be updated as the construction proceeds. Moreover, during the construction some numbers x will be marked as *inactive*: specifically, by declaring x to be *no loop-inactive* (*NL*-inactive), we let x be non-adjacent (in P_G) to any number $\leq x$; by declaring x as *loop-inactive* (L-inactive), we let x non-adjacent (in P_G) to any number $< x$ but we also let $(x, x) \in P_G$. *Active* numbers are those that are neither *NL*-inactive nor L-inactive.

At stage 0, all numbers in $\mathbb{N}$ are active; we let c be 0 and we immediately move to the next stage. At all stages $s > 0$, we see if the following condition holds:

$$(\forall x \leq c)(\chi_{E_G}(x, c)\downarrow \text{ and } \chi_{E_G}(c, x)\downarrow \text{ in less than } s \text{ stages}). \tag{$\star$}$$

If $(\star)$ is not met, we declare $2s$ to be *NL*-inactive and $2s + 1$ to be L-inactive. Then, we move to the next stage. Otherwise, if $(\star)$ is met, we can determine whether c belongs to τ_G. Then, we distinguish two cases:

1. If $c \notin \tau_G$, we define $f(c) = 2s$ and we mirror an initial segment of the G-neighborhood of c by forming the P_G-edges that are needed to ensure that, for all $x \leq c$, the following equation holds:

$$(x, c) \in E_G \Leftrightarrow (f(x), f(c)) \in E_{P_G} \ \&\ (c, x) \in E_G \Leftrightarrow (f(c), f(x)) \in E_{P_G}. \tag{$\dagger$}$$

   Finally, we declare $2s + 1$ *NL*-inactive.
2. We distinguish two subcases:
   a. If $c \in L_G \cap \tau_G$, we define $f(c) = z$ for the least number z which is L-inactive, and we mark z as active (if there is no such number, we define $f(c) = 2s + 1$). Then, we declare $2s$ to be NL-inactive.
   b. If $c \in (\mathbb{N} \setminus L_G) \cap \tau_G$, the situation is symmetric: We define $f(c) = z$ for the least number z which is NL-inactive, and we mark z as active (if there is no such number, we define $f(c) = 2s$), and we declare $2s + 1$ to be L-inactive.

In both cases 1. and 2., we increase c by one and we move to the next stage.

Verification

From the construction, it follows immediately that P_G is punctual: indeed, to decide if $(u,v) \in E_{P_G}$, for $u < v$, it suffices to run the construction until stage $\lfloor v/2 \rfloor$ and each stage s requires at most s steps to be completed. To see that P_G is isomorphic to G, we begin by arguing that f is bijection.

▶ **Lemma 9.** *The function $f : \mathbb{N} \to \mathbb{N}$ is a bijection.*

Proof. Say that a stage s is *expansionary*, if at stage s the condition $(\star)$ holds. It is not hard to see that there are infinitely many expansionary stages: indeed, since G is computable, for any choice of the parameter c, there will a stage at which we are able to entirely compute $G \restriction_c$. Now, observe that the parameter c starts as 0 and is increased by one whenever $f(c)$ is defined. Thus, f is total.

Next, let $u < v$ be numbers on which f is defined at stages $s_u < s_v$. By construction, $f(u) \in \{2s_u, 2s_{u+1}, z\}$, for some z which is *NL*-inactive or L-inactive at the beginning of stage s_u; on the other hand, $f(v) \in \{2s_v, 2s_{v+1}, z'\}$, for some z' which is NL-inactive or L-inactive at stage s_v. Observe that $z \neq z'$, since z returns active as soon as it enters the range of f. Thus, f is injective.

To deduce that f is surjective, it suffices to prove that all numbers are eventually active: indeed, if a number $z \in \{2s, 2s+1\}$ is active at all stages, then s must an expansionary stage at which z enters the range of f; similarly, if at stage s some number z returns active, after being *NL*-inactive or L-inactive, then z simultaneously enters the range of f. Towards a contradiction, suppose there is a least number u which is declared, say, L-inactive and eventually remains so. Since $L_G \cap \tau_G$ is infinite, then there must be a stage s at which the condition $(\star)$ is met and we perform action 2.a, by which z (being the least L-inactive number) enters the range of f, contradicting our hypothesis. Hence, f is surjective. ◀

Finally, we shall prove that the bijection f yields an isomorphism from G to P_G. Suppose that, for a pair of numbers $u < v$,

$$(u,v) \in E_G \not\Leftrightarrow (f(u), f(v)) \in E_{P_G}.$$

Let s_u and s_v be the stages at which $f(u)$ and $f(v)$ are, respectively, defined. By construction, $s_u < s_v$, so that $f(u)$ is already defined at stage s_v. Moreover, at stage s_v, we decide whether $(f(u), f(v)) \in E_{P_G}$: if $v \notin \tau_G$, then condition (†) ensures that $(f(u), f(v)) \in E_{P_G}$ if and only if $(u,v) \in E_G$; on the other hand, if $v \in \tau_G$, we have that $(u,v) \notin E_G$, but $f(v)$ is chosen from the inactive numbers so that $f(u)$ is non-adjacent to $f(v)$. Similarly, one proves that, for all u, $(u,u) \in E_G$ if and only $(f(u), f(u)) \in E_{P_G}$. So f is an isomorphism from G to E_{P_G}. ◀

As an immediate corollary of Proposition 3, one obtains that the following classes of algebraic structures $\mathfrak{K}$ are punctually robust: equivalence structures, strongly locally finite graphs, and – more generally – graphs with infinitely many components. In fact, Proposition 3 may serve to obtain the punctual robustness of the classes of locally finite graphs and of graph-theoretic trees (the analog result holds in the setting of feasible presentations, see Cenzer and Remmel [9]). Let us give a brief informal discussion of these cases.

Recall that a *dominating* set for a graph $G = (V_G, E_G)$ is a set $D \subseteq V_G$ so that all vertices of $V_G \setminus D$ has a neighbor in D. The *domination number* $\gamma(G)$ is the minimum size of a dominating set for G. It is simple to show that every computably presentable graph G with $\gamma(G) = \infty$ has a computable copy H with an infinite semi-transversal. Hence, by

Proposition 3, all computable graphs with infinite domination numbers admit a punctual copy. This comprises the class of infinite but locally finite graphs, as is shown by a simple application of the pigeonhole principle.

We conclude the section by observing that the class of graph-theoretic trees, (i.e., acyclic graphs) is also punctually robust. Let T be a such tree. If T has infinite domination number, then T must be punctually presentable. On the other hand, if $\gamma(T) < \infty$, then it must contain a vertex with infinite neighborhood; a standard construction proves that such T is punctually presentable (we omit the details for space reasons). In contrast, in Section 5 we will show that there is a computable functional tree which admits no punctual copy.

4 Injection structures

▶ **Definition 10** (injection structure). *(A, f) is an* injection structure*, if $f : A \to A$ is injective.*

Let $f : A \to A$. A finite sequence of pairwise distinct elements $a_1, ..., a_n \in A$ is an f-cycle of length n if, for each $i = 1, ..., n-1$, $f(a_i) = a_{i+1}$, and also $f(a_n) = a_1$. If f is clear from the context, we may refer to a cycle without specifying f.

▶ **Definition 11** (cyclic injection structure). *$f : A \to A$ is* cyclic*, if f is injective and every element of A belongs to some f-cycle. $\mathcal{A} = (\mathbb{N}, f)$ is a* cyclic injection structure*, if f is cyclic. Such $\mathcal{A}$ is called* simple*, if $\mathcal{A}$ contains at most one cycle of length l, for all $l \in \mathbb{N}$.*

▶ **Definition 12** ($\mathbb{N}$-chain, $\mathbb{Z}$-chain). *Let $f : A \to A$. If $(a_n)_{n \in I}$ is an indexed family of elements of A such that $a_i \neq a_j$ for all $i \neq j \in I$, and $f(a_n) = a_{n+1}$ for all $n \in I$, then we call (a_n) an* $\mathbb{N}$-chain *if $I = \mathbb{N}$ and there is no such x that $f(x) = a_0$, and a* $\mathbb{Z}$-chain *if $I = \mathbb{Z}$.*

If $f : \mathbb{N} \to \mathbb{N}$ is an injection, then every $n \in \mathbb{N}$ belongs to a cycle, an $\mathbb{N}$-chain, or a $\mathbb{Z}$-chain. In the remaining part of this section we prove the following:[2]

▶ **Theorem 4.** *The class of injection structures is not punctually robust.*

We construct by stages a computable simple cyclic injection structure $\mathcal{A} = (\mathbb{N}, f)$ not isomorphic to any $(\mathbb{N}, p_i)$. For each $i \in \mathbb{N}$, we have the following requirement:

$$\begin{aligned} R_i :\ & \text{if } (\mathbb{N}, p_i) \text{ is a simple cyclic injection structure, then there is} \\ & n \in \mathbb{N} \text{ so that } p_i \text{ has a cycle of length } n \text{ but } f \text{ does not.} \end{aligned}$$

Initially, for $\mathcal{A} = \emptyset$, and all requirement are inactive. At a given stage we have finitely many active requirements. Active requirements may later be deactivated: deactivating a requirement R_i will guarantee that R_i is, and will remain, satisfied. Some requirements R_i may stay eventually active, but we will argue that in that case they are satisfied vacuously (i.e., the antecedent of R_i is false).

[2] As a side remark, Theorem 3.16 from [8] says that *any* computable injective structure has a polynomial-time computable copy with the domain equal to $Bin(\omega)$, the set of all binary representations of natural numbers, or to $Tal(\omega)$, the set of all unary (tally) representations of natural numbers; call such a copy *fully polynomial-time*. The statement of Theorem 3.16 [8] turns out to be too general because the structures constructed in the proofs of our Theorem 3, or Theorem 3.13 [8] for that matter, are computable injection structures with no punctual presentation, and hence they cannot have fully polynomial-time copies either (as any fully polynomial-time structure is punctual). However, a slight weakening of the statement of Theorem 3.16 [8] can be achieved, namely: every computable injection structure which is not cyclic is punctually presentable. We omit the proof due to space constraints.

Stage s of the construction. Let $R_{i_1}, \dots, R_{i_k}$ be the list of active requirements, and let $c_{i_1}, \dots, c_{i_k}$ be the corresponding witnesses. If there is a fresh active requirement in the list, execute the strategy for the fresh requirement (see below). Then, for each active non-fresh requirement, execute the corresponding strategy (also see below). If the list of active requirements is empty or, during stage s, we have not discovered any cycle of length s while executing strategies for fresh or non-fresh R_{i_j}, then we add a fresh (i.e., composed from distinct least numbers $a_1, a_2, \dots, a_s \notin \mathcal{A}$) cycle of length s to $\mathcal{A}$ and add the least inactive requirement to the list of active requirements and call it fresh.

Strategy for non-fresh $R_{i,j}$. Compute $c_{i_j}, p_{i_j}(c_{i_j}), p^2_{i_j}(c_{i_j}), \dots, p^{s-1}_{i_j}(c_{i_j})$. The following outcomes are possible:

1. elements of the sequence are pairwise different from each other and then we shall call such a sequence a straight line of length s,
2. there is some l such that $1 \le l \le s-1$ and $p^l_{i_j}(c_{i_j}) = c_{i_j}$ and then for the least such l, we shall call $c_{i_j}, p_{i_j}(c_{i_j}), p^2_{i_j}(c_{i_j}), \dots, p^l_{i_j}(c_{i_j})$ a cycle of length l,
3. there are some l and l' such that $1 \le l < l' \le s-1$ and $p^l_{i_j}(c_{i_j}) = p^{l'}_{i_j}(c_{i_j})$ and then for the least such l' we shall call such sequence a cycle with a tail of length $l'+1$.

If precisely after $s-1$-th iteration of function p_{i_j} on its corresponding witness we close a cycle or a cycle with a tail, then we deactivate R_{i_j}.

Strategy for fresh $R_{i,j}$. We define α_p to be the sequence $a_p, p_{i_j}(a_p), p^2_{i_j}(a_p), \dots, p^{s-1}_{i_j}(a_p)$, for $p = 1, \dots s$, where a_p is the least number which is currently not in the domain of $\mathcal{A}$ and has not appeared anywhere in calculations of any of the sequences α_p.

Now we check if any of the following outcomes occurs and act accordingly:

- if some α_p is a line of length s, set $c_{i_j} := a_p$ as the witness for R_{i_j} and call R_{i_j} non-fresh,
- if some α_p is a cycle of length s or a cycle with a tail of any length, deactivate R_{i_j},
- if for every α_p we obtained a cycle of length smaller than s, deactivate R_{i_j}.

Verification

▶ **Lemma 13.** *$\mathcal{A}$ is a computable simple cyclic injection structure.*

Proof. Clearly, $\mathcal{A}$ is computable. It is a cyclic injection structure since whenever we add new elements to the structure, they are in a cycle. A cycle of length s can only be added at stage s and only once. Hence, the structure is simple. ◀

▶ **Lemma 14.** *If a requirement R_i is deactivated during some stage s, then from that moment it always remains satisfied.*

Proof. There are several possible cases of how R_i was deactivated at stage s. Below we consider all of them:

- We discovered a cycle with a tail generated by the function p_i starting from its corresponding witness. In this case R_i is satisfied because p_i is not a cyclic function,
- We discovered a cycle of length s generated by p_i during stage s. In this case R_i is satisfied because we do not have a cycle of such length in $\mathcal{A}$ and hence $\mathcal{A}$ is not isomorphic to $(\mathbb{N}, p_i)$.
- We tried s different witnesses $a_1, \dots, a_s$ for R_i and for each potential witness a we generated a cycle of length shorter than s starting with a. As a consequence of the pigeon-hole principle it is necessary that there are some witnesses a and b generating cycles of the same length. Furthermore, whenever we choose a new witness, we take a number which has not appeared anywhere in earlier calculations. Hence, the cycle containing a and the cycle containing b are not the same and they are both in $(\mathbb{N}, p_i)$. We conclude that this is not a simple cyclic injection structure and hence R_i is satisfied. ◀

▶ **Lemma 15.** *If a requirement R_i is not deactivated at any stage, then it is satisfied.*

Proof. We assume that R_i was activated at some stage s. Then at stage $s+1$ it acted as a fresh requirement and it was deactivated unless we discovered a witness c_i such that the sequence $c_i, p_i(c_i), p_i^2(c_i), \dots, p_i^s(c_i)$ is a straight line of length $s+1$. In that case, at every stage $t \geq s+1$, requirement R_i is deactivated if and only if we discover a cycle or a cycle with a tail of length t starting with c_i. If this does not happen at any stage, then the line starting with c_i is becoming longer at every stage and never closes. Hence, $(\mathbb{N}, p_i)$ contains an $\mathbb{N}$-chain with an element c_i. So this structure is not cyclic and R_i is satisfied. ◀

▶ **Lemma 16.** *The domain of $\mathcal{A}$ is $\mathbb{N}$.*

Proof. Whenever we add a new element to $\mathcal{A}$, it is the least natural number which is currently not there. It follows that the domain of $\mathcal{A}$ is either all of $\mathbb{N}$ or some initial segment of it.

It remains to show that the $dom(\mathcal{A})$ is infinite. Suppose not. Let a be the largest (as a number) element in $\mathcal{A}$. Suppose that a was added at some stage s and that at the end of stage s the only active requirements were $R_{i_1}, \dots, R_{i_k}$. We consider stages $s+1, \dots, s+i_k+1$. We observe that there are i_k+1 of them and this is more than the number of active requirements. Since each requirement gets deactivated at most once, it follows that there is some t such that $s+1 \leq t \leq s+i_k+1$ such that none of the requirements is deactivated at stage t. This implies that at stage t we have not discovered any new cycle of length t. Hence at stage t we add a cycle of length t to $\mathcal{A}$ and this cycle consists of new elements larger than a. Hence a is not the largest natural number in $\mathcal{A}$ contrary to our assumption. ◀

▶ **Lemma 17.** *$\mathcal{A}$ is not punctually presentable.*

Proof. Suppose not. Then $\mathcal{A} \cong (\mathbb{N}, p_i)$, for some $i \in \mathbb{N}$. By Lemma 13, $\mathcal{A}$ is a simple cyclic injection structure. By Lemmas 14 and 15, R_i is satisfied. Since $(\mathbb{N}, p_i)$ is also a simple cyclic injection structure, there is some cycle length occurring in p_i but not in f. This contradicts the assumption that $(\mathbb{N}, f) \cong (\mathbb{N}, p_i)$. Hence, $\mathcal{A}$ is not punctually presentable. ◀

5 Functional trees

In this section we prove the main result of the paper:

▶ **Theorem 5.** *The class of functional trees is not punctually robust.*

▶ **Definition 18** (functional tree)**.** *Let $A \neq \emptyset$ be a set and let $T : A \to A$. (A, T) is a* functional tree*, if there is a unique r such that $T(r) = r$, and for every $x \in A$ there exists $i \in \mathbb{N}$ such that $T^i(x) = r$. The unique r is called the root, and is denoted by $r(A, T)$.*

Before we start, we need several technical notions.

First, observe that a functional tree (P, T) may be viewed as a partial order $(P, \leq)$ defined as follows: $x \leq y \Leftrightarrow \exists i \in \mathbb{N}\, T^i(x) = y$. This allows us to use a convenient order-theoretic notation when speaking about trees. However, we should keep in mind that this is just a manner of speaking and that in this section we deal with functional trees. Given a partial order $(P, \leq)$, we define $P_{\leq x} = \{y \in P : y \leq x\}$ and $P_{\geq x} = \{y \in P : y \geq x\}$. The corresponding strict partial order is denoted by $<$. Elements $x, y \in P$ are adjacent, $Adj(x, y)$, if and only if $x < y \wedge \neg \exists z \in P\, x < z < y$. To avoid confusion, we sometimes write $\leq_T$ or Adj_T to indicate that the ordering or adjacency relation is induced by T.

▶ **Definition 19** (branching node, branching, binary branching). *$x \in T$ is a* branching node *of T (or x branches in T) if it has at least two children in T. Such an x induces a unique subtree of T, called a branching and defined as $br(x, T) = \{y \in T : Adj(y, x)\} \cup T_{\geq x}$. If x is a branching node, we define $|br(x, t)|$, the length of $br(x, t)$, as the length of $T_{\geq x}$. By a* binary branching *we mean a tree with exacly two leaves sharing a parent.*

▶ **Definition 20** (uniquely branching tree). *We say that a tree T is uniquely branching if for every $n \in \mathbb{N}$, there exists at most one branching node $x \in T$ such that $|T_{\geq x}| = n$.*

▶ **Definition 21** (level). *We say that a* branching *node $x \in T$ belongs to the level n of T if the set $T_{>x}$ contains precisely n branching nodes. We denote the level n of T by $T[n]$. We sometimes refer to the members of $T[n]$ as n-level nodes.*

Keep in mind that we number levels $0, 1, \dots$. Therefore, the first level is level 0. Level $T[n]$ may be empty but if $T[n] \neq \emptyset$ then $T[k] \neq \emptyset$, for $k < n$.

▶ **Definition 22** (i-level subtree). *Let T be a tree such that $T[i] \neq \emptyset$. We define $T[\leq i]$ as the least subtree of T containing all nodes at levels $\leq i$ together with their children.*

Notice that $T[\leq i]$ is the sum of the branchings $br(x, T)$ for all $x \in T[j]$ such that $j \leq i$.

Recall that a binary tree is a tree in which every internal node has at most two children. In a proper binary tree, every internal node has exactly two children.

▶ **Lemma 23.** *Let $(T, \leq)$ be a finite proper binary tree and let $F \subseteq T$ be such that, at every level of T except the first, at most one node is in F. Then there exists a leaf $x \in T$ such that $T_{\geq x} \cap F = \emptyset$.*

Proof. By assumption, $r(T) \notin F$. Suppose we have a path $x_n, x_{n-1}, \dots, x_0 = r(T)$ such that $x_i \notin F$, for $i = 0, 1, \dots, n$. If x_n is a leaf, we are done. If not, x_n has two children. These two children are at the same level and one of them is outside F. Let x_{n+1} to be a child outside F. ◁

▶ **Definition 24** (attaching a tree to a leaf). *Let $T, \hat{T}$ be disjoint finite trees and let $z \in T$ be a leaf. T' is obtained from T by attaching $\hat{T}$ to z in T if $dom(T') = dom(T) \cup (dom(\hat{T}) \setminus \{r(\hat{T})\})$ and $x, y \in dom(T')$ satisfy $Adj_{T'}(x, y)$ iff $Adj_T(x, y) \vee Adj_{\hat{T}}(x, y) \vee Adj_{\hat{T}}(x, r(\hat{T})) \wedge y = z$.*

▶ **Definition 25** (functional semitree). *Let A be a finite set and let $T : A \to A$ be a partial function. We say that (A, T) is a* functional semitree *if there is a unique r such that $T(r) = r$, and $(A, \chi_T \setminus \{(r, r)\})$ is an acyclic directed graph. The unique r is called the root, and is denoted by $r(A, T)$.*

The difference between a semitree and a tree is that the former may contain nodes that are not connected to the root via any path. We will use the letters t and q, possibly with decorations, to refer to *finite* trees and to semitrees, respectively.

▶ **Definition 26** (rooted part of T). *$R(T)$ denotes the largest subtree of a semitree T.*

At every stage of the construction, the main tree T that we build is approximated by a finite tree. As we will see, the growth of T is modulated by a set of nodes F.

▶ **Definition 27** (closed node, open node). *A node $y \in T$ is closed at a given stage if $T_{\geq y} \cap F \neq \emptyset$ at that stage. A node y is open if y is not closed, that is $T_{\geq y} \cap F = \emptyset$.*

▶ **Definition 28** (leaf extension). *Let t be a tree with a distinguished set of nodes F. We say that t' is a* leaf extension *of a tree t (in symbols: $t' \sqsupseteq t$) if, for some $k > 0$, there exist disjoint (from t and from each other) finite trees $\tilde{t}_1, \ldots, \tilde{t}_k$, and pairwise distinct open leaves $x_1, \ldots, x_k \in t$, such that t' is obtained from t by simultaneously attaching each $\tilde{t}_i$ to x_i in t. We say that t' is a* proper leaf extension *of t (in symbols: $t' \sqsupset t$) if $t' \sqsupseteq$ and $t' \neq t$.*

▶ **Definition 29** (matching). *(h, t') is a* matching of q into t *(in symbols: $(h, t') : p \hookrightarrow t$), if $t' \sqsupseteq t$ and $h : R(q) \hookrightarrow t'$ is an embedding such that $img(h) \supseteq dom(t') \setminus dom(t)$. A matching (h, t') of q into t is proper, if $t' \sqsupset t$. We say that q is* (properly) matchable with t *if there exists a (proper) matching (h, t') of q into t.*

▶ **Lemma 30.** *Let P be an infinite tree and let t be a finite tree whose level n is the maximal one (that is, $t[n] \neq \emptyset$ and $t[n+1] = \emptyset$) and $t[\leq n] = t$. Let q be a finite subtree of P. If q is not matchable with t, then $P \ncong T$, for every infinite tree $T \supset t$ such that $T[\leq n] = t$.*

Proof. Suppose there is an infinite tree $T \supset t$ such that $T[\leq n] = t$ and $P \cong T$ via $h : P \to T$. Let $h(q)$ be the isomorphic image of q (hence, a subtree of T). Let $t' = T \upharpoonright dom(t) \cup dom(h(q))$. t' is a leaf extension of t. Since q is a tree, $R(q) = q$. We observe that $(h \upharpoonright q, t')$ is a matching of q into t. ◀

▶ **Definition 31** (outside branching). *Suppose q is matchable with t. We say that q* branches outside *t at x if x branches in q and $|q_{\geq x}|$ is greater than the length of every branching in t. We say that q branches outside t if it branches outside t for some x.*

▶ **Definition 32** (inside branching). *Suppose q is matchable with t. We say that q* branches inside *t at x if x branches in $R(q)$, $br(x, q)$ embeds in t and for every matching $(h, t') : q \hookrightarrow t$, $h(q_{\leq x}) \cap t' \setminus t \neq \emptyset$ (i.e., h maps some descendants of x to $t' \setminus t$).*

We are ready to start the proof of Theorem 5.

We build an infinite computable functional tree T with domain $\mathbb{N}$ such that, for all $i \in \mathbb{N}$, the following requirements R_i are satisfied: $\mathcal{T} \ncong \mathcal{P}_i$, where $\mathcal{P}_i = (\mathbb{N}, p_i)$. T will be binary, uniquely branching and it will grow only through its leaves. T will be approximated by a sequence of finite trees $T_0 \subseteq T_1 \subseteq \ldots$, with $T = \bigcup_{s \in \mathbb{N}} T_s$. Occasionally, when it does not lead to confusion, we take the liberty to use the symbol T to refer to T at a given stage s, i.e. to T_s. During the construction, we approximate $\mathcal{P}_i$ by looking at certain specific finite approximations of $\mathcal{P}_i$, which we call anticipations. We make this notion precise in Definition 34. For now, it sufficient to know that an anticipation of $\mathcal{P}_i$ simply looks at portions of $\mathcal{P}_i$ that are large compared with the current approximation of $\mathcal{T}$. We simultaneously build a dynamic set $F \subset T$. We will put elements into F but will never withdraw them.

▶ Remark 33. An approximation $\mathcal{P}_{i,s} = (A, p_i \upharpoonright A)$ of a functional tree $\mathcal{P}_i$ may actually be a semi-tree. This is because A may contain nodes which are connected to the root via nodes from $\mathbb{N} \setminus A$.

Satisfaction of R_i will be based on two strategies that we describe below. In the description of these strategies, we mention a tree t. The role of this tree will become clear when we describe the *StrategySelection* procedure. When a strategy is called, t is given as one of the inputs and used t to compute anticipations of $\mathcal{P}_i$. These anticipations depend on the number of stage s and t, hence their name: (s, t)-anticipations.

Outside-branch Strategy

We say that the outside-branch strategy for p_i is ready at stage s for a tree t if the (s, t)-anticipation q of p_i is a functional semitree and q branches outside T_{s-1}. The strategy is called only when it is ready in the specified sense. The outside-branch strategy is called with arguments q, T_{s-1}, where q is some (s, t)-anticipation of p_i that branches outside T_{s-1}.

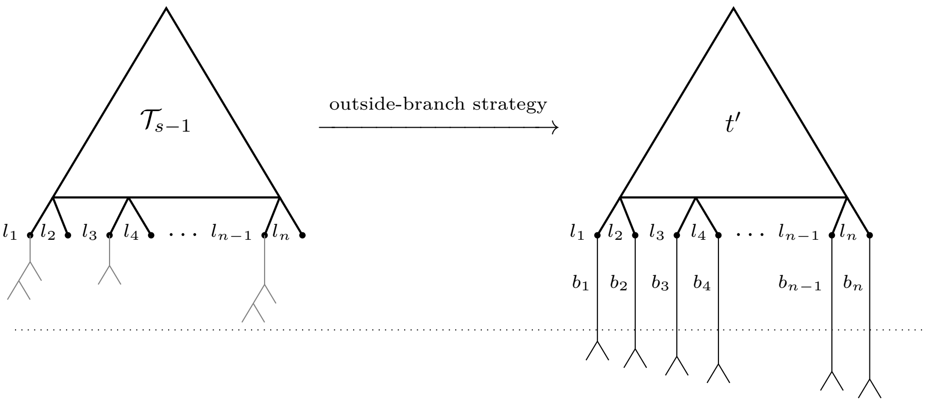


Figure 1 Outside-branch strategy.

The idea of the strategy is as follows (see, also, Figure 1). By the definition of the outside-branching, q contains a branching b that is longer than every branching in T_{s-1} (hence the word "outside-branching"). If we wanted to embed b into T, we would have to extend T_{s-1} to include the branching length of b in the extension of T_{s-1}. We purposely extend T_{s-1} to a leaf extension $t' \sqsupset T_{s-1}$ that omits the branching length of b. Recall that the construction is arranged in such a way that T grows only through its leaves and whenever we add new branchings into T, their length is greater than any branching that was in T so far. Hence, once we omit a given branching length in T, T will not have any branching of this length at all. Therefore, if we set $T_s = t'$, we would kill the potential isomorphism $\mathcal{T} \cong \mathcal{P}_i$ permanently.

More precisely, we produce t' in the following way. Let $d = max\{|q_{\geq x}| : x \text{ branches in } q\}$. Let $\{l_1, \dots, l_n\}$ be the set of all open leaves of T_{s-1}. We make binary branchings $b_1, \dots, b_n$ such that for all i, j, $|(T_{s-1})_{\geq l_i}| + |b_i| > d$ and $i \neq j \Rightarrow |(T_{s-1})_{\geq l_i}| + |b_i| \neq |(T_{s-1})_{\geq l_j}| + |b_j|$. The strategy outputs t' obtained from T_{s-1} by simultaneous attachment of each b_i to l_i. We make sure t' is extended by the least fresh numbers not occurring in T_{s-1}.

The outside-branch strategy is illustrated in Figure 1. T_{s-1} is shown on the left in black. The tree $R(q)$ (or rather a leaf extension of T_{s-1} into which $R(q)$ embeds) is shown explicitly only in part using gray subtrees on the left. The output of the outside-branch strategy is t' shown on the right which has very long branchings attached to $l_1, \dots, l_n$, the open leaves of T_{s-1}, so that t' omits some branching lengths of $R(q)$. The horizontal dotted line indicates that the lengths of the branchings below it are greater than the lengths of all the branchings in the tree on the left.

Inside-branch Strategy

We say that the inside-branch strategy for p_i is ready at stage s for a tree t if the (s, t)-anticipation q of p_i and T_{s-1} satisfy the following conditions:

$$q \text{ is a functional semitree,} \tag{1}$$

$$T_{s-1} \text{ and } R(q) \text{ have level } i+1, \tag{2}$$

$$R(q)[\leq i+1] \cong T[\leq i+1], \tag{3}$$

$$q \text{ branches inside } T_{s-1}. \tag{4}$$

The strategy is called only when it is ready in the specified sense. We call it with arguments q, T_{s-1}, where q is some (s, t)-anticipation of p_i that branches inside T_{s-1}.

The idea of the inside-branch strategy is as follows. By the definition of the inside-branching, q contains a branching node x such that $br(x, q)$ embeds into T_{s-1} (hence the word "inside-branching"). Since T_{s-1} is uniquely branching, there is only one way of embedding this branching to T_{s-1}. Suppose this unique embedding maps x to $y \in T_{s-1}$. By the definition of the inside-branching, there is no way of embedding the subtree $q_{\leq x}$ into the subtree $(T_{s-1})_{\leq y}$ without prolonging some paths in the subtree $(T_{s-1})_{\leq y}$. Hence, if we decide to put y (or some of its ancestors) into F, we would stop growing $(T_{s-1})_{\leq y}$, that is, we would have (in the limit) $T_{\leq y} = (T_{s-1})_{\leq y}$, thus killing the isomorphism $\mathcal{T} \cong \mathcal{P}_i$ permanently.

The details of the strategy follow. Suppose that the inside-branch strategy for p_i is ready and thus (1)-(4) hold. The strategy will stop the growth of T from exactly one node at level $i+1$ of T. Since q branches inside T_{s-1}, let x be a node of T_{s-1} at which q branches inside T_{s-1}. If x belongs to level $j > i+1$, then the path connecting x to the root must intersect level $i+1$ and the point of intersection is the branching node that we put into F; note that this point is unique because T is uniquely branching and (2) and (3) hold. If x belongs to level $i+1$, we simply put x to F; again, this x is unique by the same reasoning. If x belongs to level $\leq i$ but not to level $i+1$, it means that T_{s-1} did not have chance to grow up to level $i+1$ from x (and will not have such a chance anymore, because we never withdraw elements from F) and therefore we do not have to take any action.

Construction

The construction is arranged in stages $s = 0, 1, \ldots$. For $s = 0$, we set $T_0(0) = 0$ with domain $\{0\}$ and $F = \emptyset$. No R_i is satisfied at stage 0. At each subsequent stage $s > 0$, we start with a finite tree T_{s-1}. We make a call to $StrategySelection(s, T_{s-1})$ and after it finishes working, we go to the next stage. This ends the construction.

StrategySelection is a recursive procedure that selects which strategy to perform and with which arguments. It uses a specific way of growing our approximations.

▶ **Definition 34** (anticipation). *Let $p : \mathbb{N} \to \mathbb{N}$, $s \in \mathbb{N}$ and let T be a finite tree. Let $C = [0, s) \cup [0, |T| + 1)$ and $D = \bigcup_{l=0}^{H(T)+s} p^l(C)$. We say that $p \restriction D$ is the (s, T)-anticipation of p.*

We say that p_i is ready at stage s for a tree t if either the outside-branch or the inside-branch strategy for p_i is ready at stage s for t.

StrategySelection(n, t). Check whether there exists an unsatisfied R_i with $i < n$ such that p_i is ready at stage s for t. If there is no such i, return from the current call to *StrategySelection* with permission. Otherwise, let $k < n$ be the least such i.

If the (s, t)-anticipation of p_k branches inside T_{s-1}, perform the inside-branch strategy with the (s, t)-anticipation of p_k, T_{s-1} as arguments (we say that p_k triggers the inside-branch strategy). Declare R_i as satisfied and return from the current call to *StrategySelection* without permission.

If it is not the case that the (s, t)-anticipation of p_k branches inside T_{s-1}, then the (s, t)-anticipation of p_k branches outside T_{s-1}. In that case we let t' be the output of the outside-branch strategy with the (s, t)-anticipation of p_k and T_{s-1} as arguments, and we call $StrategySelection(k, t')$ (we say that p_k asks for permission). If the call returns with permission, we set $T_s = t'$ (we say that p_k triggers the outside-branch strategy), declare that R_k is satisfied, and we return without permission.

Verification

It is clear that T is computable and that if it is infinite then its domain is equal to $\mathbb{N}$.

▶ **Lemma 35.** *T is infinite and has infinitely many branchings of distinct length.*

Proof. We show by induction on the number of levels that T has infinitely many levels.

Suppose the current T has no levels at all, i.e. the domain of T is $\{0\}$. 0 is not a branching node of T and therefore it cannot trigger the inside-branch strategy. So $0 \notin F$ forever. However, at some stage $u \geq s$, we will see p_i, with $i < u$, such that the anticipation of p_i will branch outside T and since 0 will be an open leaf of T at that stage, it will trigger the outside-branch strategy and add level 0 to T. If T has only one level, i.e., level 0, the unique branching node in T will never be put into F (because the inside-branch strategy may affect only levels ≥ 1). As above, we can show that T will grow up to level 1.

Now, let us suppose that T has exactly n levels, where $n > 1$ (that is, levels $0, 1, \dots, n-1$). The construction may put into F at most one node from each of the levels $1, 2, \dots, n-1$. Once it does, it never withdraws them from F. Suppose we are at a stage s_0 after which the construction do not add any new nodes from levels $1, 2, \dots, n-1$ into F. This means that it will never be the case that, for some $0 \leq i \leq n-2$, R_i is unsatisfied and the inside-branch strategy for p_i is ready (if we later encountered such an i, the inside-branch strategy would put some element from levels $1, 2, \dots, n-1$ into F, contradicting our choice of s_0). If T has grown by the time we reached stage s_0, we are done. So suppose otherwise. Clearly, at some later stage s, some p_j, $j < s$, with unsatisfied R_j will show us an anticipation q such that q is a functional semitree and q branches outside T. Choose the least such j at stage s. Clearly, the outside-branch strategy for p_j is ready at stage s (and R_j is unsatisfied). Moreover, no outside-branch strategy for p_k with $k < j$ is ready because of the choice of j. As for the inside-branch strategy for p_k with $k < j$, either R_k is satisfied or the inside-branch strategy for p_k not ready – this follows from the choice of s_0. Therefore, p_k will be granted permission. By the arrangement of the inside-branch strategy and by Lemma 23, T has open leaves. Therefore, the outside-branch strategy for p_k will extend T by another level.

We have already shown that the outside-branch strategy acts infinitely often. Since it always adds new branchings, there must be infinitely many branchings in T. Clearly, the strategy also guarantees that any two different branchings do not have the same length. ◀

▶ **Lemma 36.** *For every $i \in \mathbb{N}$, if $\mathcal{P}_i$ is a functional tree, then R_i is eventually satisfied.*

Proof. Fix a punctual structure $\mathcal{P}_i$ and assume that there exists a stage s_0 after which for every $j < i$, either R_i is satisfied or p_j is not ready (this holds vacuously for $i = 0$). Suppose that $\mathcal{P}_i$ is a uniquely branching functional tree (if not, $\mathcal{T} \not\cong \mathcal{P}_i$). We will show that R_i will be eventually satisfied.

Choose a stage s_0 as above and assume that $i < s_0$. Suppose that at the beginning of s_0 the requirement R_i has not yet been declared as satisfied (that is, neither of the two strategies has been triggered before s_0).

If at some stage $u \geq s_0$, we see that an anticipation q of p_i is not matchable with T_{u-1}, then R_i is satisfied by Lemma 30. In the lemma, we take $P = p_i$, $t = T_{u-1}$ and q. Let n be the maximal level of t. By the construction $t[\leq n] = t$. The infinite tree T that we are building will be such that $T \supset T_{u-1} = t$ and $T[\leq n] = t$. Hence, the premises of the lemma are satisfied. It means that $\mathcal{T} \not\cong \mathcal{P}_i$ and thus R_i is satisfied. We can therefore assume that, for $u \geq s_0$, any considered anticipation of p_i is matchable with T_{u-1}.

Now, if at some stage $u \geq s_0$ the requirement R_i is still not satisfied but p_i is ready at stage u for T_{u-1}, i will be the least j with that property. So, according to *StrategySelection*, either it will trigger an inside-branch strategy, or it will ask for permission and receive it, thus triggering an outside-branch strategy. Now, it may happen that p_i becomes ready because the outside-branch strategy for p_i is ready but the inside-branch strategy is not. In that case, according to the *StrategySelection*, p_i will receive permission and thus R_i will become satisfied.

So assume p_i never becomes ready for this reason after stage s_0. Observe that if anticipations of p_i and T_s always fail to satisfy (1), (2) or (3), then $\mathcal{T} \not\cong \mathcal{P}_i$. So assume that after stage s_0 these conditions are always satisfied. Therefore, it remains to prove that at some stage $u \geq s_0$, the inside-branch strategy is triggered.

It is important to note that once the level $i+1$ of T is done, T does not change up to level $i+1$ at any later stage (it may only add some nodes from levels $\leq i+1$ to F but this will not change the structure of T up to level $i+1$).

Consider stage $u \geq s_0$. The construction calls $\mathit{StrategySelection}(u, T_{u-1})$. Let q be the initial anticipation of p_i at stage u, namely the (u, T_{u-1})-anticipation of p_i. By the definition of anticipation, q contains more nodes than T_{u-1}. Therefore, the situation in which $R(q) = q$ and $R(q) \hookrightarrow T_{u-1}$ is not possible. So we have two remaining cases:

(a) $R(q) \hookrightarrow T_{u-1}$ and $R(q) \neq q$. In this case, there are nodes $z \in q \setminus R(q)$ that are disconnected from the root. Moreover, by the definition of anticipation, if $z \in q \setminus R(q)$ and z does not have children in q, then z gives rise to a chain of length $H(T_{u-1}) + u$ in q.

(b) $R(q) \not\hookrightarrow T_{u-1}$. In this case, q must be properly matchable with T_{u-1}. To see why, recall that q is matchable with T_{u-1}. So $R(q)$ embeds in a leaf extension T' of T_{u-1}. But any such leaf extension must be proper – otherwise $R(q) \hookrightarrow T_{u-1}$. From this it follows that q branches inside T_{u-1}. For take a proper matching of q into T_{u-1} that embeds q into $T' \sqsupset T_{u-1}$. Take the least $z \in T' \setminus T_{u-1}$. This z is connected to the root of T_{u-1} via a path that overlaps with initial $i+1$ levels of T_{u-1}. But every such path (the one in T_{u-1} or its preimage in q) contains branching nodes because q contains a copy of initial $i+1$ levels of T_{u-1}. Therefore, q branches inside T_{u-1}, and thus q is ready at stage u for T_{u-1}. Hence the inside branch strategy will handle R_i.

Assume that (b) holds. Therefore, the inside-branch strategy for p_i is ready at stage u for T_{u-1}. Hence, R_i becomes satisfied immediately after *StrategySelection* is called.

Assume that we are in case (a). Then p_i is not ready at stage u for T_{u-1}. But other p_j, with $j > i$ will become ready later on, say at stage v. If such p_j triggers an inside-branch strategy to satisfy R_j, our tree does not grow and we therefore do not miss an opportunity to satisfy R_i. However, if p_j, with $j > i$ wants to trigger an outside-branch strategy (which must happen eventually), it will produce $T' \supset T_{u-1}$ and ask for permission. So before p_j is allowed to act by extending the current tree from T_{u-1} to $T' \supset T_{u-1}$, we first check whether p_i is ready at stage v for the (bigger) tree T' that we would grow if we allowed p_j to act first. If it is not ready, it means that we can permit p_j to trigger an outside-branch strategy and set $T_v = T'$. For it means that all the long disconnected chains that we had in q remain disconnected, even after prolonging them up to length $H(T') + v$ in the (v, T')-anticipation of p_i. Notice that if we did not insist on asking for permission, we would set $T_v := T' \supset T_{u-1}$ and it could happen that prolonging a bit the disconnected chains in q would change q into a tree that embeds into T_v and we would miss the opportunity to satisfy R_i. Asking for permission prevents this.

Finally, it suffices to observe that, since p_i is a tree, at some point these disconnected chains will connect to the root, either at the beginning of *StrategySelection* or after being asked for permission by some other p_j, with $j > i$. Since these chains are kept very long (longer than any path in the current T), once they become connected, they necessarily stick out of T and thus branch inside it. At this point the inside-branch strategy is triggered and R_i is satisfied. ◀

References

1 Pavel E. Alaev and Viktor L. Selivanov. Polynomial computability of fields of algebraic numbers. *Doklady Mathematics*, 98(1):341–343, 2018. doi:10.1134/S1064562418050137.

2 Nikolay Bazhenov, Rod Downey, Iskander Kalimullin, and Alexander Melnikov. Foundations of online structure theory. *Bulletin of Symbolic Logic*, 25(2):141–181, 2019. Publisher: Cambridge University Press. doi:10.1017/bsl.2019.20.

3 Nikolay Bazhenov, Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Online presentations of finitely generated structures. *Theoretical Computer Science*, 844:195–216, 2020. doi:10.1016/j.tcs.2020.08.021.

4 Nikolay Bazhenov, Keng Meng Ng, Luca San Mauro, and Andrea Sorbi. Primitive recursive equivalence relations and their primitive recursive complexity. *Computability*, 11(3-4):187–221, 2022. doi:10.3233/COM-210375.

5 Nilkolay Bazhenov, Matthew Harrison-Trainor, Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Automatic and polynomial-time algebraic structures. *The Journal of Symbolic Logic*, 84(4):1630–1669, 2019. doi:10.1017/jsl.2019.26.

6 Douglas Cenzer, Valentina Harizanov, and Jeffrey B. Remmel. Computability-theoretic properties of injection structures. *Algebra and Logic*, 53(1):39–69, 2014. doi:10.1007/s10469-014-9270-0.

7 Douglas Cenzer and Jeffrey B. Remmel. Polynomial-time versus recursive models. *Annals of Pure and Applied Logic*, 54(1):17–58, 1991. doi:10.1016/0168-0072(91)90008-A.

8 Douglas Cenzer and Jeffrey B. Remmel. Polynomial-time abelian groups. *Annals of Pure and Applied Logic*, 56(1):313–363, 1992. doi:10.1016/0168-0072(92)90076-C.

9 Douglas Cenzer and Jeffrey B. Remmel. Feasible graphs with standard universe. *Annals of Pure and Applied Logic*, 94(1):21–35, 1998. doi:10.1016/S0168-0072(97)00064-X.

10 Rod Downey, Noam Greenberg, Alexander Melnikov, Keng Meng Ng, and Daniel Turetsky. Punctual categoricity and universality. *The Journal of Symbolic Logic*, 85(4):1427–1466, 2020. doi:10.1017/jsl.2020.51.

11 Noam Greenberg, Matthew Harrison-Trainor, Alexander Melnikov, and Dan Turetsky. Non-density in punctual computability. *Annals of Pure and Applied Logic*, 172(9):102985, 2021. doi:10.1016/j.apal.2021.102985.

12 Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Algebraic structures computable without delay. *Theoretical Computer Science*, 674:73–98, 2017. doi:10.1016/j.tcs.2017.01.029.

13 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In Daniel Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer Berlin Heidelberg, 1995. doi:10.1007/3-540-60178-3_93.

14 Anatolii I. Mal'tsev. Constructive algebras i. *Russian Mathematical Surveys*, 16(3):77, 1961. doi:10.1070/RM1961v016n03ABEH001120.

15 Antonio Montalbán. *Computable structure theory: Within the arithmetic*. Cambridge University Press, 2021.

16 Michael O. Rabin. Computable algebra, general theory and theory of computable fields. *Transactions of the American Mathematical Society*, 95(2):341–360, 1960. doi:10.2307/1993295.

Twin-Width of Graphs on Surfaces

Daniel Kráľ
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Kristýna Pekárková
Faculty of Informatics, Masaryk University, Brno, Czech Republic

Kenny Štorgel
Faculty of Information Studies in Novo mesto, Slovenia

Abstract

Twin-width is a width parameter introduced by Bonnet, Kim, Thomassé and Watrigant [FOCS'20, JACM'22], which has many structural and algorithmic applications. Hliněný and Jedelský [ICALP'23] showed that every planar graph has twin-width at most 8. We prove that the twin-width of every graph embeddable in a surface of Euler genus g is at most $18\sqrt{47g} + O(1)$, which is asymptotically best possible as it asymptotically differs from the lower bound by a constant multiplicative factor. Our proof also yields a quadratic time algorithm to find a corresponding contraction sequence. To prove the upper bound on twin-width of graphs embeddable in surfaces, we provide a stronger version of the Product Structure Theorem for graphs of Euler genus g that asserts that every such graph is a subgraph of the strong product of a path and a graph with a tree-decomposition with all bags of size at most eight with a single exceptional bag of size $\max\{6, 32g - 37\}$.

2012 ACM Subject Classification Mathematics of computing → Graphs and surfaces

Keywords and phrases twin-width, graphs on surfaces, fixed parameter tractability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.66

Related Version *Full Version*: https://arxiv.org/abs/2307.05811

Funding The first two authors have been supported by the MUNI Award in Science and Humanities (MUNI/I/1677/2018) of the Grant Agency of Masaryk University. The third author is supported by the Slovenian Research Agency (research program P1-0383, research projects J1-3002, J1-4008).

Acknowledgements The substantial part of the work presented in this article was done during the Brno–Koper research workshop on graph theory topics in computer science held in Kranjska Gora in April 2023, which all three authors have participated in. The authors would like to thank the anonymous reviewers for their numerous constructive comments, which helped to clarify and improve various aspects of the presentation.

1 Introduction

Twin-width is a graph parameter, which has recently been introduced by Bonnet, Kim, Thomassé and Watrigant [16, 17]. It has quickly become one of the most intensively studied graph width parameters due to its many connections to algorithmic and structural questions in both computer science and mathematics. In particular, classes of graphs with bounded twin-width (we refer to Section 2 for the definition of the parameter) include at the same time well-structured classes of sparse graphs and well-structured classes of dense graphs. Particular examples are classes of graphs with bounded tree-width, with bounded rank-width (or equivalently with bounded clique-width), and classes excluding a fixed graph as a minor. As the first order model checking is fixed parameter tractable for classes of graphs with bounded twin-width [16, 17], the notion led to a unified view of various earlier results on fixed parameter tractability of first order model checking of graph properties [21, 22, 30–33, 49], and more generally first order model checking properties of other combinatorial structures such as matrices, permutations and posets [4, 12, 19, 20].

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 66; pp. 66:1–66:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The foundation of the theory concerning twin-width has been laid by Bonnet, Kim, Thomassé and their collaborators in a series of papers [8–14,16,17], also see [51]. The amount of literature on twin-width is rapidly growing and includes exploring algorithmic aspects of twin-width [6,10,12,15,48], its combinatorial properties [2,4,7,10,14,24,47], and connections to logic and model theory [12,16,17,19,20,34]. While many important graph classes have bounded twin-width, good bounds are known only in a small number of specific cases. One of the examples is the class of graphs of bounded tree-width where an asymptotically optimal bound, exponential in tree-width, was proven by Jacob and Pilipczuk [40]. Another example is the class of planar graphs. The first explicit bound of 583 by Bonnet, Kwon and Wood [18] was gradually improved [5,35,40] culminating with a bound of 8 obtained by Hliněný and Jedelský [38]; also see [36,37] for a simpler proof and [41] for a promising approach of obtaining the upper bound of 7, which would be tight since Lamaison and the first author [42] constructed a planar graph with twin-width 7. In this paper, we extend this list by providing an asymptotically optimal upper bound on the twin-width of graphs embeddable in surfaces of higher genera. We prove the following two results (the latter is used to prove the former):

- We show that the twin-width of a graph embeddable in a surface of Euler genus g is at most $18\sqrt{47g}+O(1)$, which is asymptotically best possible; our proof also yields a quadratic time algorithm to find a witnessing sequence of vertex contractions.
- We provide a strengthening of the Product Structure Theorem for graphs embeddable in a surface of Euler genus g by showing that such graphs are subgraphs of a strong product of a path and a graph that almost has a bounded tree-width.

We next present the two results in more detail while also presenting the related existing results. While we prove both results in purely structural way, their proofs are algorithmic and yield a quadratic time algorithm (when the genus $g>0$ is fixed) that given a graph G embeddable in a surface of genus g, constructs a sequence of contractions witnessing that the twin-width of G is at most $18\sqrt{47g}+O(1)$. Further details are discussed in Section 5.

1.1 Twin-width of graphs embeddable in surfaces

Graphs that can be embedded in surfaces of higher genera, such as the projective plane, the torus and the Klein bottle, form important minor-closed classes of graphs with many applications and connections [45]. While the general theory concerning minor-closed classes of graphs yields that graphs embeddable in a fixed surface have bounded twin-width, the bounds are quite enormous: the results from [14, Section 4] on d-contractible graphs (graphs embeddable in a surface of Euler genus g are $O(g)$-contractible [39]) yields a bound double exponential in g, and the Product Structure Theorem for graphs embeddable in surfaces [26,28] together with results on the twin-width of graphs with bounded tree-width [40], of the strong product of graphs [46] and their subgraph closure [9] yields an exponential bound.

Bonnet, Kwon and Wood [18] showed that every graph embeddable in a surface of Euler genus g has twin-width at most $205g+583$. Our main result asserts that twin-width of every graph embeddable in a surface of Euler genus g is at most $18\sqrt{47g}+O(1)\approx 123.4\sqrt{g}+O(1)$. This bound is asymptotically optimal as any graph with $\sqrt{6g}-O(1)$ vertices can be embedded in a surface of Euler genus g and the n-vertex Erdős-Rényi random graph $G_{n,1/2}$ has twin-width at least $n/2-O(\sqrt{n\log n})$ [1], i.e., there exists a graph with twin-width $\sqrt{3g/2}-o(\sqrt{g})$ embeddable in a surface of Euler genus g. In particular, our upper bound asymptotically differs from the lower bound by a multiplicative factor $6\sqrt{282}\approx 100.76$. While several parts of our argument can be refined to decrease the multiplicative constant (to around 20), we have decided not to do so due to the technical nature of such refinements and the absence of additional structural insights gained by doing so.

1.2 Product Structure Theorem

On the way to our main result, we prove a modification of the Product Structure Theorem that applies to graphs embeddable in surfaces. The Product Structure Theorem is a recent significant structural result obtained by Dujmović, Joret, Micek, Morin, Ueckerdt and Wood [26, 28], which brought new substantial insights into the structure of planar graphs and led to breakthroughs on several long standing open problems concerning planar graphs, see, e.g. [25]. We also refer to the survey by Dvořák et al. [29] on the topic. The statement of the Product Structure Theorem originally proven by Dujmović et al. [26, 28] reads as follows (we remark that the statement in [26, 28] does not include the condition on planarity of the graph of bounded tree-width, however, an easy inspection of the proof yields this).

▶ **Theorem 1.** *Every planar graph is a subgraph of the strong product of a path and a planar graph with tree-width at most* 8*.*

Ueckerdt et al. [52] improved this as follows (we state a corollary of their main result to avoid defining the notion of simple tree-width, which is not needed in our further presentation).

▶ **Theorem 2.** *Every planar graph is a subgraph of the strong product of a path and a planar graph with tree-width at most* 6*.*

Dujmović et al. [26, 28] also proved two extensions of the Product Structure Theorem to graphs embeddable in surfaces.

▶ **Theorem 3.** *Every graph embeddable in a surface of Euler genus $g > 0$ is a subgraph of the strong product of a path, the complete graph K_{2g} and a planar graph with tree-width at most* 9*.*

▶ **Theorem 4.** *Every graph embeddable in a surface of Euler genus $g > 0$ is a subgraph of the strong product of a path, the complete graph $K_{\max\{2g,3\}}$ and a planar graph with tree-width at most* 4*.*

A stronger version was proven by Distel at el. [23]; the discussion of the even stronger statement implied by the proof of the next theorem given in [23] can be found after Theorem 6.

▶ **Theorem 5.** *Every graph embeddable in a surface of Euler genus $g > 0$ is a subgraph of the strong product of a path, the complete graph $K_{\max\{2g,3\}}$ and a planar graph with tree-width at most* 3*.*

We remark that it is not possible to replace K_{2g} in the statement of Theorems 3, 4 and 5 with a complete graph with $o(g)$ vertices as long as the bound on the tree-width stays constant since the layered tree-width of graphs embeddable in a surface of Euler genus g is linear in g [27] (the definition of layered tree-width is given in Section 2). To prove our upper bound on the twin-width of graphs embeddable in surfaces, we strengthen the statement of the Product Structure Theorem for graphs embeddable in surfaces as follows. Theorems 3, 4 and 5 imply that every graph embeddable in a surface of Euler genus $g > 0$ is a subgraph of the strong product of a path and a graph with tree-width at most $20g - 1$, $\max\{10g - 1, 14\}$ and $\max\{8g - 1, 11\}$, respectively. The next theorem, which we prove in Section 3, asserts that it is possible to assume that the tree-width of the graph in the product is almost at most 7 in the sense that all bags except possibly for a single bag have size at most 8.

▶ **Theorem 6.** *Every graph embeddable in a surface of Euler genus $g > 0$ is a subgraph of the strong product of a path and a graph H that has a rooted tree-decomposition such that*

- *the root bag has size at most $\max\{6, 32g - 37\}$, and*
- *every bag except the root bag has size at most* 8*.*

Similarly as it is not possible to replace K_{2g} with a complete graph with $o(g)$ vertices in Theorem 3, it is necessary to permit at least one of the bags to have a size linear in g in Theorem 6. Hence, the statement of Theorem 6 is the best possible asymptotically.

We remark that the proof of Theorem 5 given in [23] implies that every graph embeddable in a surface of Euler genus $g > 0$ is a subgraph of the strong product of a path and a graph that can be obtained from a planar graph with tree-width at most 3 by replacing one vertex of this planar graph with K_{2g} and the remaining vertices with K_3 (and replacing each edge of the planar graph with a complete bipartite graph between the corresponding sets of vertices). However, the vertex of the planar graph that is replaced with K_{2g} can be contained in many bags of the tree-decomposition and so the proof given in [23] does not yield a statement similar to that of Theorem 6 since the number of bags in the tree-decomposition with size linear in g can be arbitrary (although each such bag contains the same $2g$ vertices of K_{2g} in addition to 9 other vertices). The main new component in the proof of Theorem 6 (compared to the proofs given in [23, 26, 28]) is Lemma 12 given in Section 3, which is crucial so that we are able to restrict the sizes of all but one bag in a tree-decomposition to a constant size.

We also note the following corollary of Theorem 6 for projective planar graphs.

▶ **Corollary 7.** *Every graph embeddable in the projective plane is a subgraph of the strong product of a path and a graph with tree-width at most* 7*.*

2 Preliminaries

In this section, we introduce notation used throughout the paper. We use $[n]$ to denote the set of the first n positive integers, i.e., $\{1, \ldots, n\}$. All graphs considered in this paper are simple and have no parallel edges unless stated otherwise; if G is a graph, we use $V(G)$ to denote the vertex set of G. A *triangulation* of the plane or a surface of Euler genus $g > 0$ is a graph embedded in such a surface such that every face is a 2-cell, i.e., homeomorphic to a disk, and bounded by a triangle. A *near-triangulation* is a 2-connected graph G embedded in the plane such that each inner face of G is bounded by a triangle.

We next give a formal definition of twin-width. A *trigraph* is a graph with some of its edges being red; the *red degree* of a vertex v is the number of red edges incident with v. If G is a trigraph and v and v' form a pair of its (not necessarily adjacent) vertices, then the trigraph obtained from G by *contracting* the vertices v and v' is the trigraph obtained from G by removing the vertices v and v' and introducing a new vertex w such that w is adjacent to every vertex u that is adjacent to at least one of the vertices v and v' in G and the edge wu is red if u is not adjacent to both v and v' or at least one of the edges vu and $v'u$ is red. The *twin-width* of a graph G is the smallest integer k such that there exists a sequence of contractions that reduces the graph G, i.e., the trigraph with the same vertices and edges as G and no red edges, to a single vertex, and none of the intermediate graphs contains a vertex of red degree more than k.

A *rooted tree-decomposition* $\mathcal{T}$ of a graph G is a rooted tree such that each vertex of $\mathcal{T}$ is a subset of $V(G)$, which we refer to as a *bag*, and that satisfies the following:

- for every vertex v of G, there exists a bag containing v,
- for every vertex v of G, the bags containing v form a connected subgraph (subtree) of $\mathcal{T}$, and
- for every edge e of G, there exists a bag containing both end vertices of e.

If the choice of the root is not important, we just speak about a *tree-decomposition* of a graph G. The *width* of a tree-decomposition $\mathcal{T}$ is the maximum size of a bag of $\mathcal{T}$ decreased by one, and the *tree-width* of a graph G is the minimum width of a tree-decomposition of G.

A *k-tree* is defined recursively as follows: the complete graph K_k is a k-tree and if G is a k-tree, then any graph obtained from G by introducing a new vertex and making it adjacent to any k vertices of G that form a complete subgraph in G is also a k-tree. Note that a graph G is a 1-tree if and only if G is a tree. More generally, a graph G has tree-width at most k if and only if G is a subgraph of a k-tree, and if G has at least k vertices, then G is a spanning subgraph of a k-tree. Note that k-trees have a tree-like structure given by their recursive definition, which also gives a rooted tree-decomposition of G with width k: the rooted tree-decomposition of K_k consists of a single bag containing all k vertices, and the rooted tree-decomposition of the graph obtained from a k-tree G by introducing a vertex w can be obtained from the rooted tree-decomposition $\mathcal{T}_G$ of G by introducing a new bag containing w and its k neighbors and making this bag adjacent to the bag of $\mathcal{T}_G$ that contains all k neighbors of w (such a bag exists since the subtrees of a tree have the Helly property).

A *BFS spanning tree* T of a (connected) graph G is a rooted spanning tree such that the path from the root to any vertex v in T is the shortest path from the root to v in G; in particular, a BFS spanning tree can be obtained by the breadth-first search (BFS). A *layering* is a partition of a vertex set of a graph G into sets $V_1, \dots, V_k$, which are called *layers*, such that every edge of G connects two vertices of the same or adjacent layers. i.e., layers whose indices differ by one. If T is a BFS spanning tree of G, then the partition of $V(G)$ into sets based on the distance from the root of T is a layering. A *BFS spanning forest* F of a (not necessarily connected) graph G is a rooted spanning forest, i.e., a forest consisting of rooted trees, such that there exists a layering $V_1, \dots, V_k$ compatible with F, i.e., for every tree of F, there exists d such that the vertices at distance ℓ from the root are contained in $V_{d+\ell}$. Note that if G is a graph and T a BFS spanning tree of G, then removing the same vertices in G and T results in a graph G' and a BFS spanning forest of G'. Finally, the *layered tree-width* of a graph G is the minimum k for which there exists a tree-decomposition $\mathcal{T}$ of G and a layering such that every bag of $\mathcal{T}$ contains at most k vertices from the same layer.

Consider a graph G and a BFS spanning tree T of G. A *vertical* path is a path contained in T with no two vertices from the same layer, i.e., a subpath of a path from a leaf to the root of T. The vertex of a vertical path closest to the root is its *top* vertex and the vertex farthest is its *bottom* vertex. Vertical paths with respect to a BFS spanning forest are defined analogously. If $\mathcal{P}$ is a partition of the vertex set of G to vertical paths, the graph $G/\mathcal{P}$ is the graph obtained by contracting each of the paths contained in $\mathcal{P}$ to a single vertex; note that the vertices of $G/\mathcal{P}$ can be viewed as the vertical paths contained in $\mathcal{P}$ and two vertical paths P and P' are adjacent in $G/\mathcal{P}$ if the graph G has an edge between $V(P)$ and $V(P')$.

3 Product Structure Theorem for graphs on surfaces

In this section, we provide the version of the Product Structure Theorem for graphs on surfaces, which we need to prove our upper bound on the twin-width. We start with recalling the following lemma proven by Dujmović et al. [26, 28]; note that the fundamental cycles determined by the edges $a_1b_1, \dots, a_gb_g$ generate the fundamental group of the surface Σ.

▶ **Lemma 8.** *Let G be a triangulation of a surface Σ of Euler genus $g > 0$ and let T be a BFS spanning tree of G. There exist edges $a_1b_1, \dots, a_gb_g$ not contained in the tree T with the following property. Let F_0 be the subset of edges of G comprised of the g edges $a_1b_1, \dots, a_gb_g$ and the edges of the $2g$ paths in T from the root of T to the vertices $a_1, \dots, a_g$ and $b_1, \dots, b_g$. The surface Σ after the removal of the edges contained in F_0 is homeomorphic to a disk and its boundary is formed by a closed walk comprised of the edges contained in F_0.*

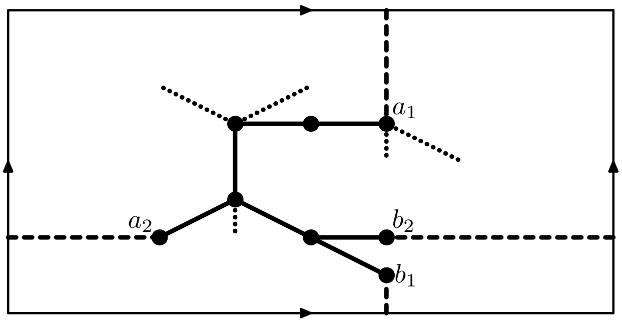


Figure 1 A rooted tree T_0 and edges a_1b_1 and a_2b_2, which are drawn dashed, bounding a part of the torus that is homeomorphic to a disk as in Lemma 9. Possible additional edges of the BFS spanning tree T are drawn dotted.

Using Lemma 8, one can prove the following; we refer to Figure 1 for the illustration of the notation in the case of the torus. The proof of the lemma is omitted due to space constraints.

▶ **Lemma 9.** *Let G be a triangulation of a surface of Euler genus $g > 0$ and let T be a BFS spanning tree of G. There exist a closed walk W in G, a subtree T_0 of T that contains the root of T, and k vertex-disjoint vertical paths $P_1, \ldots, P_k$, $k \leq 2g$, such that*

- *the closed walk W bounds a part of the surface homeomorphic to a disk,*
- *the sets $V(P_1), \ldots, V(P_k)$ form a partition of $V(T_0)$, i.e., $V(T_0) = V(P_1) \cup \cdots \cup V(P_k)$, and*
- *the sequence of vertices given by traversing the closed walk W can be split into at most $6g - 1$ segments such that all vertices of each segment belong to the same vertical path.*

Lemma 9 is one of two key ingredients for the proof of Theorem 13. The second, which is Lemma 12, relates to partitioning disk regions bounded by vertical paths. Similarly to [26, 28, 52], we make use of Sperner's Lemma, see e.g. [3, 50].

▶ **Lemma 10.** *Let G be a near-triangulation. Suppose that the vertices of G are colored with three colors in such a way that the vertices of each of the three colors on the outer face are consecutive, i.e., they form a non-empty path. There exists an inner face that contains one vertex of each of the three colors.*

The proof of the next lemma follows the lines of the proof of [52, Lemma 8] and is omitted due to space constraints. We say that a cycle is *covered* by paths $P_1, \ldots, P_k$ if each path is a subpath of the cycle and each vertex of the cycle belongs to one of the paths $P_1, \ldots, P_k$.

▶ **Lemma 11.** *Let G be a near-triangulation and let T be a BFS rooted spanning forest such that all roots of T are on the outer face. If the boundary cycle of the outer face can be covered by at most 6 vertex-disjoint vertical paths, say $P_1, \ldots, P_k$, $k \leq 6$, then there exists a collection $\mathcal{P}$ of vertex-disjoint vertical paths such that*

- *the collection $\mathcal{P}$ contains the paths $P_1, \ldots, P_k$,*
- *every vertex of G is contained in one of the paths in $\mathcal{P}$, and*
- *$G/\mathcal{P}$ has a rooted tree-decomposition of width at most seven such that the root bag contains the k vertices corresponding to the paths $P_1, \ldots, P_k$.*

To state the next lemma, which is the second ingredient to prove the main result of this section, we need the following definition: if G' is a subgraph of a graph G embedded in a surface, a face of G' is a *region* if its interior contains a vertex or an edge of G; an *inner region* is an inner face that is a region. The proof of the lemma is omitted due to space constraints.

▶ **Lemma 12.** *Let G be a near-triangulation and let T be a BFS rooted spanning forest such that all roots of T are on the outer face. If the boundary cycle of the outer face can be covered by $k \geq 6$ vertex-disjoint vertical paths $P_1, \ldots, P_k$, then there exist a 2-connected subgraph G' of G and a collection $\mathcal{P}$ of vertex-disjoint vertical paths such that*

- *$\mathcal{P}$ contains the vertical paths $P_1, \ldots, P_k$,*
- *$\mathcal{P}$ contains at most $\max\{6, 6k - 32\}$ vertical paths,*
- *the vertex set of G' is the union of the vertex sets of the vertical paths contained in $\mathcal{P}$,*
- *the graph G' contains the boundary of the outer face,*
- *the graph G' has at most $\max\{1, 3k - 18\}$ inner regions, and*
- *the boundary cycle of each inner region of G' can be covered by at most six paths such that each is a subpath of a path from $\mathcal{P}$.*

We can now prove the main result of this section. Since a triangulation G in Theorem 13 is a subgraph of the strong product of a path and the graph $G/\mathcal{P}$, Theorem 13 readily implies Theorem 6. We remind that a tree-decomposition of $G/\mathcal{P}$ is a tree whose vertices are bags containing paths from the set $\mathcal{P}$ (the vertices of $G/\mathcal{P}$ can be viewed as these paths).

▶ **Theorem 13.** *Let G be a triangulation of a surface of Euler genus $g > 0$ and let T be a BFS spanning tree of G. There exists a collection $\mathcal{P}$ of vertical paths that partition the vertex set of G and the graph $G/\mathcal{P}$ has a rooted tree-decomposition such that*

- *the root bag has size at most $\max\{6, 32g - 37\}$,*
- *the root bag has at most $6 \cdot \max\{1, 18g - 21\}$ children, and*
- *every bag except the root bag has size at most 8.*

Moreover, every subtree $\mathcal{T}'$ of the tree-decomposition formed by a child of the root and all its descendants satisfies the following:

- *the bags of $\mathcal{T}'$ contain at most six paths that are contained in the root bag, and*
- *if $P_1, \ldots, P_k$ are all paths from $\mathcal{P}$ that are contained in the bags of $\mathcal{T}'$ but not in the root bag, the subgraph induced by $V(P_1) \cup \cdots \cup V(P_k)$ has a component joined by an edge to each of the paths that are contained both in the root bag and in $\mathcal{T}'$.*

Proof. Fix a triangulation G of a surface of Euler genus $g > 0$ and a BFS spanning tree T of G. We apply Lemma 9 to obtain a closed walk W, a subtree T_0 of T and k vertex-disjoint vertical paths $P_1, \ldots, P_k$, $k \leq 2g$, with the properties given in Lemma 9. Let $\ell \leq 6g - 1$ be the number of segments that cover the closed walk W as in the statement of the lemma.

We first deal with the general case $\ell \geq 7$ (note that if $\ell \geq 7$, then $g \geq 2$). We apply Lemma 12 to the near-triangulation obtained by cutting along the closed walk W, the BFS spanning forest obtained from T_0 by duplicating the vertices contained in W as needed, and the ℓ vertical paths that corresponds to the segments that cover the closed walk W. We obtain a collection $\mathcal{P}_0$ of vertex-disjoint vertical paths that contains at most $5\ell - 32$ additional vertical paths and a 2-connected subgraph G' such that the boundary of each inner region of G' can be covered by at most six paths contained in $\mathcal{P}_0$. In addition, the number of inner regions of G', further denoted by f, is at most $3\ell - 18$. Since $\ell \leq 6g - 1$, we obtain that $\mathcal{P}_0$ contains at most $30g - 37$ additional vertical paths and that $f \leq 18g - 21$. We now identify the duplicated vertices of T_0, i.e., G' has been modified to a subgraph of G, and we replace in the collection $\mathcal{P}_0$ the ℓ paths that cover the closed walk W with the paths $P_1, \dots, P_k$. Hence, the size of the collection $\mathcal{P}_0$ is at most $32g - 37$ (note that $k \le 2g$) and the boundary of each region of G' is still covered by at most six paths such that each is a subpath of the vertical paths contained in $\mathcal{P}_0$ (two different paths can be subpaths of the same vertical path).

If $\ell \le 6$ (and so $k \le 6$), we set $\mathcal{P}_0$ to be the collection $\{P_1, \dots, P_k\}$ and G' the graph consisting of the vertices and the edges of the closed walk W; note that the only face of G' bounds a near-triangulation in G and $f = 1$.

We now proceed jointly for all values of ℓ. Suppose there is a region of G' such that the subgraph of G induced by the vertices of G inside this region does not have a component joined by an edge to each of the (at most six) paths that cover the boundary of the region and that are subpaths of paths from $\mathcal{P}_0$. Then, because G is a triangulation, there are two vertices on the boundary of this region joined by an edge not contained in G' and we add this edge to G'. We proceed as long as such a region exists and eventually obtain a graph G'' with $f' \le 6f$ regions such that the boundary of each region can be covered by at most six paths, each subpath of a path contained in $\mathcal{P}_0$, and each region contains a component that is joined by an edge to each of the (at most six) paths that cover its boundary. We now apply Lemma 11 to each of the f' near-triangulations bounded by the regions of G'' and obtain rooted tree-decompositions $\mathcal{T}_1, \dots, \mathcal{T}_{f'}$ with width at most seven of each them. Let $\mathcal{P}$ be the collection of vertical paths obtained from $\mathcal{P}_0$ by including all additional vertical paths obtained by these f' applications of Lemma 11.

We now construct a rooted tree-decomposition of $G/\mathcal{P}$. The root bag contains the vertices corresponding to the paths in $\mathcal{P}_0$ and the subtrees rooted at its children are $\mathcal{T}_1, \dots, \mathcal{T}_{f'}$. So, the root bag has size $|\mathcal{P}_0| \le \max\{6, 32g - 37\}$, it has $f' \le 6f \le 6\max\{1, 18g - 21\}$ children, and all bags except the root bag has size at most 8. Consider now a subtree $\mathcal{T}_i$, $i \in [f']$. The only paths from $\mathcal{P}_0$ contained in the bags of $\mathcal{T}_i$ are the at most six paths whose subpaths cover the boundary of the corresponding region of G'' and the vertices contained in the paths of the bags of $\mathcal{T}_i$ but not in the paths of $\mathcal{P}_0$ are exactly the vertices of G contained inside the region. Hence, the subgraph of G induced by such vertices has a component joined by an edge to each of the paths from $\mathcal{P}_0$ contained in the root bag of $\mathcal{T}_i$. We conclude that the obtained rooted tree-decomposition of $G/\mathcal{P}$ has the properties given in the statement. ◀

4 Bound on twin-width

We now present the asymptotically optimal upper bound on the twin-width of graphs embeddable in surfaces.

▶ **Theorem 14.** *The twin-width of every graph G of Euler genus $g \ge 1$ is at most*

$$6 \cdot \max\left\{3\sqrt{47g} + 1, 2^{24}\right\} = 18\sqrt{47g} + O(1).$$

Proof. Fix a graph G of Euler genus $g > 0$ and let G_0 be any triangulation of the surface with Euler genus g that G is a spanning subgraph of G_0, i.e., $V(G_0) = V(G)$ (to avoid unnecessary technical issues related to adding new vertices, G_0 may contain parallel edges).

We apply Theorem 13 to G_0 and an arbitrary BFS spanning tree T_0; let $\mathcal{P}$ be a collection of vertical paths and $\mathcal{T}$ a rooted tree-decomposition as in Theorem 13. Let $P_1, \dots, P_k$ be the vertical paths contained in the root bag and let $\mathcal{T}_1, \dots, \mathcal{T}_\ell$ be the subtrees rooted at the children of the root bag (note that $k \le 32g$ and $\ell \le 108g$). Further, let V_i, $i \in [\ell]$, be the vertices contained in the vertical paths in the bags of $\mathcal{T}_i$ that are not contained in the root bag. Note that for every $i = 1, \dots, \ell$, the subgraph induced by a set V_i has a component that is joined by an edge to each path P_j, $j \in [k]$, that is contained in a bag of the subtree $\mathcal{T}_i$.

Let H_0 be the graph obtained from G_0 by contracting each of the $k+\ell$ sets $V(P_1),\dots,V(P_k)$ and $V_1,\dots,V_\ell$ to a single vertex. Let $a_1,\dots,a_k$ and $b_1,\dots,b_\ell$ be the resulting vertices. Observe that H_0 can be obtained from G_0 by contracting edges and deleting vertices. Indeed, the vertices $a_1,\dots,a_k$ are obtained by contracting paths $P_1,\dots,P_k$ and each vertex b_i, $i=1,\dots,\ell$, can be obtained as follows: first contract the component of the subgraph induced by the set V_i that is joined by an edge to each of the paths $P_1,\dots,P_k$ contained in the subtree $\mathcal{T}_i$ to a single vertex, and then delete all the vertices of V_i not contained in this component. Since H_0 can be obtained from G_0 by contracting edges and deleting vertices, the graph H_0 can be embedded in the same surface as G_0. Hence, the number of the edges of H_0 is at most $3(k+\ell)-6+3g \le 3(k+\ell+g)$; the latter bound applies even if $k+\ell=2$. Since each subtree $\mathcal{T}_i$ contains at most six of the paths $P_1,\dots,P_k$, each of the vertices $b_1,\dots,b_\ell$ has degree at most six and all its (at most six) neighbors are among the vertices $a_1,\dots,a_k$.

Let $s=3\sqrt{47g}$; note that $s\ge 6$. We next split the vertices $a_1,\dots,a_k$ into sets $A_1,\dots,A_{k'}$ and the vertices $b_1,\dots,b_\ell$ into sets $B_1,\dots,B_{\ell'}$ as follows; a similar argument has also been used in [2]. Keep adding the vertices $b_1,\dots,b_\ell$ to the set B_1 until the sum of their degrees just exceeds s, then keep adding the remaining vertices to the set B_2 until the sum of their degrees just exceeds s, etc. Observe that the sum of the degrees of the vertices in each of the sets $B_1,\dots,B_{\ell'}$ is at most $s+6\le 2s$ and the sum of the degrees of the vertices in each of the sets $B_1,\dots,B_{\ell'-1}$ is at least s. Each of the vertices $a_1,\dots,a_k$ with degree larger than s forms a set of size one, and the remaining vertices are split in the same way as the vertices $b_1,\dots,b_k$. Each of the sets $A_1,\dots,A_{k'}$ has either size one or the sum of the degrees of its vertices is at most $2s$, and the sum of the degrees of the vertices in each of the sets $A_1,\dots,A_{k'-1}$ is at least s. Let H_0' be the graph obtained from H_0 by contracting the vertices in each of the sets $A_1,\dots,A_{k'}$ and each of the sets $B_1,\dots,B_{\ell'}$ to a single vertex; note that the graph H_0' does not need to be embeddable in the same surface as H_0. Since the sum of the degrees of the vertices $a_1,\dots,a_k$ and $b_1,\dots,b_\ell$ is at most $6(k+\ell+g)\le 846g$ (as H_0 has at most $3(k+\ell+g)$ edges), we obtain that $k'+\ell'\le \frac{846g}{s}+2=2s+2$, i.e., H_0' has at most $2s+2$ vertices.

We now describe the order in which we contract the vertices of G, and we analyze the described order later. In what follows, when we say a *layer*, we always refer to the layers given by the BFS spanning tree T_0 from the application of Theorem 13. In particular, each vertex of G is adjacent only to the vertices in its own layer and the two neighboring layers. To make the presentation clearer, we split contracting vertices into three phases.

Phase I. This phase consists of ℓ subphases. In the i-th subphase, $i\in[\ell]$, we contract all the vertices of the set V_i that are contained in the same layer to a single vertex in the way that we now describe. Then, we possibly contract them to some of the vertices created in the preceding subphases, i.e., those obtained by contracting vertices in $V_1\cup\dots\cup V_{i-1}$. In this phase, *we never contract two vertices contained in different layers* and no contraction involves any vertex from $V(P_1)\cup\dots\cup V(P_k)$.

Subphase. Fix $i\in[\ell]$. Let G_i be the subgraph of $G_0/\mathcal{P}$ induced by the vertices contained in the bags of the subtree $\mathcal{T}_i$ and let n' be the number of the paths $P_1,\dots,P_k$ that are contained in the bags of the subtree $\mathcal{T}_i$; note that $n'\le 6$. If the graph G_i has less than 8 vertices, we proceed directly to the conclusion of the subphase, which is described below. If the graph G_i has at least 8 vertices, G_i is a subgraph of a spanning subgraph of a 7-tree G_i' such that the n' vertices corresponding to the paths from the set $\{P_1,\dots,P_k\}$ are contained in the initial complete graph of G_i'.

Fix any order $Q_1,\dots,Q_n$ of the vertical paths corresponding to the vertices of G_i' such that the neighbors of Q_j, $j\in[n]$, among $Q_1,\dots,Q_{j-1}$ form a complete graph of order at most 7 in G_i' and the n' paths from the set $\{P_1,\dots,P_k\}$ are the paths $Q_1,\dots,Q_{n'}$. Let C_j be the complete subgraph of G_i formed by Q_j and its (at most 7) neighbors among $Q_1, \ldots, Q_{j-1}$. Note that the neighbors in G of each vertex of a path Q_j, $j \in [n]$, are contained in at most seven of the paths $Q_1, \ldots, Q_{j-1}$, which are exactly the paths forming the complete graph C_j. We define the *j-shadow* of a vertex $v \in V_i$ to be the set of its neighbors contained in the paths $Q_1, \ldots, Q_{j-1}$. Since every vertex of Q_j has at most 21 neighbors on the paths $Q_1, \ldots, Q_{j-1}$ (as its neighbors must be in the same or adjacent layers), the j-shadow of a vertex contained in the path Q_j has at most 21 vertices.

We now use the tree-like structure of the 7-tree G'_i to define the order of contractions of the vertices contained in V_i; this part of our argument is analogous to that used in [40] in relation to twin-width of graphs with bounded tree-width. We proceed iteratively for $j = n-1, \ldots, n'$. Before we describe the order of contractions, we present the properties satifised at the end of the iterations. At the end of the iteration for $j = n-1, \ldots, n'+1$, all vertices of V_i that

- are contained in paths of the same component of $G'_i \setminus \{Q_1, \ldots, Q_{j-1}\}$,
- have the same j-shadow, and
- are in the same layer

will have been contracted to a single vertex. At the end of the iteration for $j = n'$, all vertices of V_i with the same $(n'+1)$-shadow that are contained in the same layer will have been contracted to a single vertex. In particular, at the end of the iteration for $j = n'$, all vertices of V_i contained in the same layer will have been contracted to at most $2^{3n'}$ vertices (a vertex can have at most three neighbors on each path Q_j, $j \in [n']$, which are the vertex on the same layer and the two vertices on the adjacent layers, and the $(n'+1)$-shadow is a subset of these $3n'$ vertices).

For $j \in \{n-1, \ldots, n'\}$, we now describe the order of contractions of the vertices in the iteration for j. Let m be the number of components of $G'_i \setminus \{Q_1, \ldots, Q_j\}$ that are included in the component of $G'_i \setminus \{Q_1, \ldots, Q_{j-1}\}$ that contains Q_j, and let $W_1, \ldots, W_m$ be the sets of vertices obtained by contracting (in the previous iterations) vertices on the paths of these m components; note that the vertices in $W_1, \ldots, W_m$ are obtained by contracting some vertices contained in $V(Q_{j+1}) \cup \cdots \cup V(Q_n) \subseteq V_i$. Observe that each of the sets $W_1, \ldots, W_m$ has at most 2^{21} vertices in each layer (as the $(j+1)$-shadow of vertices on the same layer are subsets of the same set of $3 \cdot 7 = 21$ vertices). We first contract each vertex of W_2 to the vertex W_1 with the same $(j+1)$-shadow on the same layer if such vertex exists. Next, we contract each vertices of W_3 to the vertex of $W_1 \cup W_2$ with the same $(j+1)$-shadow on the same layer if such vertex exists, etc. At the end of this process, all vertices of $W_1 \cup \cdots \cup W_m$ with the same $(j+1)$-shadow that are on the same layer have been contracted to a single vertex (note that there are at most 2^{24} such vertices in each layer as the $(j+1)$-shadows are subsets of the same set of $3 \cdot 8 = 24$ vertices). If $j > n'$, we contract all resulting vertices with the same j-shadow that are on the same layer to a single vertex, and subsequently, we contract the vertex contained on the path Q_j to the vertex with the same j-shadow on the same layer (if such vertex exists). The description of the iteration for j is now finished.

Conclusion of subphase. The i-th subphase concludes by contracting all the vertices of V_i in the same layer to a single vertex, and if the vertex b_i is not the vertex with the smallest index in the set $B_{i'}$ such that $b_i \in B_{i'}$, i.e., $b_{i-1} \in B_{i'}$, then we contract each resulting vertex w to the vertex obtained in the $(i'-1)$-th subphase that is in the same layer as w (if such vertex exists).

Phase II. The graph obtained after Phase I has at most $k + \ell'$ vertices in each layer: k correspond to the vertices $a_1, \ldots, a_k$ of the graph H_0, i.e., they are contained on the paths $P_1, \ldots, P_k$, and the remaining ℓ' to the sets $B_1, \ldots, B_{\ell'}$ (see Figure 2). For every $i = 1, \ldots, k'$,

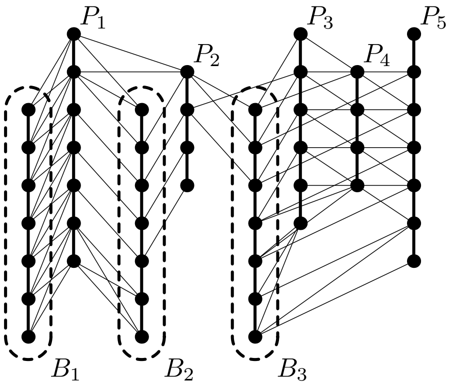


Figure 2 An example of a graph obtained after Phase I in the proof of Theorem 14 ($k = 5$ and $\ell' = 3$). The edges of vertical paths are drawn in bold. Note that there are no edges between paths corresponding to the set B_1, B_2 and B_3.

we contract all the vertices on the paths of A_i that are in the same layer to a single vertex as follows. Let $P_{i_1}, \dots, P_{i_n}$ be the paths corresponding to the vertices of A_i. We first contract the vertices of P_{i_1} and P_{i_2} that are in the same layer, proceeding from top to bottom (starting with the layer that contains both such vertices). We then contract the vertices of P_{i_3} to the vertices created previously, again in each layer proceeding from top to bottom, then the vertices of P_{i_4}, etc. At the end of this phase, we obtain a graph that is a subgraph of the strong product of a path and the graph H'_0. Since H'_0 has $k' + \ell' \leq 2s + 2$ vertices, each layer now contains at most $2s + 2$ vertices.

Phase III. We now contract all the vertices contained in the top layer to a single vertex, then all the vertices of the next layer to a single vertex, etc. Finally, we contract the vertices one after another to eventually obtain a single vertex, starting with the two vertices of the top two layers, then contracting the vertex in the third layer, etc.

Analysis of red degrees. We now establish an upper bound on the maximum possible red degree of the vertices of the graphs obtained throughout the described sequence of contractions. We start with Phase I. During the i-th subphase and the iteration for j, the only new red edges ever created are among the vertices of $W_1, \dots, W_m$ and the path Q_j. Since the vertices of $W_1 \cup \dots \cup W_m$ have at most 2^{24} different $(j+1)$-shadows (the neighbors in their shadows are only on the paths contained in C_j), each vertex has neighbors in its and the two neighboring layers, and we first contract all vertices of $W_1 \cup W_2$ with the same $(j+1)$-shadow, then all vertices of $W_1 \cup W_2 \cup W_3$, etc., the red degree of any vertex does not exceed $2 \cdot 3 \cdot 2^{24} = 3 \cdot 2^{25}$. We eventually arrive at having at most 2^{24} vertices in each layer and so their red degrees do not exceed $3 \cdot 2^{24}$. Then, the vertices with the same j-shadow that are on the same layer are contracted, which can result in the vertices of Q_j (temporarily) having the red degree up to $3 \cdot 2^{24}$. At the end of iteration for $j > n'$, there are at most 2^{21} vertices in each layer that have been obtained from $W_1 \cup \dots \cup W_m$ and so the red degree of each of them is at most $3 \cdot 2^{21}$. Also note that there is no red edge between the vertices on the paths $Q_1, \dots, Q_{j-1}$ and the remaining vertices of V_i.

At the beginning of the conclusion of the subphase, each layer has at most 2^{18} vertices obtained from contracting the vertices of V_i (note that this bound also holds when G_i has less than eight vertices, i.e., when we proceeded directly to the conclusion of the subphase). The conclusion of the subphase starts with contracting these vertices to a single vertex per layer: this can increase the red degree of vertices on at most six paths $P_1, \dots, P_k$ and the red degree of each vertex on these paths can increase by at most three. When the subphase finishes, each of the vertices contained in the paths $P_1, \dots, P_k$ has at most $3\ell'$ red neighbors (although during the subphase it can have upto three additional red neighbors), and each of the vertices obtained by contracting the vertices of $V_1, \dots, V_i$ has red degree at most $6s$ (since the sum of the degrees of the vertices in each set $B_1, \dots, B_{\ell'}$ is at most $2s$). In particular, the red degree of each vertex on the paths $P_1, \dots, P_k$ never exceeds $3(\ell' + 1)$. We conclude that the red degree of none of the vertices exceeds the largest of the following three bounds: $3 \cdot 2^{25}$, $3(\ell' + 1)$ and $6s$. Moreover, the red degree of no vertex exceeds $\max\{3\ell', 6s\} \leq 6s + 3$ (recall that $\ell' \leq 2s + 1$) at the end of each subphase (and so also at the end of Phase I).

During Phase II, each vertex has at most $\max\{k' + \ell', 2s\}$ red neighbors in its layer and in each of the neighboring layers. Indeed, the vertices obtained from those on the paths $P_1, \dots, P_k$ have at most $k' + \ell'$ red neighbors in each layer (at most k' neighbors among vertices obtained from contracting vertices on the paths $P_1, \dots, P_k$, and there are at most ℓ' vertices in each layer obtained by contracting vertices not on the paths $P_1, \dots, P_k$) and the vertices obtained from those not on the paths $P_1, \dots, P_k$ have at most $2s$ red neighbors in each layer as this is simply the upper bound on the number of their neighbors on the paths $P_1, \dots, P_k$. Hence, during the entire Phase II, the red degree of any vertex never exceeds

$$3 \max\{k' + \ell', 2s\} \leq 3 \max\{2(s + 1), 2s\} = 6(s + 1).$$

Finally, since the number of vertices contained in each layer at the end of Phase II is at most $k' + \ell'$, during the entire Phase III, the red degree of no vertex exceeds $3(k' + \ell') - 1$.

Hence, we have established that the red degree of no vertex exceeds $\max\left\{6(s + 1), 3 \cdot 2^{25}\right\}$ during the whole process, which implies the bound claimed in the statement. ◀

5 Algorithmic aspects

We now overview the main steps of the algorithm based on the proof of Theorem 14 that computes a witnessing sequence of vertex contractions of a graph embeddable in a fixed surface. We remark that we measure the time complexity in terms of the number of vertices, and we recall the number of edges of an n-vertex graph embeddable in a surface of Euler genus g is at most $3n + 3g - 6$, i.e., linear in the number of vertices when g is fixed.

Since it is possible to find an embedding of a graph in a fixed surface in linear time [43, 44], we can assume that the input graph G is given together with its embedding in the surface. When the embedding of G in the surface is fixed, we complete it to a triangulation G' (we permit adding parallel edges if needed). We next choose an arbitrary BFS spanning tree T of G' and identify g edges $a_1b_1, \dots, a_gb_g$ as described in Lemma 8, which was proven in [26, 28]. The proof of Lemma 8 in [26, 28] proceeds by constructing a spanning tree in the dual graph that avoids the edges of T and choosing the edges contained in neither T nor the spanning tree of the dual graph as the edges $a_1b_1, \dots, a_gb_g$; this can be implemented in linear time. When the edges $a_1b_1, \dots, a_gb_g$ are fixed, the construction of the walk W and the vertical paths described in Lemma 9 requires linear time.

We next compute the vertical paths described in Lemma 12 such that the boundary of each region of the graph G' obtained from the near-triangulation bounded by W can be covered by subpaths of at most six vertical paths. This requires processing the near-triangulation repeatedly following the steps of the inductive proof of Lemma 12: each step can be implemented in linear time and the number of steps is also at most linear. We then apply the recursive procedure described in the proof of Lemma 11 to each graph contained in one of the regions of G'; again, the number of steps in the recursive procedure is linear and each can be implemented in linear time. In this way, we obtain the collection $\mathcal{P}$ of vertical paths and the tree-decomposition $\mathcal{T}$ of $G'/\mathcal{P}$ described in Theorem 13. Note that the paths $\mathcal{P}$ and the tree-decomposition $\mathcal{T}$ fully determine the order of the contraction of the vertices and the order can be easily determined in linear time following the proof of Theorem 14.

We conclude that there is a quadratic time algorithm that constructs a sequence of contractions such that the red degree of trigraphs obtained during contractions does not exceed the bound given in Theorem 14. We remark that we have not attempted to optimize the running time of the algorithm, which would particularly require to implement the recursive steps of the proofs of Lemmas 11 and 12 more efficiently.

References

1 Jungho Ahn, Debsoumya Chakraborti, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Twin-width of random graphs, 2022. arXiv:2212.07880.

2 Jungho Ahn, Kevin Hendrey, Donggyu Kim, and Sang-il Oum. Bounds for the twin-width of graphs. *SIAM Journal on Discrete Mathematics*, 36:2352–2366, 2022.

3 M. Aigner and G.M. Ziegler. *Proofs from THE BOOK.* Spinger, 2010.

4 Jakub Balabán and Petr Hliněný. Twin-width is linear in the poset width. In *16th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 214 of *LIPIcs*, pages 6:1–6:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

5 Michael A. Bekos, Giordano Da Lozzo, Petr Hliněný, and Michael Kaufmann. Graph product structure for h-framed graphs. In *33rd International Symposium on Algorithms and Computation (ISAAC)*, volume 248 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

6 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. In *49th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229 of *LIPIcs*, pages 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022.

7 Benjamin Bergougnoux, Jakub Gajarský, Grzegorz Guśpiel, Petr Hliněný, Filip Pokrývka, and Marek Sokołowski. Sparse graphs of twin-width 2 have bounded tree-width, 2023. arXiv:2307.01732.

8 Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: Delineation and win-wins. In *17th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 249 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

9 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: Small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021.

10 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: Max independent set, min dominating set, and coloring. In *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

11 Édouard Bonnet, Colin Geniet, Romain Tessera, and Stéphan Thomassé. Twin-width VII: Groups, 2022. arXiv:2204.12330.

12 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: Ordered graphs and matrices. In *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC)*, pages 924–937, 2022.

13 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-Width V: Linear Minors, Modular Counting, and Matrix Multiplication. In *40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023)*, volume 254 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

14 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: The lens of contraction sequences. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1036–1056, 2022.

15 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. In *16th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 214 of *LIPIcs*, pages 10:1–10:16. 2021.

16 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO model checking. In *61th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 601–612, 2020.

17 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: Tractable FO model checking. *Journal of the ACM*, 69:3:1–3:46, 2022.

18 Édouard Bonnet, O-joung Kwon, and David R. Wood. Reduced bandwidth: a qualitative strengthening of twin-width in minor-closed classes (and beyond), 2022. arXiv:2202.11858.

19 Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations, 2021. arXiv:2102.06880.

20 Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. In *Proceedings of the 12th European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'23)*, EUROCOMB, pages 156–162, Brno, Czech Republic, 2023. Masaryk University Press.

21 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

22 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000.

23 Marc Distel, Robert Hickingbotham, Tony Huynh, and David R. Wood. Improved product structure for graphs on surfaces. *Discrete Mathematics & Theoretical Computer Science*, 24:8877:1–8877:10, 2022.

24 Jan Dreier, Jakub Gajarský, Yiting Jiang, Patrice Ossona de Mendez, and Jean-Florent Raymond. Twin-width and generalized coloring numbers. *Discrete Mathematics*, 345(3):112746, 2022.

25 Vida Dujmović, Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David R. Wood. Planar graphs have bounded nonrepetitive chromatic number. *Advances in Combinatorics*, pages 5:1–5:11, 2020.

26 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *Journal of the ACM*, 67(4):22:1–22:38, 2020.

27 Vida Dujmović, Pat Morin, and David R. Wood. Layered separators in minor-closed graph classes with applications. *Journal of Combinatorial Theory, Series B*, 127:111–147, 2017.

28 Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. In *60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 862–875, 2019.

29 Zdeněk Dvořák, Tony Huynh, Gwenael Joret, Chun-Hung Liu, and David R. Wood. Notes on graph product structure theory. In *2019-20 MATRIX Annals*, volume 4 of *MATRIX Book Series*, pages 513–533. Springer, 2021.

30 Zdeněk Dvořák, Daniel Král', and Robin Thomas. Deciding first-order properties for sparse graphs. In *51th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 133–142, 2010.

31 Zdeněk Dvořák, Daniel Král', and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM*, 60(5):36:1–36:24, 2013.

32 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposalbe graphs. In *26th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1644 of *Lecture Notes in Computer Science*, pages 331–340. Springer, 1999.

33 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, 48(6):1184–1206, 2001.

34 Jakub Gajarský, Michał Pilipczuk, and Szymon Toruńczyk. Stable graphs of bounded twin-width. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 39:1–39:12. ACM, 2022.

35 Petr Hliněný. Twin-width of planar graphs is at most 9, and at most 6 when bipartite planar, 2022. `arXiv:2205.05378`.

36 Petr Hliněný. Twin-width of planar graphs; a short proof, 2023. `arXiv:2302.08938`.

37 Petr Hliněný. Twin-width of planar graphs; a short proof. In *Proceedings of the 12th European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB'23)*, EUROCOMB, pages 595–600, Brno, Czech Republic, 2023. Masaryk University Press.

38 Petr Hliněný and Jan Jedelský. Twin-width of planar graphs is at most 8, and at most 6 when bipartite planar. In *50th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 261 of *LIPIcs*, pages 75:1–75:18. 2023.

39 J. Ivančo. The weight of a graph. *Annals of Discrete Mathematics*, 51:113–116, 1992.

40 Hugo Jacob and Marcin Pilipczuk. Bounding twin-width for bounded-treewidth graphs, planar graphs, and bipartite graphs. In *Graph-Theoretic Concepts in Computer Science (WG 2022)*, volume 13453 of *LNCS Lecture Notes Ser.*, pages 287–299. Springer International Publishing, 2022.

41 Jan Jedelský. *Twin-width of planar graphs*. Master's thesis, Masaryk University, 2024.

42 Daniel Král' and Ander Lamaison. Planar graph with twin-width seven. *European Journal of Combinatorics*, pages 103749, 16pp, 2023.

43 Bojan Mohar. Embedding graphs in an arbitrary surface in linear time. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 392–397. ACM, 1996.

44 Bojan Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, 1999.

45 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001.

46 William Pettersson and John Sylvester. Bounds on the twin-width of product graphs. *Discrete Mathematics & Theoretical Computer Science*, 25:1, 2023.

47 Michał Pilipczuk and Marek Sokołowski. Graphs of bounded twin-width are quasi-polynomially χ-bounded. *Journal of Combinatorial Theory, Series B*, 161:382–406, 2023.

48 Michał Pilipczuk, Marek Sokołowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. In *39th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 219 of *LIPIcs*, pages 52:1–52:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022.

49 Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

50 E Sperner. Fifty years of further development of a combinatorial lemma. *Numerical solution of highly nonlinear problems*, pages 183–197, 1980.

51 Stéphan Thomassé. A brief tour in twin-width. In *49th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 229 of *LIPIcs*, pages 6:1–6:5. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022.

52 Torsten Ueckerdt, David R. Wood, and Wendy Yi. An improved planar graph product structure theorem. *Electronic Journal of Combinatorics*, 29(2), 2022.

Agafonov's Theorem for Probabilistic Selectors

Ulysse Léchine
Université Sorbonne Paris Nord, France

Thomas Seiller
CNRS, Paris, France

Jakob Grue Simonsen
University of Copenhagen, Denmark

Abstract

A normal sequence over $\{0,1\}$ is an infinite sequence for which every word of length k appears with frequency 2^{-k}. Agafonov's eponymous theorem states that selection by a finite state selector preserves normality, i.e. if α is a normal sequence and A is a finite state selector, then the subsequence $A(\alpha)$ is either finite or a normal sequence.

In this work, we address the following question: does this result hold when considering probabilistic selectors? We provide a partial positive answer, in the case where the probabilities involved are rational. More formally, we prove that given a normal sequence α and a rational probabilistic selector P, the selected subsequence $P(\alpha)$ will be a normal sequence with probability 1.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Probabilistic computation; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases Normal sequences, probabilistic automata, Agafonov's theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.67

Funding *Thomas Seiller*: Partially supported by the ANR-22-CE48-0003-01 project DySCo.

1 Introduction

Let $\alpha = x_1 x_2 \cdots$ be an infinite sequence over $\{0,1\}$; α is said to be *normal* if every finite string of length n over Σ occurs with limiting frequency 2^{-n} in α [5]. By standard reasoning, almost all infinite sequences are normal when the set $\{0,1\}^\omega$ of infinite sequences is equipped with the usual Borel measure. Concrete examples of normal sequences include Champernowne's binary sequence $0100011011000001\cdots$ [11], and many more examples exist [7].

A *finite-state selector* is a deterministic finite automaton (DFA) that selects the nth symbol from α if the length $n-1$ prefix of α is accepted by the DFA. *Agafonov's Theorem* [15, 1] is the celebrated result that a sequence α is normal iff any DFA that selects an infinite sequence from α, selects a normal sequence. While alternative proofs, generalizations [20] – and counter-examples to generalizations [13] – abound, all results in the literature consider deterministic or non-deterministic DFAs, but none consider probabilistic computation.

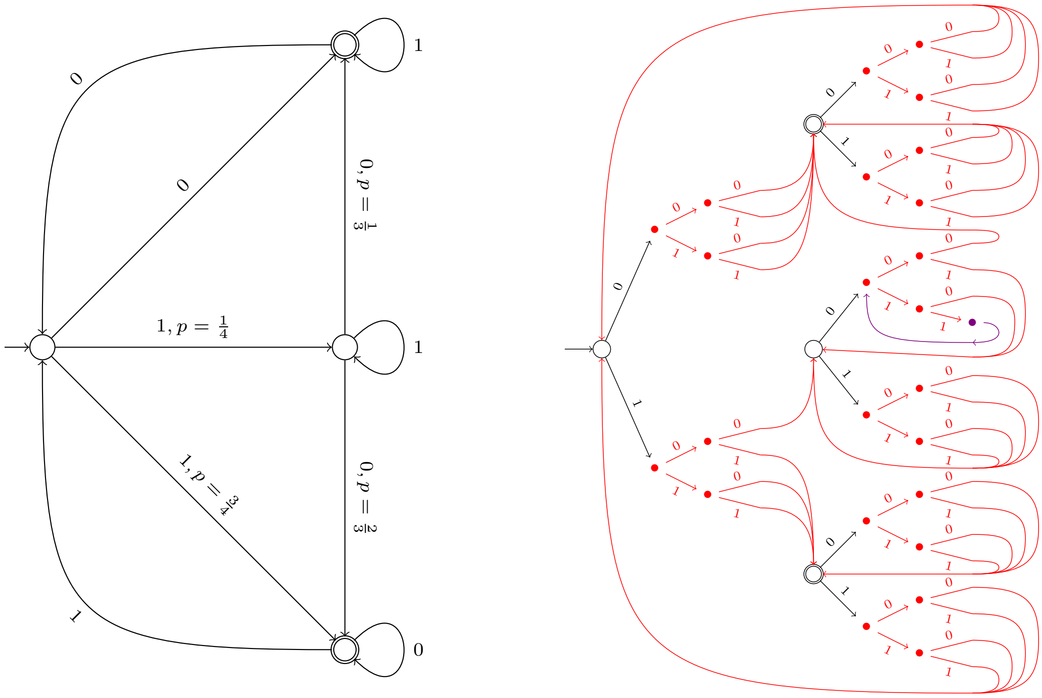
The extension to probabilistic selection is quite natural – not only are the underlying notions probabilistic in nature (i.e., normality of the transformed sequence), but the machinery of finite automata and similar computational devices itself has a 60-year history [18] of being extended to probabilistic devices.

In the present paper we study finite-state selectors equipped with probabilistic transitions from each state. As finite-state selectors can be viewed as devices sequentially processing successively larger prefixes of infinite sequences, we eschew the machinery of stochastic languages (where the initial state is a probability distribution on the states, and a string is accepted according to thresholding rules) – instead initial and accepting states are kept "as usual" in finite-state selectors. Probabilistic selection entails that normality may not be

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 67; pp. 67:1–67:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

preserved in all runs of an automaton: For example, consider an automaton with two states S_1 and S_2, only one of which is accepting, and transitions on 0 and 1 from S_i to S_i ($i \in \{1,2\}$) with probability $1/2$ and from S_i to $S_{i+1 \mod 2}$ with probability $1/2$; then for any normal sequence α, there is a run of the automaton on α that will select the sequence $0^\omega = 000\cdots$. The main result of the present paper is to show that the probability of having such runs is zero – in fact that for any probabilistic finite-state selector A with rational probabilities and any normal sequence α, *the probability that a run of A on α will select a normal sequence is 1.* The proof progresses by treating the relatively tame case of *dyadic* probabilities (i.e., of the form $a/2^k$ with a and k non-negative integers) first, and subsequently "simulating" finite-state selectors with arbitrary rational probabilities by "determinized" selectors with dyadic probabilities.

Figure 1 shows a probabilistic finite-state selector (Figure 1a) with two probabilistic transitions: one involves dyadic probabilities, the second one involves rational but non-dyadic probabilities. On the right-hand side (Figure 1b) is the determinization of this selector[1]. All unlabelled edges correspond to transitions valid for both 0 and 1. Determinizing the selector is done by introducing *gadgets* (shown in red) that simulate the probabilistic choices by drawing bits from a random advice sequence.



(a) A probabilistic selector.

(b) Determinization.

Figure 1 A probabilistic finite state selector and its determinization.
Unlabelled edges correspond to blind transitions, i.e. transitions valid for both 0 and 1.

[1] In fact, the determinisation as defined below would impose that all transition are represented as rationals of denominator 6 to ensure some regularity (this is discussed in Remark 20). For pedagogical purposes we however decided to show both a gadget for a dyadic transition and one for a rational but non-dyadic one.

Contributions. We prove that (the pertinent analogue of) Agafonov's Theorem holds in the setting of probabilistic selection, namely that a probabilistic finite state selector with rational probabilities preserves normality with probability 1. An added contribution is that the proof methods involved are novel, and may be of independent interest. As in Agafonov's original paper, and to keep complexity simple, all results are stated for binary alphabets. We fully expect all results to hold for arbitrary finite alphabets, *mutatis mutandis.*

Related work. Agafonov's Theorem has been generalized in multiple ways beyond finite automata (see, e.g.,[3, 2, 4, 10, 12, 20]). Conversely, it is known that when adding trifling computational expressivity to finite-state selectors, counterexamples to Agafonov's Theorem for the resulting selectors can be constructed [13]. While some existing work considers preservation of more general measures by finite automata, or similar selectors [9], and substantial work exists relating equidistribution to various types of automata [19, 4] no extant work considers stochastic *selection.* Agafonov's Theorem itself has been proved by a multitude of different techniques, e.g. [6, 8, 4]; it is conceivable that some of these can be adapted to alternative proofs, or extensions, of the results reported in the present paper.

2 Preliminaries and notation

Elements of $\{0,1\}^\omega$ are denoted by α, β, etc. Finite sequences (elements of $\{0,1\}^*$), or *words* are denoted by u, v, w, etc.

If $v, w \in \{0,1\}^*$, $v \cdot w$ denotes the concatenation of v and w; the definition extends to $v \cdot \alpha$ for $\alpha \in \{0,1\}^\omega$ *mutatis mutandis.*

The non-negative integers are denoted by $\mathbf{N}$, and the positive integers by $\mathbf{N}_{>0}$. If $N \in \mathbf{N}$ and $\alpha = a_1 a_2 \cdots \in \{0,1\}^\omega$, we denote by $\alpha|_{\leq N}$ the finite sequence $a_1 a_2 \cdots a_n$.

Given a set S we write $\mathrm{Dist}(S)$ the space of probability distributions on S. Given a probability distribution $\delta \in \mathrm{Dist}(S)$, we say that δ is *dyadic* (resp. *rational*) when for all $s \in S$, $\delta(s)$ is a dyadic number (resp. a rational number), that is a number of the form $\frac{p}{2^k}$ for integers p, k.

We consider the standard probability *measure* $\mathrm{Prob}_{\rho \in \{0,1\}^\omega}$ on $\{0,1\}^\omega$ equipped with the least Σ-algebra induced by the cylinder sets $C_w = \{\alpha \mid \exists \alpha' \in \{0,1\}^\omega, \alpha = w \cdot \alpha'\}$ and such that $\mathrm{Prob}_{\rho \in \{0,1\}^\omega} C_w = 2^{-|w|}$ for $w \in \{0,1\}^*$. Elements of $\{0,1\}^\omega$ drawn according to this measure are called *fair random infinite sequence.*

▶ **Definition 1.** *Let $a = a_1 \cdots a_m$ and $b = b_1 \cdots b_n$ be finite sequences such that $n < m$. An* occurrence *of b in a is an integer i with $1 \leq i \leq m$ such that $a_i = b_1, a_{i+1} = b_2, \dots a_{i+n-1} = b_n$. If $\alpha = a_1 a_2 \cdots$ is an infinite sequence and $w = w_1 \cdots w_n$ is a word, we denote by $\sharp_N\{w\}(\alpha)$ the number of occurrences of w in $a_1 a_2 \cdots a_N$.*

A sequence $\alpha \in \{0,1\}^\omega$ is said to be normal *if, for any $m \in \mathbf{N}$ and any $w = w_1 \cdots w_m \in \{0,1\}^m$, the limit $\lim_{N\to\infty} \sharp_N\{w\}(\alpha)/N$ exists and equals 2^{-m}.*

▶ **Definition 2.** *Let $\alpha = a_1 a_2 \cdots$ be an infinite sequence, and i and n be positive integers. The ith* block of size n in α, *denoted $B^i_n(\alpha)$, is the finite sequence $a_{(i-1)n+1} a_{(i-1)n+2} \cdots a_{in}$. If w is a finite sequence of length k with $k = jn + r$ for appropriate j and $r < n$, the ith block of size n in a finite sequence of length $k \geq n$ is defined* mutatis mutandis *for any $i \leq j$.*

Given a word $w \in \{0,1\}^n$, we write $\sharp^{(n)}_N\{w\}(\alpha)$ for the number of blocks of size n that are equal to w in the prefix of size $N \times n$:

$$\sharp^{(n)}_N\{w\}(\alpha) = \mathrm{Card}\{i \in [0, N-1] \mid B^i_n(\alpha) = w\},$$

for $k \in \mathbf{N}$ and $w \in \{0,1\}^k$. We define $\mathrm{freq}(\alpha, w)$ *as the following limit, when it exists:*

$$\mathrm{freq}(\alpha, w) = \lim_{N \to \infty} \frac{\sharp_N^{(n)}\{w\}(\alpha)}{N}.$$

Let $k \in \mathbf{N}$, the sequence is length-k*-normal if for all words $w \in \Sigma^k$,* $\mathrm{freq}(\alpha, w)$ *is well defined and equal to 2^{-k}.*

The sequence α is said to be block normal *if it is* length-k*-normal for all k.*

By standard results, an infinite sequence α is normal iff it is block-normal [14, 16, 17].

We now define the crucial notion of *probabilistic finite state selector*. The usual notion of deterministic finite state selectors is a special case of this definition.

▶ **Definition 3.** *A probabilistic finite state selector $\mathcal{S}$ is a tuple (Q, t, ι, A) where Q is a finite set of states, $\iota \in Q$ is the initial state, $A \subset Q$ is the subset of accepting states, and $t : Q \times \{0,1\} \to \mathrm{Dist}(Q)$ is a probabilistic transition function.*

A rational (resp. dyadic, resp. deterministic) finite selector is a probabilistic finite state selector in which all distributions $t(q, a)$ (for $q \in Q$ and $a \in \{0,1\}$) are rational (resp. dyadic, resp. deterministic).

Given a probabilistic selector $\mathcal{S}$ and a sequence $\alpha \in \{0,1\}^\omega$, one can define a probability distribution over $\{0,1\}^\omega$ defined through *selection* of elements of α by $\mathcal{S}$.

Observe that if $\mathcal{S}$ is deterministic, the induced probability distribution assigns probability 1 to *the unique* selected subsequence $\mathcal{S}(\alpha)$ of α considered in the standard Agafonov theorem, i.e. the sequence of bits α_i in α such that $\mathcal{S}$ reaches an accepting state when given the prefix $\alpha_0 \dots \alpha_{i-1}$.

▶ **Definition 4.** *Given a probabilistic finite state selector $\mathcal{S}$ and an infinite sequence α, the* selection random variable *$S(\alpha)$ is the random variable over $\{0,1\}^\omega$ defined as follows on cylindrical sets C_w:*

$$S(\alpha)(C_w) = \sum_{i_1 < \dots < i_{|w|} \in \mathbf{N}, \alpha_{i_1}\alpha_{i_2}\dots\alpha_{i_{|w|}} = w} \mathrm{Prob}(\mathcal{S}, i_1 < \dots < i_{|w|})$$

where $\mathrm{Prob}(\mathcal{S}, i_1 < \dots < i_{|w|})$ denotes the probability that the first $|w|$ times the selector $\mathcal{S}$ reaches an accepting state on input α correspond to the indices $i_1 - 1, i_2 - 1, \dots, i_{|w|} - 1$.

We finally recall the standard Agafonov theorem.

▶ **Theorem 5.** *Let $\alpha \in \{0,1\}^\omega$ be a sequence, and $\mathcal{S}$ a deterministic finite selector. Then α is normal if and only if for all deterministic finite selector $\mathcal{S}$, the subsequence $\mathcal{S}(\alpha)$ is either finite or normal.*

2.1 Technical lemmas about normality

We will establish a few results on normal sequences that will be useful in later proofs. We first define notions that will be used in the proofs.

▶ **Definition 6.** *Let $\alpha \in \{0,1\}^\omega$ be a sequence, and $w \in \{0,1\}^*$ a word. We say that α is* w-normal *if $\lim_{N \to \infty} \frac{\sharp_N\{w\}(\alpha)}{N} = 2^{-|w|}$.*

Given $\epsilon \in \mathbf{R}$, we say that α is w-normal up to ϵ *if $\exists N_0, \forall N > N_0, \left| \frac{\sharp_N\{w\}(\alpha)}{N} - 2^{-|w|} \right| < \epsilon$.*

We now restate a weaker property for sequences than normality: being normal for words of a fixed length k. The main lemma associated to that notion will be that if a sequence is normal for words of length mk for a fixed integer k and all integers m, then it is normal (i.e. normal for words of arbitrary length).

▶ **Definition 7.** *Let $k \in \mathbf{N}$, the sequence is* length-k-normal *if for all words $w \in \{0,1\}^k$, $\mathrm{freq}(\alpha, w)$ is well defined and equal to 2^{-k}.*

▶ **Lemma 8.** *Let α be a sequence in $\{0,1\}^\omega$. The following are equivalent:*

- *α is normal*
- *there exists $m \in \mathbf{N}_{>0}$ such that α is* length-km *normal for all $k \in \mathbf{N}$.*

Proof. Consider a sequence α which is length-km normal for all $k \in \mathbf{N}$, and fix a word $w \in \{0,1\}^n$. We will use the length-mn normality of α. For this, we note that we can write:

$$\begin{aligned}
\sharp^{(n)}_N\{w\}(\alpha) &= \sum_{w_1,\dots,w_m\in\{0,1\}^n} \mathrm{Card}\{i \in \{1,\dots,m\} \mid w_i = w\}.\sharp^{(mn)}_{N/m}\{w_1\cdot\dots\cdot w_m\}(\alpha)\\
&= \sum_{w_1,\dots,w_{m-1}\in\{0,1\}^n} \sum_{j=1}^{m} \sharp^{(mn)}_{N/m}\{w_1\cdot\dots\cdot w_{j-1}\cdot w\cdot w_j\cdot w_{m-1}\}(\alpha)\\
&= \sum_{w_1,\dots,w_{m-1}\in\{0,1\}^n} m\sharp^{(mn)}_{N/m}\{w\cdot w_1\cdot\dots\cdot w_{m-1}\}(\alpha)
\end{aligned}$$

As a consequence:

$$\begin{aligned}
\mathrm{freq}(\alpha, w) &= \lim_{N\to\infty} \frac{\sharp^{(n)}_N\{w\}(\alpha)}{N}\\
&= \lim_{N\to\infty} \sum_{w_1,\dots,w_m\in\{0,1\}^n} \frac{\mathrm{Card}\{i \in \{1,\dots,m\} \mid w_i = w\}.\sharp^{(mn)}_{N/m}\{w_1\cdot\dots\cdot w_m\}(\alpha)}{N}\\
&= \lim_{N\to\infty} \sum_{w_1,\dots,w_{m-1}\in\{0,1\}^n} \frac{m.\sharp^{(mn)}_{N/m}\{w\cdot w_1\cdot\dots\cdot w_{m-1}\}(\alpha)}{N}\\
&= \sum_{w_1,\dots,w_{m-1}\in\{0,1\}^n} \lim_{N\to\infty} \frac{\sharp^{(mn)}_{N/m}\{w\cdot w_1\cdot\dots\cdot w_{m-1}\}(\alpha)}{N/m}\\
&= \sum_{w_1,\dots,w_{m-1}\in\{0,1\}^n} \mathrm{freq}(\alpha, w\cdot w_1\cdot\dots\cdot w_{m-1})\\
&= \sum_{w_1,\dots,w_{m-1}\in\{0,1\}^n} 2^{-mn} = 2^{n(m-1)}2^{-mn} = 2^{-n}
\end{aligned}$$ ◀

Now the following lemma states that the proportion of blocks equal to a fixed word w in a prefix of size N of a normal sequence asymptotically behaves as a linear function. The proof is quite straightforward.

▶ **Lemma 9.** *Let α be a normal sequence and $w \in \{0,1\}^n$. Then $\sharp^{(n)}_N\{w\}(\alpha) = 2^{-n}N + o(N)$.*

Proof. If it were not true, we would have that there exists some $\epsilon > 0$ and a sequence $(N_i)_{i\in\mathbf{N}}$ such that $\left|\sharp^{(n)}_{N_i}\{w\}(\alpha) - 2^{-n}N_i\right| > \epsilon N_i$ for all $i \in \mathbf{N}$. In other words,

$$\left|\frac{\sharp^{(n)}_{N_i}\{w\}(\alpha)}{N_i} - 2^{-n}\right| > \epsilon.$$

This contradicts the normality of α since it implies that $\lim_{i\to\infty}\left|\frac{\sharp^{(n)}_{N_i}\{w\}(\alpha)}{N_i} - 2^{-n}\right| = 0$. ◀

We will now define a partition of the set of blocks $(B^j_K(\alpha))_{j\in\mathbf{N}}$ into groups $(E_i)_{i\in\mathbf{N}}$ such that each $r \in \{0,1\}^K$ appears exactly once in each E_i. Those will be defined from a partition $(V_i^K(\alpha))_{i\in\mathbf{N}}$ of $\mathbf{N}$ such that the set $V_i^K(\alpha)$ contains the indices of the blocks of size K of α contained in E_i.

▶ **Definition 10.** *Let $\alpha \in \{0,1\}^\omega$ be a normal sequence, and $K \in \mathbf{N}$. We define $\theta_i(\alpha) : \{0,1\}^K \to \mathbf{N}$ as mapping a word w to the value j such that $B^j_K(\alpha)$ is exactly the i-th block of size K of α equal to w. (i.e. there are exactly $i-1$ indices $j_1 < j_2 < j_3 < \ldots < j_{i-1} < j$ such that $\forall k$, $B^{j_k}_K(\alpha) = w \wedge B^j_K(\alpha) = w$)*

The sets of indices $(V_i^K(\alpha))_{i\in\mathbf{N}} \subset \mathbf{N}$ are then defined as the image $\operatorname{Im}(\theta_i(\alpha))$.

The next lemma gives useful bounds on the V_i^K.

▶ **Lemma 11.** *Let α be a normal sequence, $K \in \mathbf{N}$. Consider the sets $V_i^K(\alpha)$ from Definition 10. We have that $\max_{i=1}^N \max V_i^K(\alpha) = N2^K + o(N)$, and $|[N] \setminus \bigcup_{i=1}^{N/2^K} V_i| = o(N)$.*

Proof. This comes from the fact that for any $w \in \{0,1\}^n$, $\sharp_N^{(n)}\{w\}(\alpha) = 2^{-|w|}N + o(N)$. ◀

The following probabilistic lemma is needed for the proof of Lemma 30.

▶ **Lemma 12.** *Let Ω_1 and Ω_2 be two sample spaces. Let $(X_i)_{i\in\mathbf{N}}$ be an iid family of r.v. which take value in $\Omega_1^{\mathbf{N}}$, $(Y_i)_{i\in\mathbf{N}}$ another iid family of r.v. which take value in $\Omega_2^{\mathbf{N}}$. Let $f(X,Y)$ be a function in $\Omega_1 \times \Omega_2 \mapsto \mathbf{R}$. Suppose $\forall x \in \Omega_1, \forall y \in \Omega_2, f(X_0, y)$ and $f(x, Y_0)$ have finite expected value and variance then*

$$\mathbb{P}_{Y_i}\left(\sum_x \left[\mathbb{P}(X = x)\sum_{i=1}^N f(x, Y_i)\right] = N \times \mathbb{E}_{X_0,Y_0}(f(X_0,Y_0)) + o(N)\right) = 1.$$

Proof. This is a direct application of the law of large numbers . ◀

3 Dyadic case

We first restrict to dyadic selectors. Given a dyadic selector $S = (Q, t, \iota, A)$, we define its dyadicity degree as the smallest integer D such that for all states $q, q' \in Q$ and element $a \in \{0,1\}$, the probability $t(q,a)(q')$ can be written as $\frac{m}{2^D}$.

We first define the determinisation of a dyadic selector.

▶ **Definition 13.** *Given a dyadic selector $\mathcal{S} = (Q, t, \iota, A)$ of dyadicity degree D, we define a* determinisation $\operatorname{Det}(\mathcal{S})$ *of S as the deterministic selector (Q', t', ι', A') where:*

- *$Q' = Q \cup Q \times \{0,1\} \times \{0,1\}^{\leq D-1}$;*
- *$\iota' = \iota$ and $A' = A$;*
- *the transition function t is defined as follows:*
  - *for all $q \in Q$, $t'(q,a) = (q, a, \epsilon)$ where ϵ is the empty word;*
  - *for all $((q,b,w)$ with $w \in \{0,1\}^{\leq D-2}$, $t'((q,b,w),a) = (q,b,w\cdot a)$;*
  - *for all (q,b,w) with $w \in \{0,1\}^{D-1}$, $t'((q,b,w),a) = q'$ where $q' = \phi(w\cdot a)$ for a chosen $\phi_{q,b} : 2^D \to Q$ such that the preimage of any $s \in Q$ has cardinality m_s where m_s is defined by $t(q,b)(s) = \frac{m_s}{2^D}$.*

Now, the principle is that the behaviour of a dyadic selector S on the sequence α can be simulated by the behaviour of a determinisation $\operatorname{Det}^D(\mathcal{S})$ computing on an interleaving of α and a random *advice* string ρ.

▶ **Definition 14.** *Let α, ρ be sequences in $\{0,1\}^\omega$, and $D \in \mathbf{N}$. The interwoven sequence $I^D(\alpha,\rho)$ is defined as the sequence:*

$$\alpha_0\rho_0 \dots \rho_{D-1}\alpha_1\rho_D \dots \rho_{2D-1} \dots .$$

Note that the interweaving of two normal sequences can be a non-normal sequence, e.g. the interweaving of α with itself $I^1(\alpha,\alpha)$ is not normal.

▶ **Lemma 15.** *Let $\mathcal{S}$ be a dyadic selector of dyadicity degree D. Then for all sequences $\alpha \in \{0,1\}^\omega$ the random variables $\mathcal{S}(\alpha)$ and $\mathrm{Det}(\mathcal{S})(I^D(\alpha,\rho))$ where ρ is drawn uniformly at random in $\{0,1\}^\omega$ have the same distribution.*

Proof. This is a special case of Lemma 23. ◀

Consider given a normal sequence α. We now prove that for almost all random advice sequence $\rho \in \{0,1\}^\omega$, the interwoven sequence $I^D(\alpha,\rho)$ is normal. This is the key lemma in the proof of Theorem 17.

▶ **Lemma 16.** *Let $\alpha \in \{0,1\}^\omega$ be a normal sequence. Then for all $D \in \mathbf{N}$,*

$$\mathrm{Prob}_{\rho\in\{0,1\}^\omega}[I^D(\alpha,\rho) \textit{is normal}] = 1.$$

Proof. In case $D = 0$, the interwoven sequence $I^D(\alpha,\rho)$ is equal to α. As a consequence, $\mathrm{Prob}_{\rho\in\{0,1\}^\omega}[I^D(\alpha,\rho)$ is normal$]$ is equal to 1.

We now suppose that $D \neq 0$. Given $m \in \mathbf{N}_{>0}$, we will show that $I^D(\alpha,\rho)$ is length-$(D+1)m$ normal with probability 1. This implies that for almost all $\rho \in \{0,1\}^\omega$, the sequence $I^D(\alpha,\rho)$ is length-$(D+1)m$ normal for every $m \in \mathbf{N}_{>0}$. By Lemma 8, this implies that for almost all $\rho \in \{0,1\}^\omega$, the sequence $I^D(\alpha,\rho)$ is normal.

We now fix $m \in \mathbf{N}_{>0}$, and $w \in \{0,1\}^{(D+1)m}$. We will consider the block decomposition of $I^D(\alpha,\rho)$ into blocks of size $(D+1)m$ and prove that:

$$\mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{N\to\infty}\frac{\sharp_N^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{N} = 2^{-(D+1)m}\right) = 1.$$

We note that blocks of size $(D+1)m$ follow the pattern:

$$\alpha_i r_j \dots r_{j+D}\alpha_{i+1}r_{j+D}\dots r_{j+2D}\dots\alpha_{i+m-1}r_{j+(m-1)D}\dots r_{j+mD}.$$

We will consider $\lfloor w \rfloor_D = w_0 w_{D+1} w_{2(D+1)} \dots w_{(m-1)(D+1)}$ the subword of w corresponding to the positions of bits from α in this pattern.

We will consider the block decomposition of α into blocks of size m. Let $\mathrm{idx}(i) = j$ where j is the i-th block such that $B_j^m(\alpha) = \lfloor w \rfloor_D$. Note that this function is well defined because α is a normal sequence. Note that if a given block $B_i^{(D+1)m}(I^D(\alpha,\rho))$ is equal to w, then $\lfloor B_i^{(D+1)m}(I^D(\alpha,\rho))\rfloor_D$ should be equal to $\lfloor w \rfloor_D$. We write $\tilde{N} = \sharp_N^{(m)}\{\lfloor w \rfloor_D\}(\alpha)$, note that it is the maximal i such that $\mathrm{idx}(i) < N$. We also define $\bar{w}$ as the complementary subsequence of w:

$$\bar{w} = w_1 \dots w_D w_{D+2} \dots w_{2(D+1)-1} w_{2(D+1)+1} \dots w_{m(D+1)-1}.$$

We introduce a new notation: we will write $\sharp_{\mathrm{Im(idx)}<N}^{((D+1)m)}\{w\}(I^D(\alpha,\rho))$ to denote the number of blocks of size $(D+1)m$ equal to w within the blocks indexed by some $j < N$ in Im(idx).

$$P = \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{N\to\infty}\frac{\sharp_N^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{N} = 2^{-(D+1)m}\right)$$
$$= \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{N\to\infty}\frac{\sharp_{\mathrm{Im(idx)}<N}^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{N} = 2^{-(D+1)m}\right)$$

Now by Lemma 9 we have that $\lim_{N\to\infty}\tilde{N} = N.2^{-m}$. Hence:

$$P = \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp_{\mathrm{Im(idx)}<N}^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{\tilde{N}.2^{-m}} = 2^{-(D+1)m}\right)$$
$$= \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp_{\mathrm{Im(idx)}<N}^{((D+1)m)}\{w\}(I^D(\alpha,\rho))}{\tilde{N}} = 2^{-Dm}\right)$$
$$= \mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp_{\mathrm{Im(idx)}<N}^{(Dm)}\{\bar{w}\}(\rho)}{\tilde{N}} = 2^{-Dm}\right)$$

By the law of large numbers, we have that

$$\mathrm{Prob}_{\rho\in\{0,1\}^\omega}\left(\lim_{\tilde{N}\to\infty}\frac{\sharp_{\mathrm{Im(idx)}<N}^{(Dm)}\{\bar{w}\}(\rho)}{\tilde{N}} = 2^{-Dm}\right) = 1,$$

which concludes the proof. ◀

This lemma then leads to the following theorem.

▶ **Theorem 17.** *Let $\alpha \in \{0,1\}^\omega$ be a sequence. Then α is normal if and only if for all dyadic finite selector $\mathcal{S}$ the probability that $\mathcal{S}(\alpha)$ is either finite or normal is equal to 1.*

Proof. The right to left implication is simply a consequence of Agafonov's theorem (Theorem 5) since if for all dyadic finite selector $\mathcal{S}$ the probability that $\mathcal{S}(\alpha)$ is either finite or normal is equal to 1, then for all deterministic finite selector $\mathcal{S}$ the selected subsequence $\mathcal{S}(\alpha)$ is either finite or normal.

Now, suppose that the above implication from left to right is false. Then by Lemma 15 there exists a subset $R \subset \{0,1\}^\omega$ of strictly positive measure such that $\mathrm{Det}(\mathcal{S})(I^D(\alpha,\rho))$ is infinite and not normal for all $\rho \in R$. Since almost for almost all $\rho \in \{0,1\}^\omega$ the interwoven sequence $I^D(\alpha,\rho)$ is normal, this implies that there exists a ρ such that $I^D(\alpha,\rho)$ is normal and $\mathrm{Det}(\mathcal{S})(I^D(\alpha,\rho))$ is infinite and not normal. But this contradict Agafonov's theorem (Theorem 5). ◀

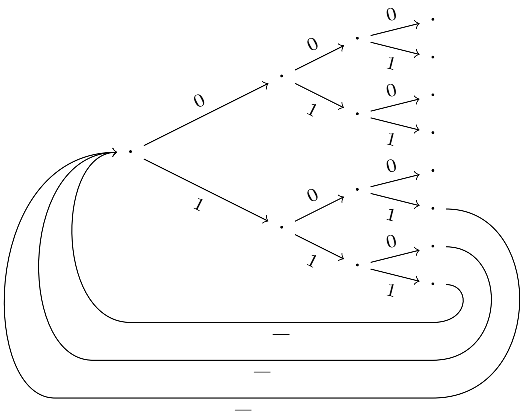
We will now consider the case of rational selectors. The difficulty in adapting the proof lies in the fact that the interwoven sequence has a less regular structure. In the above proof, each block of size $(D+1)m$ followed the same pattern. But in the case of rational selectors, the presence of feedback loops renders those pattern random, this makes the proof significantly harder. Indeed in the dyadic case the value of a block of size $(D+1)m$ was independent of the value of other blocks of size $(D+1)m$, in the rational case this is no longer true, thus we cannot apply the law of large numbers. Informally to make our proof work we divide $\mathcal{S}(\alpha)$ into non adjacent blocks whose values are independent, some bits are not contained in any blocks but we argue they are few of them and thus they don't prevent $\mathcal{S}(\alpha)$ from being normal.

4 Rational selector

4.1 Determinisation

The first step in extending the results to the rational case is to define the determinization. This will follow the same principle as for the dyadic case, but the parts of the determinized automaton that simulates probabilistic choices will contain feedback loops. Nonetheless, incorporating feedback loops is enough to represent any rational distribution, as shown in the next lemma.

▶ **Definition 18.** *A (k, f)-gadget is a regular binary tree of depth k, extended with blind transitions from the last F leaves to the root.*



▶ **Lemma 19.** *Any rational distribution* $\mathrm{Dist}(S)$ *is simulated by a gadget.*

Proof. Let $p_1, \dots, p_k$ be rationals with $\sum_i p_i = 1$, and suppose $p_i \leq p_{i+1}$ for all i. Consider M the smallest common multiple of all denominators of the elements p_i, and write $p_i = \frac{\tilde{p}_i}{M}$. We will denote by $q_i = \sum_{j=1}^{i} \tilde{p}_i$. Note that $q_0 = 0$ and $q_k = \sum_i \tilde{p}_i = M$. Now consider P the smallest natural number such that $2^P \geq M$. We build the regular automaton of depth P with feedback loops on $2^P - M$ leaves. We will show that the probability p of reaching a leaf within $[q_i + 1, q_{i+1}]$ is equal to p_i. One only need to compute:

$$p = \frac{\tilde{p}_i}{2^P} \sum_{m \geq 0} \left(\frac{2^P - M}{2^P} \right)^m = \frac{\tilde{p}_i}{2^P} \frac{1}{1 - \frac{2^P - M}{2^P}} = \frac{\tilde{p}_i}{2^P} \frac{2^P}{2^P - (2^P - M)} = \frac{\tilde{p}_i}{2^P} \frac{2^P}{M} = \frac{\tilde{p}_i}{M} = p_i \quad ◀$$

We will now define the determinisation of a rational selector in a similar way as for the dyadic case. First, since the selector is finite, one can write all rational numbers involved with a common denominator, say k. Given a rational selector $\mathcal{S}$, we will call k the *rationality degree* of $\mathcal{S}$. Then each transition will be simulated by a gadget as defined above.

▶ Remark 20. Note that since all rational distribution are represented with the same denominator, then all gadgets will have the same size. Indeed, let us define the *dyadicity degree* D of a rational selector $\mathcal{S}$ as the smallest integer such that $2^D \geq k$, where k is its rationality degree. Then the feedback edges of all gadgets corresponding to rational transitions correspond to the $2^D - k$ last edges in the gadget, and this does not depend on the specific transition considered.

The determinisation therefore has a quite *regular* structure which will be mirrored in the corresponding interwoven sequences.

▶ **Definition 21.** *Let $\mathcal{S} = (Q, \iota, t, A)$ be a rational selector of rationality degree k and dyadicity degree D. We define its* determinisation $\mathrm{Det}(\mathcal{S})$ *as the deterministic selector (Q', ι, t, A) where:*

- *$Q' = Q \cup Q \times \{0,1\} \times \big(\{0,1\}^{\leq k-1} \cup \{\text{return}\}\big)$;*
- *$\iota' = \iota$ and $A' = A$;*
- *the transition function t is defined as follows:*
  - *for all $q \in Q$, $t'(q, a) = (q, a, \epsilon)$ where ϵ is the empty list;*
  - *for all $((q, b, w)$ with $w \in \{0,1\}^{\leq k-2}$, $t'((q, b, w), a) = (q, b, w \cdot a)$;*
  - *for all (q, b, w) with $w \in \{0,1\}^{k-1}$ and $a \in \{0,1\}$:*
    - *if $w \cdot a$ belongs to the $2^D - k$ last leaves (i.e. the $2^D - k$ largest elements of $\{0,1\}^D$ for the natural order), then $t'((q, b, w), a) = (q, b, \text{return})$;*
    - *otherwise, $t'((q, b, w), a) = q'$ where $q' = \phi(w \cdot a)$ for a chosen $\phi_{q,b} : k \to Q$ such that the preimage of any $s \in Q$ has cardinality m_s where m_s is defined by $t(q, b)(s) = \frac{m_s}{k}$;*
  - *for all (q, b, return) and any $a \in \{0,1\}$, $t'(q, b, \text{return}) = (q, b, \epsilon)$.*

▶ **Definition 22.** *Let $\mathcal{S}$ be a rational selector of rationality degree k and dyadicity degree D, $\alpha \in \{0,1\}^\omega$ an input sequence, and $\rho \in \{0,1\}^\omega$ an* advice *sequence. We define the interwoven sequence $I_k^D(\alpha, \rho)$ as:*

$$\alpha_1 \rho_1 \dots \rho_{i_1} \alpha_2 \rho_{i_1+1} \dots \rho_{i_2} \alpha_3 \dots,$$

where $i_1 < i_2 < \dots$ is the sequence of indices i_j such that $\rho_{i_j+1} \dots \rho_{i_{j+1}}$ is equal to $w_1 r_1 w_2 r_2 \dots r_m w_{m+1}$ where:

- *$w_1, w_2, \dots, w_m$ are among the $2^D - k$ greatest elements in $\{0,1\}^D$ (considered with the natural alphabetical order);*
- *w_{m+1} belongs to the k smallest elements in $\{0,1\}^D$;*
- *r_i are bits in $\{0,1\}$ which we will call* return bits*, corresponding to feedback loops.*

We note that this is a direct generalisation of the dyadic case, i.e. if the considered selector is dyadic, then the interwoven sequence $I_{2^D}^D$ just defined coincides with the definition from the previous section. Similarly, the determinisation of a dyadic selector is a special case of the determinisation of a rational selector. We can see here the difficulty in adapting the proof to the rational case arising: instead of interweaving one block of ρ of size D between each bit of α, we interweave a block of bits from ρ of variable length.

Note however that we carefully defined the determinisation so that the size of these blocks does not depend on the values α_i. Moreover, feedback loops introduce random *return* bits, allowing us to write the interwoven sequence as a sequence of blocks of the form $a r_1 \dots r_D$ where a is either a bit from α or a return bit from ρ and $r_1 \dots r_D$ are bits from ρ.

First, we check that the determinisation simulates the rational selector when given random advice strings.

▶ **Lemma 23.** *Let $\mathcal{S}$ be a rational selector of rationality degree k and dyadicity degree D. Then for all sequence $\alpha \in \{0,1\}^\omega$ the random variables $\mathcal{S}(\alpha)$ and $\mathrm{Det}(\mathcal{S})(I_k^D(\alpha, \rho))$ where ρ is an infinite fair random sequence, have the same distribution.*

Proof. Let ρ be an infinite fair random sequence. By construction of $\mathrm{Det}(\mathcal{S})(I_k^D(\alpha, \rho))$, for a any two state q and q' the probability of going from q to q' in $\mathrm{Det}(\mathcal{S})(I_k^D(\alpha, \rho))$ (ignoring the gadget states in between) is equal to the probability of going from q to q' in $\mathcal{S}(\alpha)$. ◀

4.2 Rational selectors preserve normality

In the following, α will be a infinite sequence, not considered normal unless explicitly stated. We will write w to denote a finite word. We denote by ρ and τ fair infinite random sequences, and by r a finite random sequence. Lastly, q will be the probability to loop back at the end of a gadget, equal to $1 - \frac{k}{2^D}$. We will denote by $A(N) \xrightarrow{N} B(N)(1 \pm \epsilon)$ the fact that

$$\exists N_0, \forall N > N_0, B(N)(1-\epsilon) < A(N) < B(N)(1+\epsilon).$$

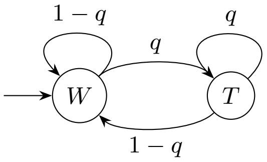
To prove that rational selectors preserve normality, we will prove in this section that $\mathbb{P}_\rho(I_k^D(\alpha,\rho)\text{is normal}) = 1$, that is the generalised version of Lemma 16. As in the dyadic case, this is the crux of the problem, and the proof of the main theorem will easily follow. In order to prove this technical lemma, we analyze a process we call $\mathcal{F}$ which takes a sequence α and inserts in between every bit of α a random amount of random bits. We will then show that if α is normal the sequence $\mathcal{F}(\alpha)$ obtained in this way is normal. Finally we will argue that normality of $I_k^D(\alpha,\rho)$ amounts to the normality of $\mathcal{F}(\alpha)$.

▶ **Definition 24** (Random process $\mathcal{F}_q$)**.** *Suppose given $K \in \mathbf{N}$, $w \in \{0,1\}^K$, $q \in [0;1[$, and $\tau \in \{0,1\}^\omega$. We define $\mathcal{F}_q(w,\tau) \in \{0,1\}^*$ as the random variable described in Figure 2 where we consume a bit of w when we get to state W and a bit of τ when we get to state T. The process stops when the state W is reached and there are no more bits of w to be consumed. The output is all the consumed bit in timely order.*

We denote by $\mathcal{F}_q(w)$ the random variable $\mathcal{F}_q(w,\tau)$ where τ is a fair random infinite sequence.

In the following, we may not specify q and just write $\mathcal{F}(w,\tau)$ when the context is clear.

▶ Remark 25. Note that τ needs to be infinite because we have no bound on how many bits of it we may consume.



Example: if $w = 0110$, $\tau = 10010...$, then

$$\mathcal{F}(w) = 010110010,$$

with the sequence of states

$$WTTWWTWTTW.$$

Figure 2 The random process $\mathcal{F}$.

For now $\mathcal{F}$ has only been defined on finite strings. We extend it to infinite strings in an intuitive way.

▶ **Definition 26.** *Suppose given $\alpha \in \{0,1\}^\omega$, $K \in \mathbf{N}$, and $q \in \mathbf{R}$. Let $(\mathcal{F}_i)_{i\in\mathbf{N}}$ be an iid family of random variables of law $\mathcal{F}$. The random variable $\mathcal{F}_q(\alpha)$ is the infinite sequence distributed as the concatenation of the $\mathcal{F}_i$ applied to the blocks $B_K^i(\alpha)$:*

$$\mathcal{F}_0(B_K^0(\alpha))\mathcal{F}_1(B_K^1(\alpha))\mathcal{F}_2(B_K^2(\alpha))\ldots.$$

Note that the value of K does not change the distribution of the random variable $\mathcal{F}(\alpha)$, hence the definition is unambiguous.

In the next lemma we analyze the length of $\mathcal{F}(w)$.

▶ **Lemma 27.** *Let $q \in [0;1[$ and $(\mathcal{F}_i)$ be an iid family of random variables of law $\mathcal{F}_q$. Then for all $K \in \mathbf{N}$ and for any family $(w_i)_{i\in\mathbf{N}} \in (\{0,1\}^K)^{\mathbf{N}}$,*

$$\mathbb{P}\left(\sum_{i\leq N} |\mathcal{F}_i(w_i)| = NKq^{-1} + o(NKq)\right) = 1.$$

Proof. By standard Markov chain analysis, the expected value of $|\mathcal{F}_i(w_i)|$ is Kq^{-1} and its variance is finite, furthermore the $\mathcal{F}_i(w_i)$ are independent the strong law of large number therefore applies and we get the desired result. ◀

In the next lemma we show that for any w we can approximate the number of w in $\mathcal{F}(\alpha) = \mathcal{F}_0(B^0_K(\alpha))\mathcal{F}_1(B^1_K(\alpha))\mathcal{F}_2(B^2_K(\alpha))\dots$ by adding up the number of w in each $\mathcal{F}_i(B^i_K(\alpha))$ separately. The larger K the more precise the approximation. What we gain from this separation is that the random variable $\sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha)))$ are independent and we can apply the law of large numbers. In contrast in $\mathcal{F}(\alpha)$ where we concatenate the $\mathcal{F}_i(B^i_K(\alpha))$ we do not have independence because knowing that w appears at the end of $\mathcal{F}_1(B^1_K(\alpha))$ may influence that it appears at the beginning of $\mathcal{F}_2(B^2_K(\alpha))$.

▶ **Lemma 28.** *Let $\alpha \in \{0,1\}^\omega$ be a normal sequence, $w \in \{0,1\}^M$ be a word, and $(\mathcal{F}_i)_{i\in\mathbf{N}}$ be an iid family of random variables of law $\mathcal{F}$. For all $K \in \mathbf{N}$, we write $\beta = \mathcal{F}_0(B^0_K(\alpha))\mathcal{F}_1(B^1_K(\alpha))\mathcal{F}_2(B^2_K(\alpha))\dots$ and for all i we define $s_i = |\mathcal{F}_i(B^i_K(\alpha))|$ and $S_N = \sum_{i=0}^N s_i$. Then we have that*

$$\left[\sum_{i=0}^N \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha)))\right] - \sharp_{S_N}\{w\}(\beta) < MN.$$

Proof. First note that we count indices up to $s_i - |w|$ in $\sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha)))$ because if w appears in $\mathcal{F}_i(B^i_K(\alpha))$ it must appear before the last $|w|$ bits. For this reason we also mention that $\sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha))) = \sharp_{|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha)))$.

Then note that $\sharp_{S_N}\{w\}(\beta) \geq \left[\sum_{i=0}^N \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha)))\right]$ indeed if w appears somewhere in one of the $\mathcal{F}_i(B^i_K(\alpha))$ then it also appears in β.

Therefore every w is counted in $\sharp_{S_N}\{w\}(\beta)$ and not in $\left[\sum_{i=0}^N \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha)))\right]$ appears at an index in the $|w|$ last bits of an $F_i(B^i_K(\alpha))$. There are at most $|w| \times N = MN$ of those. ◀

We have that $\sum_{i=0}^N |\mathcal{F}_i(B^i_K(\alpha))|$ tends to NKq^{-1}, thus by taking large values of K the discrepancy MN of the number of w noticed in the previous theorem can be made negligible when compared to the size of the string.

In the next theorem we just prove that for a random ρ the proportion of w in $\mathcal{F}_i(B^i_K(\rho))$ is approximately $2^{-|w|}|\mathcal{F}_i(B^i_K(\rho))|$ on average.

▶ **Lemma 29.** *Suppose given $\rho \in \{0,1\}^\omega$ a fair random infinite sequence, $q \in [0;1[$, and $w \in \{0,1\}^M$. Let $(\mathcal{F}_i)_{i\in\mathbf{N}}$ be an iid family of random variables of law $\mathcal{F}_q$. For all i, we write $s_i = |\mathcal{F}_i(B^i_K(\rho))|$ and for all N, $S_N = \sum_{i=0}^N s_i$. Then for any $\epsilon > 0$, there exists $K \in \mathbf{N}$ such that:*

$$\left|\mathbb{E}(\sharp_{s_i-|w|}\{K\}(\mathcal{F}_i(B^i_w(\rho)))) - 2^{-|w|}Kq^{-1}\right| < \epsilon.$$

Proof. This result can be shown by standard analysis of fair random sequences of size Kq^{-1}. Indeed for a random ρ, $\mathcal{F}_i(B^i_K(\rho))$ is just a random sequence of expected size Kq^{-1}. Let $\epsilon \in \mathbf{R}$. If K is large enough, there exists some $\epsilon' \in \mathbf{R}$ such that a random sequence of size Kq^{-1} contains on average $2^{-|w|}Kq^{-1} + \epsilon'$ occurrences of w where $|\epsilon'| < \epsilon$. ◀

▶ **Lemma 30.** *Let α be a normal sequence, for any $w \in \{0,1\}^*$ and $q \in [0,1[$, $\mathcal{F}_q(\alpha)$ is w-normal with probability 1.*

Proof of Lemma 30. Let $(\mathcal{F}_i)_{i\in\mathbf{N}}$ be random independent processes $\mathcal{F}$. Let α be a normal sequence. Let $w \in \{0,1\}^*$, $\epsilon' \in \mathbf{R}$, and $q \in [0;1[$. Then

$$\mathbb{P}\left(\mathcal{F}_q(\alpha)\text{is } w\text{-normal up to } \epsilon'\right) = 1 \Leftrightarrow \mathbb{P}\left(\lim_N \frac{\sharp_N\{w\}(\mathcal{F}_q(\alpha))}{N} = 2^{-|w|} \pm \epsilon\right) = 1.$$

Using Lemma 28 by introducing independent random variables $\mathcal{F}_i$ of law $\mathcal{F}_q$, and writing $s_i = |\mathcal{F}_i(B^i_K(\alpha))|$, the above result is implied by:

$$\forall\epsilon, \exists K, \mathbb{P}\left(\sum_{i\leq N} \sharp_{s_i-|w|}\{w\}(\mathcal{F}_i(B^i_K(\alpha))) \xrightarrow{N} 2^{-|w|}KNq^{-1}(1\pm\epsilon)\right) = 1.$$

Take $V_i(\alpha)$ as defined in definition 10. Call $B_N = \{i \in [1; \frac{N}{2^K}] \mid \max(V_i(\alpha)) < N\}$. We group the indices of blocks $B^j_K(\alpha)$ into sets V_i of size 2^K and such that $|V_i| = 2^K$. We may also change $s_j - |w|$ to s_j as explained in the proof of Lemma 28. Then the above is equivalent to

$$\forall\epsilon, \exists K, \mathbb{P}\left(S_1 + S_2 \xrightarrow{N} 2^{-|w|}KNq^{-1}(1\pm\epsilon)\right) = 1,$$

where

$$S_1 = \sum_{i\in B_N}\sum_{j\in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))), \qquad S_2 = \sum_{j\in[N]\setminus\bigcup_{i\in B_N} V_i} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))).$$

By Lemma 11, we have that $|B_N| = \frac{N}{2^K} + g(N)$, where $g(N) = o(N)$. The equation can thus be further rewritten as:

$$\mathbb{P}\left(T_1 + T_2 + T_3 \xrightarrow{N} 2^{-|w|}KNq(1\pm\epsilon)\right) = 1,$$

where:

$$T_1 = \sum_{i\in[N/2^K]}\sum_{j\in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))),$$

$$T_2 = \sum_{i=1+N/2^K}^{\frac{N}{2^K}+g(N)}\sum_{j\in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))),$$

$$T_3 = \sum_{j\in[N]\setminus\bigcup_{i\in B_N} V_i} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))).$$

We now consider each term separately.

The term T_1. By construction of the V_i, as j ranges across all values in V_i, $B^j_K(\alpha)$ takes all values in $\{0,1\}^K$. By creating an appropriate bijection between j and (i,r), we can write

$$\sum_{i\in[\frac{N}{2^K}]}\sum_{j\in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))) = \sum_{i\in[\frac{N}{2^K}]}\sum_{r\in\{0,1\}^K} \sharp_{s_{i,r}}\{w\}(\mathcal{F}_{i,r}(r)).$$

We recognize a sum over expectations as in Lemma 12. By Lemma 29, we can take K big enough such that the expected value of $\sharp_{s_{i,r}}\{w\}(\mathcal{F}_{i,r}(r))$ is $Kq^{-1}2^{-|w|}(1\pm\epsilon)$. Thus:

$$\forall\epsilon, \exists K, \mathbb{P}\left(\sum_{i\in[\frac{N}{2^K}]}\sum_{j\in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B^j_K(\alpha))) \xrightarrow{N} 2^{-|w|}KNq^{-1}(1\pm\epsilon)\right) = 1.$$

The term T_2. We have that

$$\forall \epsilon, \forall K, \mathbb{P}\left(\left[\sum_{i=1+N/2^K}^{\frac{N}{2^K}+g(N)} \sum_{j \in V_i(\alpha)} \sharp_{s_j}\{w\}(\mathcal{F}_j(B_K^j(\alpha)))\right] = o(N)\right) = 1,$$

because $g(N) = o(N)$ and the random variable $\sharp_{s_j}\{w\}(\mathcal{F}_j(B_K^j(\alpha)))$ has finite expected value and variance (in particular constant in N).

The term T_3. We have that

$$\mathbb{P}\left(\sum_{j \in [N] \setminus \bigcup_{i \in B_N} V_i} \sharp_{s_j}\{w\}(\mathcal{F}_j(B_K^j(\alpha))) = o(n)\right) = 1.$$

The sum is over $o(N)$ terms by Lemma 11 and the random variable $\sharp_{s_j}\{w\}(\mathcal{F}_j(B_K^j(\alpha)))$ is of finite expected value and variance.
By combining the three results we can get that

$$\forall \epsilon' \in \mathbf{R}, \mathbb{P}(\mathcal{F}(\alpha)\text{is } w\text{-normal up to } \epsilon') = 1.$$

We define the sequence $(\epsilon_n)_{n \in \mathbf{N}} \in \mathbf{R}^{\mathbf{N}}$ as $\epsilon_n = 1/(n+1)$. We have that

$$\mathbb{P}(\forall n, \mathcal{F}(\alpha)\text{is } w\text{-normal up to } \epsilon_n) = 1$$

as an intersection of countably many events of probability 1. We then get, using the fact that all Cauchy sequences converge on $\mathbf{R}$, that $\mathbb{P}(\mathcal{F}(\alpha)\text{is } w\text{-normal}) = 1$. ◀

▶ **Lemma 31.** *Let α be a normal number then $\mathcal{F}(\alpha)$ is normal with probability 1.*

Proof. By lemma 30 $\forall w \in \{0,1\}^*$, $\mathcal{F}(\alpha)$ is w-normal with probability 1. $\mathbb{P}(\mathcal{F}(\alpha)\text{is normal}) = \mathbb{P}(\forall w \in \{0,1\}^*, \mathcal{F}(\alpha)\text{is } w\text{-normal})$, since this is an intersection of countably many event of probability 1, we have that $\mathcal{F}(\alpha)$ is normal with probability 1. ◀

▶ **Lemma 32.** *For any positive integer D, any $k \in [2^{D-1}; 2^D]$*

$$\mathbb{P}_\rho(I_k^D(\alpha, \rho)\textit{is normal}) = 1.$$

Proof. Let D be a positive integer, $k \in [2^{D-1}; 2^D]$. There are 3 kinds of bits in $I_k^D(\alpha, \rho)$: bits from α, bits from ρ appearing inside the gadgets (we call this sequence γ) and bits from ρ corresponding to return bits (we call this infinite sequence of bits τ). Note that γ and τ are both independent fair random infinite sequences.

Note that in $I_k^D(\alpha, \rho)$, we find every bit from α and τ at indices multiple of $D+1$. We define the infinite sequence y as such: $\forall i \in \mathbf{N}, y_i = I_k^D(\alpha, \rho)_{i(D+1)}$.

Notice that $I_k^D(\alpha, \rho) = I^D(y, \gamma)$ (where the second I is from defintion 14). Since γ is a fair infinite random sequence then by using theorem 16 if y is normal then so is $I_k^D(\alpha, \rho)$ with probability 1 over γ.

Thus now we only need to show that y is normal with probability 1. Notice that the distribution of y is the same as $\mathcal{F}_q(\alpha)$ with $q = 1 - \frac{k}{2^D}$. Therefore by theorem 31 y is normal with probability 1. ◀

This gives the main theorem. The proof follows the proof of Theorem 17, using Lemma 32 and and Lemma 23.

▶ **Theorem 33.** *Let $\alpha \in \{0,1\}^\omega$ be a sequence. Then α is normal if and only if for all rational selector $\mathcal{S}$ the probability that $\mathcal{S}(\alpha)$ is either finite or normal is equal to 1.*

While we think the equivalent statement to hold for general probabilistic selectors, we believe that establishing such a result would require a different proof method.

References

1 V. N. Agafonov. Normal sequences and finite automata. *Sov. Math., Dokl.*, 9:324–325, 1968. Originally published in Russian [15].

2 Dylan Airey and Bill Mance. Normality preserving operations for Cantor series expansions and associated fractals, i. *Illinois J. Math.*, 59(3):531–543, 2015.

3 J. Auslander and Y. N. Dowker. On disjointness of dynamical systems. *Mathematical Proceedings of the Cambridge Philosophical Society*, 85(3):477–491, 1979.

4 Veronica Becher, Olivia Carton, and Pablo Ariel Heiber. Normality and automata. *Journal of Computer and System Sciences*, 81(8):1592–1613, 2015.

5 Émile Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rend. Circ. Matem. Palermo*, 27:247–271, 1909.

6 A. Broglio and P. Liardet. Predictions with automata. symbolic dynamics and its applications. *Contemporary Mathematics*, 135:111–124, 1992. Also appeared in Proceedings of the AMS Conference in honor of R. L. Adler. New Haven CT – USA 1991.

7 Yann Bugeaud. *Distribution modulo one and Diophantine approximation.* Cambridge Tracts in Mathematics. Cambridge University Press, 2012.

8 Olivier Carton. A direct proof of agafonov's theorem and an extension to shift of finite type. *CoRR*, abs/2005.00255, 2020. `arXiv:2005.00255`.

9 Olivier Carton. A direct proof of Agafonov's theorem and an extension to shifts of finite type. *Preprint*, 2020.

10 Olivier Carton and Joseph Vandehey. Preservation of normality by non-oblivious group selection. *Theory of Computing Systems*, 2020.

11 D. G. Champernowne. The construction of decimals normal in the scale of ten. *Journal of the London Mathematical Society*, s1-8(4):254–260, 1933.

12 T. Kamae and B. Weiss. Normal numbers and selection rules. *Israel Journal of Mathematics*, pages 101–110, 1975.

13 Wolfgang Merkle and Jan Reimann. Selection functions that do not preserve normality. *Theory Comput. Syst.*, 39(5):685–697, 2006.

14 Ivan Niven and H. S. Zuckerman. On the definition of normal numbers. *Pacific J. Math.*, 1(1):103–109, 1951.

15 В. Н. Агафонов. Нормальные последовательности и конечные автоматы. Докл. АН СССР, 179(2):255–256, 1968.

16 Л. П. Постникова. О связи понятий коллектива Мизеса–Черча и нормальной по Бернулли последовательности знаков. Теория вероятн. и ее примен., 6(2):232–234, 1961.

17 LP Postnikova. On the connection between the concepts of collectives of Mises-Church and normal Bernoulli sequences of symbols. *Theory of Probability & Its Applications*, 6(2):211–213, 1961. translation of [16] by Eizo Nishiura.

18 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

19 Claus-Peter Schnorr and H. Stimm. Endliche Automaten und Zufallsfolgen. *Acta Informatica*, 1:345–359, 1972.

20 Xiaowen Wang and Teturo Kamae. Selection rules preserving normality. *Israel Journal of Mathematics*, 232:427–442, 2019.

Unweighted Geometric Hitting Set for Line-Constrained Disks and Related Problems

Gang Liu ✉
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang ✉ 🏠
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given a set P of n points and a set S of m disks in the plane, the disk hitting set problem asks for a smallest subset of P such that every disk of S contains at least one point in the subset. The problem is NP-hard. This paper considers a line-constrained version in which all disks have their centers on a line. We present an $O(m\log^2 n+(n+m)\log(n+m))$ time algorithm for the problem. This improves the previous result of $O(m^2\log m+(n+m)\log(n+m))$ time for the weighted case of the problem where every point of P has a weight and the objective is to minimize the total weight of the hitting set. Our algorithm also solves a more general line-separable problem with a single intersection property: The points of P and the disk centers are separated by a line ℓ and the boundary of every two disks intersect at most once on the side of ℓ containing P.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases hitting set, line-constrained, line-separable, unit disks, half-planes, coverage

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.68

Related Version *Full Version*: `http://arxiv.org/abs/2407.00331`

Funding This research was supported in part by NSF under Grant CCF-2300356.

1 Introduction

Let P be a set of n points and S a set of m disks in the plane. The *hitting set* problem is to compute a smallest subset of P such that every disk in S contains at least one point in the subset (i.e., every disk is *hit* by a point in the subset, and the subset is called a *hitting set*). The problem is NP-hard, even if all disks have the same radius [18, 25]. Polynomial-time approximation algorithms are known for the problem, e.g., [3, 15, 16, 21, 24, 25].

In this paper, we consider the *line-constrained* version of the problem, where centers of all disks are on a line while the points of P can be anywhere in the plane. The weighted case of the problem was studied by Liu and Wang [22], where each point of P has a weight and the objective is to minimize the total weight of the hitting set. Their algorithm runs in $O((m+n)\log(m+n)+\kappa\log m)$ time, where κ is the number of pairs of disks that intersect and $\kappa=O(m^2)$ in the worst case. They reduced the runtime to $O((m+n)\log(m+n))$ for the *unit-disk case*, where all disks have the same radius [22]. Our problem in this paper is for the unweighted case. To the best of our knowledge, we are not aware of any previous work that particularly studied the unweighted hitting set problem for line-constrained disks. We propose an algorithm of $O(m\log^2 n+(n+m)\log(n+m))$ time, which improves the weighted case algorithm of $O(m^2\log m+(n+m)\log(n+m))$ worst-case time [22]. Perhaps theoretically more interesting is that the worst-case runtime of our algorithm is near linear.

1.1 Related work

A closely related problem is the disk coverage problem, which is to compute a smallest subset of S that together cover all the points of P. This problem is also NP-hard because it is dual to the hitting set problem in the unit-disk case (i.e., all disks have the same radius). Polynomial-time algorithms are known for certain special cases, e.g., [1,4,9,10]. In particular, the line-constrained problem (in which all disks are centered on a line) was studied by Pedersen and Wang [26]. Their algorithm runs in $O((m+n)\log(m+n)+\kappa\log m)$ time, where κ is the number of pairs of disks that intersect and $\kappa=O(m^2)$ in the worst case; they also solved the unit-disk case in $O((m+n)\log(m+n))$ time. As noted above, in the unit-disk case, the coverage and hitting set problems are dual to each other and therefore the two problems can essentially be solved by the same algorithm. However, this is not the case if the radii of the disks are different.[1]

In addition, the $O((m+n)\log(m+n)+\kappa\log m)$ time algorithm of Pedersen and Wang [26] also works for the weighted *line-separable unit-disk* version, where all disks have the same radius and the disk centers are separated from the points of P by a line.

All the above results are for the weighted case. The unweighted disk coverage case was also particularly studied before. Liu and Wang [23][2] considered the line-constrained problem and gave an $O(m\log n\log m+(n+m)\log(n+m))$ time algorithm. For the line-separable unit-disk case, Ambühl et al. [1] derived an algorithm of $O(m^2n)$ time, which was used as a subroutine in their algorithm for the general coverage problem in the plane (without any constraints). An improved $O(nm+n\log n)$ time algorithm is presented in [8]. Liu and Wang's approach [23] solves this case in $O((n+m)\log(n+m))$ time.

If disks of S are half-planes, the problem becomes the half-plane coverage problem. For the weighted case, Chan and Grant [4] proposed an algorithm for the *lower-only case* where all half-planes are lower ones; their algorithm runs in $O(n^4)$ time when $m=n$. With the observation that a half-plane may be considered as a unit disk of infinite radius, the lower-only half-plane coverage problem is essentially a special case of the line-separable unit-disk coverage problem [26]. Consequently, applying the algorithm of [26] can solve the weighted lower-only case in $O(n^2\log n)$ time (when $m=n$) and applying the algorithm of [23] can solve the unweighted lower-only case in $O(n\log n)$ time. Wang and Xue [28] derived another $O(n\log n)$ time algorithm for the unweighted lower-only case with a different approach and also proved an $\Omega(n\log n)$ lower bound under the algebraic decision tree model by a reduction from the set equality problem [2] (note that this leads to the same lower bound for the line-separable unit-disk coverage problem). For the general case where both upper and lower half-planes are present, Har-Peled and Lee [17] solved the weighted problem in $O(n^5)$ time. Pedersen and Wang [26] showed that the problem can be reduced to $O(n^2)$ instances of the lower-only case problem. Consequently, applying the algorithms of [26] and [23] can solve the weighted and unweighted cases in $O(n^4\log n)$ and $O(n^3\log n)$ time, respectively. Wang and Xue [28] gave a more efficient algorithm of $O(n^{4/3}\log^{5/3}n\log^{O(1)}\log n)$ time for the unweighted case.

[1] Note that [12] provides a method to reduce certain coverage problems to instances of the hitting set problem; however, the reduction algorithm, which takes more than $O(n^5)$ time, is not efficient.

[2] See the arXiv version of [23], which improves the result in the original conference paper. The algorithms follow the same idea, but the arXiv version provides more efficient implementations.

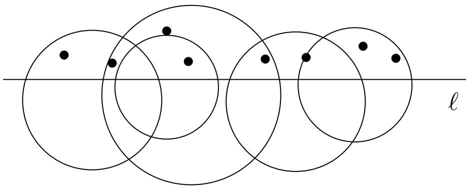


Figure 1 Illustrating the line-separable single-intersection case: Centers of all disks are below ℓ.

1.2 Our result

Instead of solving the line-constrained problem directly, we tackle a more general problem in which the points of P and the centers of the disks of S are separated by a line ℓ such that the boundaries of every two disks intersect at most once on the side of ℓ containing P (see Fig. 1). We refer to it as the *line-separable single-intersection* hitting set problem (we will explain it later why this problem is more general than the line-constrained problem). We present an algorithm of $O(m \log^2 n + (n+m)\log(n+m))$ time for the problem. To this end, we find that some points in P are "useless" and thus can be pruned from P. More importantly, the remaining points have certain properties so that the problem can be reduced to the 1D hitting set problem, which can then be easily solved. The algorithm itself is relatively simple and quite elegant. However, one challenge is to show its correctness, and specifically, to prove why the "useless" points are indeed useless. The proof is lengthy and fairly technical, which is one of our main contributions.

The line-constrained problem. To solve the line-constrained problem, where all disks of S are centered on a line ℓ, the problem can be reduced to the line-separable single-intersection case. Indeed, without loss of generality, we assume that ℓ is the x-axis. For each point p of P below ℓ, we replace p by its symmetric point with respect to ℓ. As such, we obtain a set of points that are all above ℓ. Since all disks are centered on ℓ, it is not difficult to see that an optimal solution using this new set of points corresponds to an optimal solution using P. Furthermore, since disks are centered on ℓ, although their radii may not be equal, the boundaries of any two disks intersect at most once above ℓ. Therefore, the problem becomes an instance of the line-separable single-intersection case. As such, applying the algorithm for line-separable single-intersection problem solves the line-constrained problem in $O(m \log^2 n + (n+m)\log(n+m))$ time. Therefore, in the rest of the paper, we will focus on solving the line-separable single-intersection problem.

The unit-disk case. As mentioned earlier, the unit-disk case problem where all disks have the same radius can be reduced to the coverage problem (and vice versa). More specifically, if we consider the set of unit disks centered at the points of P as a set of "dual disks" and consider the centers of the disks of S a set of "dual points", then the hitting set problem is equivalent to finding a smallest subset of dual disks whose union covers all dual points. Consequently, applying the line-separable unit-disk coverage algorithm in [23] solves the hitting set problem in $O((n+m)\log(n+m))$ time. Nevertheless, we show that our technique can directly solve the hitting set problem in this case in the same time complexity.

The half-plane hitting set problem. As in the coverage problem discussed above, if disks of S are half-planes, the problem becomes the half-plane hitting set problem. For the weighted case, the approach of Chan and Grant [4] solves the lower-only case in $O(n^4)$ time when

$m = n$. Again, with the observation that a half-plane may be viewed as a unit disk of infinite radius, the lower-only half-plane hitting set problem is a special case of the line-separable unit-disk hitting set problem. As such, applying the algorithm of [22] can solve the weighted lower-only case in $O(n^2 \log n)$ time (when $m = n$) and applying the unit-disk case algorithm discussed above can solve the unweighted lower-only case in $O(n \log n)$ time. For the general case where both upper and lower half-planes are present, Har-Peled and Lee [17] solved the weighted problem in $O(n^6)$ time. Liu and Wang [22] showed that the problem (for both the weighted and unweighted cases) can be reduced to $O(n^2)$ instances of the lower-only case problem. Consequently, applying the above algorithms for the weighted and unweighted lower-only case problems can solve the weighted and unweighted general case problems in $O(n^4 \log n)$ and $O(n^3 \log n)$ time, respectively.

Lower bound. As discussed above, the $\Omega(n \log n)$ lower bound in [28] for the lower-only half-plane coverage problem leads to the $\Omega(n \log n)$ lower bound for the line-separable unit-disk coverage problem when $m = n$. As the unit-disk hitting set problem is dual to the unit-disk coverage problem, it also has $\Omega(n \log n)$ as a lower bound. Since the unit-disk hitting set problem is a special case of the line-separable single-intersection hitting set problem, $\Omega(n \log n)$ is also a lower bound of the latter problem. Similarly, since the lower-only half-plane hitting set is dual to the lower-only half-plane coverage, $\Omega(n \log n)$ is also a lower bound of the former problem.

An algorithm in the algebraic decision tree model. In the algebraic decision tree model, where the time complexity is measured only by the number of comparisons, our method, combining with a technique recently developed by Chan and Zheng [6], shows that the line-separable single-intersection problem (and thus the line-constrained problem) can be solved using $O((n+m)\log(n+m))$ comparisons, matching the above lower bound. To ensure clarity in the following discussion, unless otherwise stated, all time complexities are based on the standard real RAM model.

Outline. The rest of the paper is organized as follows. After introducing the notation in Section 2, we describe our algorithm in Section 3. The algorithm correctness is proved in Section 4. We show how to implement the algorithm efficiently in Section 5. The algebraic decision tree algorithm and the unit-disk case algorithm are also discussed in Section 5.

2 Preliminaries

This section introduces some notation and concepts that will be used throughout the paper.

As discussed above, we focus on the line-separable single-intersection case. Let P be a set of n points and S a set of m disks in the plane such that the points of P and the centers of the disks of S are separated by a line ℓ and the boundaries of every two disks intersect at most once on the side of ℓ which contains P. Note that the points of P and the disk centers may be on ℓ. Without loss of generality, we assume that ℓ is the x-axis and the points of P are above (or on) ℓ while the disk centers are below (or on) ℓ (see Fig. 1). As such, the boundaries of every two disks intersect at most once above ℓ. Our goal is to compute a smallest subset of P such that each disk of S is hit by at least one point in the subset.

Under this setting, for each disk $s \in S$, only its portion above ℓ matters for our problem. Hence, unless otherwise stated, a disk s refers only to its portion above (and on) ℓ. As such, the boundary of s consists of an *upper arc*, i.e., the boundary arc of the original disk above ℓ, and a *lower segment*, i.e., the intersection of s with ℓ. Note that s has a single leftmost (resp., rightmost) point, which is the left (resp., right) endpoint of the lower segment of s.

If P' is a subset of P that form a hitting set for S, we call P' a *feasible solution*. If P' is a feasible solution of minimum size, then P' is an *optimal solution*.

We assume that each disk of S is hit by at least one point of P since otherwise there would be no feasible solution. Our algorithm is able to check whether the assumption is met.

We make a general position assumption that no two points of A have the same x-coordinate, where A is the union of P and the set of the leftmost and rightmost points of the upper arcs of all disks. Degenerate cases can be handled by standard perturbation techniques, e.g., [14].

For any point p in the plane, we denote its x-coordinate by $x(p)$. We sort all points in P in ascending order of their x-coordinates, resulting in a sorted list $\{p_1, p_2, \cdots, p_n\}$. We use $P[i, j]$ to denote the subset $\{p_i, p_{i+1}, \cdots, p_j\}$, for any $1 \leq i \leq j \leq n$. We sort all disks in ascending order of the x-coordinates of the leftmost points of their upper arcs; let $\{s_1, s_2, \cdots, s_m\}$ be the sorted list. We use $S[i, j]$ to denote the subset $\{s_i, s_{i+1}, \cdots, s_j\}$, for $1 \leq i \leq j \leq m$. For convenience, let $P[i, j] = \emptyset$ and $S[i, j] = \emptyset$ if $i > j$. For each disk s_i, let l_i and r_i denote the leftmost and rightmost points of its upper arc, respectively.

For any disk $s \in S$, we use $S_l(s)$ (resp., $S_r(s)$) to denote the subset of disks S whose leftmost points are to the left (resp., right) of that of s, that is, if the index of s is i, then $S_l(s) = S[1, i-1]$ and $S_r(s) = S[i+1, m]$. For any disk $s' \in S_l(s)$, we also say that s' is *to the left* of s; similarly, if $s' \in S_r(s)$, then s' is *to the right* of s. For convenience, if s' is to the left of s, we use $s' \prec s$ to denote it.

For a point $p_i \in P$ and a disk $s_k \in S$, we say that p_i is *vertically above* s_k (or s_k is *vertically below* p_i) if p_i is outside s_k and $x(l_k) < x(p_i) < x(r_k)$.

The non-containment property. If a disk s_i contains another disk s_j completely, then s_i is redundant for our problem since any point hitting s_j also hits s_i. It is easy to find those redundant disks in $O(m \log m)$ time (indeed, this is a 1D problem since s_i contains s_j if and only if the lower segment of s_i contains that of s_j). Therefore, to solve our problem, we first remove such redundant disks from S and then work on the remaining disks. For simplicity, from now on we assume that no disk of S contains another. Therefore, S has the following *non-containment* property, which is critical to our algorithm.

▶ **Observation 1.** (Non-Containment Property) *For any two disks $s_i, s_j \in S$, $x(l_i) < x(l_j)$ if and only if $x(r_i) < x(r_j)$.*

3 The algorithm description

In this section, we describe our algorithm. We follow the notation defined in Section 2.

We begin with the following definition, which is critical for our algorithm.

▶ **Definition 2.** *For each disk $s_i \in S$, among all the points of P covered by s_i, define $a(i)$ as the smallest index of these points and $b(i)$ the largest index of them.*

Since each disk s_i contains at least one point of P, both $a(i)$ and $b(i)$ are well defined.

▶ **Definition 3.** *For any point $p_k \in P$, we say that p_k is* prunable *if there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) < k < b(i)$.*

We now describe our algorithm. Although the description seems simple, establishing its correctness is by no means an easy task. We devote Section 4 to the correctness proof. The implementation of the algorithm, which is also not straightforward, is presented in Section 5.

Algorithm description. The algorithm has three main steps.

1. Compute $a(i)$ and $b(i)$ for all disks $s_i \in S$. We will show in Section 5 that this can be done in $O(m \log^2 n + (n+m)\log(n+m))$ time.
2. Find all prunable points; let Q be the set of all prunable points. We will show in Section 5 that Q can be computed in $O((n+m)\log(n+m))$ time.
   Let $P^* = P \setminus Q$. We will prove in Section 4 that P^* contains an optimal solution to the hitting problem on P and S. This means that it suffices to work on P^* and S.
3. Reduce the hitting set problem on P^* and S to a 1D hitting set problem, as follows. For each point of P^*, we project it perpendicularly onto ℓ. Let $\tilde{P}$ be the set of all projected points. For each disk $s_i \in S$, we create a segment on ℓ whose left endpoint has x-coordinate equal to $x(p_{a(i)})$ and whose right endpoint has x-coordinate equal to $x(p_{b(i)})$. Let $\tilde{S}$ be the set of all segments thus created.
   We solve the following 1D hitting set problem: Find a smallest subset of points of $\tilde{P}$ such that every segment of $\tilde{S}$ is hit by a point of the subset. This 1D problem can be easily solved in $O((|\tilde{S}|+|\tilde{P}|)\log(|\tilde{S}|+|\tilde{P}|))$ time [22],[3] which is $O((m+n)\log(m+n))$ since $|\tilde{P}| \leq n$ and $|\tilde{S}| = m$.
   Suppose that $\tilde{P}_{\text{opt}}$ is any optimal solution for the 1D problem. We create a subset P^*_{opt} of P^* as follows. For each point of $\tilde{P}_{\text{opt}}$, suppose that it is the projection of a point $p_i \in P^*$; then we add p_i to P^*_{opt}. We will prove in Section 4 that P^*_{opt} is an optimal solution to the hitting set problem on P^* and S.

   We summarize the result in the following theorem.

▶ **Theorem 4.** *Given a set P of n points and a set S of m disks in the plane such that the disk centers are separated from the points of P by a line and the single-intersection condition is satisfied, the hitting set problem is solvable in $O(m\log^2 n + (n+m)\log(n+m))$ time.*

4 Algorithm correctness

In this section, we prove the correctness of our algorithm. More specifically, we will argue the correctness of the second and the third main steps of the algorithm. We start with the third main step, as it is relatively straightforward. In fact, arguing the correctness of the second main step is quite challenging and is a main contribution of our paper.

Correctness of the third main step. For each disk $s_i \in S$, let s'_i refer to the segment of $\tilde{S}$ created from s_i. For each point $p_j \in P$, let p'_j refer to the point of $\tilde{P}$ which is the projection of p_j. Lemma 5 justifies the correctness of the third main step of the algorithm.

▶ **Lemma 5.** *A point $p_j \in P^*$ hits a disk $s_i \in S$ if and only if p'_j hits s'_i.*

Proof. Suppose p_j hits s_i. Then, $p_j \in s_i$. By definition, we have $a(i) \leq j \leq b(i)$. Hence, $x(p_{a(i)}) \leq x(p_j) \leq x(p_{b(i)})$, and thus p'_j hits s'_i by the definitions of p'_j and s'_i.

On the other hand, suppose that p'_j hits s'_i. Then, according to the definitions of p'_j and s'_i, $x(p_{a(i)}) \leq x(p_j) \leq x(p_{b(i)})$ holds. If $j = a(i)$ or $j = b(i)$, then p_j must hit s_i following the definitions of $a(i)$ and $b(i)$. Otherwise, we have $a(i) < j < b(i)$. Observe that p_j must be inside s_i since otherwise p_j would be a prunable point and therefore could not be in P^*. As such, p_j must hit s_i. ◀

[3] The algorithm in [22], which uses dynamic programming, is for the weighted case where each point has a weight. Our problem is simpler because it is the unweighted case. We can use a simple greedy algorithm to solve it.

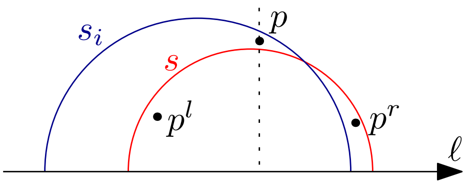


Figure 2 Illustrating the proof of Observation 6.

4.1 Correctness of the second main step

In what follows, we focus on the correctness of the second main step.

For any disk s, let $P(s)$ denote the subset of points of P inside s. For any point p, let $S(p)$ denote the subset of disks of S hit by p. For any subset $P' \subseteq P$, by slightly abusing notation, let $S(P')$ denote the subset of disks of S hit by at least one point of P', i.e., $S(P') = \bigcup_{p \in P'} S(p)$.

The following observation follows directly from the definition of prunable points.

▶ **Observation 6.** *Suppose a point p is a prunable point in P. Then, there is a disk $s \in S$ vertically below p such that the following are true.*

1. *$P(s)$ has both a point left of p and a point right of p.*
2. *For any two points $p^l, p^r \in P(s)$ with one left of p and the other right of p, we have $S(p) \subseteq S(p^l) \cup S(p^r)$.*

Proof. The first statement directly follows the definition of prunable points. For the second statement, without loss of generality, assume that p^l is left of p while p^r is right of p (see Fig. 2). Consider any disk $s_i \in S(p)$. By definition, $p \in s_i$. As $p \notin s$, $s_i \neq s$. Hence, s_i is either in $S_l(s)$ or in $S_r(s)$. If $s_i \in S_l(s)$, then due to the non-containment property, s_i must contain the area of s to the left of p and therefore must contain p^l, which implies $s_i \in S(p^l)$. Similarly, if $s_i \in S_r(s)$, then s_i must be in $S(p^r)$. ◀

The following lemma establishes the correctness of the second main step of the algorithm.

▶ **Lemma 7.** *P^* contains an optimal solution for the hitting set problem on S and P.*

Proof. Let P_{opt} be an optimal solution for S and P. Let Q be the set of all prunable points. Recall that $P^* = P \setminus Q$. If $P_{\text{opt}} \cap Q = \emptyset$, then $P_{\text{opt}} \subseteq P^*$ and thus the lemma is vacuously true. In what follows, we assume that $|P_{\text{opt}} \cap Q| \geq 1$.

Pick an arbitrary point from $P_{\text{opt}} \cap Q$, denoted by $\hat{p}_1$. Below, we give a process that can find a point p^* from P^* to replace $\hat{p}_1$ in P_{opt} such that the new set $P^1_{\text{opt}} = \{p^*\} \cup P_{\text{opt}} \setminus \{\hat{p}_1\}$ is a feasible solution, implying that P^1_{opt} is still an optimal solution since $|P^1_{\text{opt}}| = |P_{\text{opt}}|$. As $p^* \in P^*$, we have $|P^1_{\text{opt}} \cap Q| = |P_{\text{opt}} \cap Q| - 1$. Therefore, if $P^1_{\text{opt}} \cap Q$ is still nonempty, then we can repeat the process for other points in $P^1_{\text{opt}} \cap Q$ until we obtain an optimal solution P^*_{opt} with $P^*_{\text{opt}} \cap Q = \emptyset$, which will prove the lemma. The process involves induction. To help the reader understand it better, we first provide the details for the first two iterations of the process (we will introduce some notation that appears unnecessary for the first two iterations, but these will be needed for explaining the inductive hypothesis later).

The first iteration. Let $P'_{\text{opt}} = P_{\text{opt}} \setminus \{\hat{p}_1\}$. Since $\hat{p}_1 \in Q$, by Observation 6, S has a disk $\hat{s}_1$ vertically below $\hat{p}_1$ such that $P(\hat{s}_1)$ contains both a point left of $\hat{p}_1$, denoted by $\hat{p}^l_1$, and a point right of $\hat{p}_1$, denoted by $\hat{p}^r_1$. Furthermore, $S(\hat{p}_1) \subseteq S(\hat{p}^l_1) \cup S(\hat{p}^r_1)$. Since $\hat{p}_1 \notin \hat{s}_1$ and $P_{\text{opt}} = P'_{\text{opt}} \cup \{\hat{p}_1\}$ forms a hitting set of P, P'_{opt} must have a point p that hits $\hat{s}_1$. Clearly, p

is left or right of $\hat{p}_1$. Without loss of generality, we assume that p is right of $\hat{p}_1$. Since $\hat{p}_1^r$ refers to an arbitrary point to the right of $\hat{p}_1$ that hits $\hat{s}_1$ and p is also a point to right of $\hat{p}_1$ that hits $\hat{s}_1$, for notational convenience, we let $\hat{p}_1^r$ refer to p. As such, $\hat{p}_1^r$ is in P'_{opt}.

Consider the point $\hat{p}_1^l$. Since $S(\hat{p}_1) \subseteq S(\hat{p}_1^l) \cup S(\hat{p}_1^r)$ and $\hat{p}_1^r$ is in P'_{opt}, it is not difficult to see that $S(P_{\text{opt}}) \subseteq S(P'_{\text{opt}}) \cup S(\hat{p}_1^l)$ and thus $P'_{\text{opt}} \cup \{\hat{p}_1^l\}$ is a feasible solution. As such, if $\hat{p}_1^l \notin Q$, then we can use $\hat{p}_1^l$ as our target point p^* and our process (to find p^*) is performed. In what follows, we assume $\hat{p}_1^l \in Q$.

We let $\hat{p}_2 = \hat{p}_1^l$. Define $A_1 = \{\hat{p}_1^r\}$. According to the above discussion, $A_1 \subseteq P'_{\text{opt}}$, $S(\hat{p}_1) \subseteq S(A_1) \cup S(\hat{p}_2)$, $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is a feasible solution, $\hat{p}_1$ is vertically above $\hat{s}_1$, and $\hat{p}_2 \in \hat{s}_1$.

The second iteration. We are now entering the second iteration of our process. First, notice that $\hat{p}_2$ cannot be $\hat{p}_1$ since $\hat{p}_2 = \hat{p}_1^l$, which cannot be $\hat{p}_1$. Our goal in this iteration is to find a *candidate point* p' to replace $\hat{p}_2$ so that $P'_{\text{opt}} \cup \{p'\}$ also forms a hitting set of S. Consequently, if $p' \notin Q$, then we can use p' as our target p^*; otherwise, we need to guarantee $p' \neq \hat{p}_1$ so that our process will not enter a loop. The discussion here is more involved than in the first iteration.

Since $\hat{p}_2 \in Q$, by Observation 6, S has a disk $\hat{s}_2$ vertically below $\hat{p}_2$ such that $P(\hat{s}_2)$ contains both a point left of $\hat{p}_2$, denoted by $\hat{p}_2^l$, and a point right of $\hat{p}_2$, denoted by $\hat{p}_2^r$. Further, $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(\hat{p}_2^r)$. Depending on whether $\hat{s}_2$ is in $S(A_1)$, there are two cases.

- If $\hat{s}_2 \notin S(A_1)$, since $\hat{p}_2$ does not hit $\hat{s}_2$ and $S(\hat{p}_1) \subseteq S(A_1) \cup S(\hat{p}_2)$, we obtain $\hat{s}_2 \notin S(\hat{p}_1)$. Now we can basically repeat our argument from the first iteration. Since $\hat{p}_2$ does not hit $\hat{s}_2$ and $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is a feasible solution, P'_{opt} must have a point p that hits $\hat{s}_2$. Clearly, p is either left or right of $\hat{p}_2$.
  We first assume that p is right of $\hat{p}_2$. Since $\hat{p}_2^r$ refers to an arbitrary point to the right of $\hat{p}_2$ that hits $\hat{s}_2$ and p is also a point to right of $\hat{p}_2$ that hits $\hat{s}_2$, for notational convenience, we let $\hat{p}_2^r$ refer to p. As such, $\hat{p}_2^r$ is in P'_{opt}.
  We let $\hat{p}_2^l$ be our candidate point, which satisfies our need as discussed above for p'. Indeed, since $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is an optimal solution, $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(\hat{p}_2^r)$, and $\hat{p}_2^r \in P'_{\text{opt}}$, we obtain that $P'_{\text{opt}} \cup \{\hat{p}_2^l\}$ also forms a hitting set of S. Furthermore, since $\hat{p}_2^l$ hits $\hat{s}_2$ while $\hat{p}_1$ does not, we know that $\hat{p}_2^l \neq \hat{p}_1$. Therefore, if $\hat{p}_2^l \notin Q$, then we can use $\hat{p}_2^l$ as our target p^* and we are done with the process. Otherwise, we let $\hat{p}_3 = \hat{p}_2^l$ and then enter the third iteration. In this case, we let $A_2 = A_1 \cup \{\hat{p}_2^r\}$. According to our above discussion, $A_2 \subseteq P'_{\text{opt}}$, $S(\hat{p}_2) \subseteq S(A_2) \cup S(\hat{p}_3)$, $\{\hat{p}_3\} \cup P'_{\text{opt}}$ is a feasible solution, $\hat{p}_2$ is vertically above $\hat{s}_2$, and $\hat{p}_3 \in \hat{s}_2$.
  The above discussed the case where p is right of $\hat{p}_2$. If p is left of $\hat{p}_2$, then the analysis is symmetric.[4]
- If $\hat{s}_2 \in S(A_1)$, we let $\hat{p}_2^l$ be our candidate point. We show below that it satisfies our need as discussed above for p', i.e., $\{\hat{p}_2^l\} \cup P'_{\text{opt}}$ forms a hitting set of S and $\hat{p}_2^l \neq \hat{p}_1$.
  Indeed, since $A_1 = \{\hat{p}_1^r\}$ and $\hat{s}_2 \in S(A_1)$, $\hat{s}_2$ is hit by $\hat{p}_1^r$. Since $\hat{p}_1^r$ is to the right of $\hat{p}_1$, and $\hat{p}_2$, which is $\hat{p}_1^l$, is to the left of $\hat{p}_1$, we obtain that $\hat{p}_1^r$ is to the right of $\hat{p}_2$. Since $\hat{p}_2^l$ hits $\hat{s}_2$, $\hat{p}_2^l$ is to the left of $\hat{p}_2$, $\hat{p}_1^r$ hits $\hat{s}_2$, and $\hat{p}_1^r$ is to the right of $\hat{p}_2$, by Observation 6, $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(\hat{p}_1^r)$, i.e., $S(\hat{p}_2) \subseteq S(\hat{p}_2^l) \cup S(A_1)$. Since $P'_{\text{opt}} \cup \{\hat{p}_2\}$ is a feasible solution and $A_1 \subseteq P'_{\text{opt}}$, it follows that $\{\hat{p}_2^l\} \cup P'_{\text{opt}}$ is also a feasible solution. On the other hand, since $\hat{p}_2^l$ is to the left of $\hat{p}_2$ while $\hat{p}_2$ (which is $\hat{p}_1^l$) is to the left of $\hat{p}_1$, we know that $\hat{p}_2^l$ is to the left of $\hat{p}_1$ and thus $\hat{p}_2^l \neq \hat{p}_1$.

[4] More specifically, if $\hat{p}_2^r \notin Q$, then we can use $\hat{p}_2^r$ as our target p^* and the process is complete. Otherwise, we let $\hat{p}_3 = \hat{p}_2^r$ and enter the third iteration; in this case, we let $A_2 = A_1 \cup \{\hat{p}_2^l\}$.

As such, if $\hat{p}_2^l \notin Q$, we can use $\hat{p}_2^l$ as our target p^* and we are done with the process. Otherwise, we let $\hat{p}_3 = \hat{p}_2^l$ and continue with the third iteration. In this case, we let $A_2 = A_1$. According to our above discussion, $A_2 \subseteq P'_{\text{opt}}$, $S(\hat{p}_2) \subseteq S(A_2) \cup S(\hat{p}_3)$, $P'_{\text{opt}} \cup \{\hat{p}_3\}$ is a feasible solution, $\hat{p}_2$ is vertically above $\hat{s}_2$, and $\hat{p}_3 \in \hat{s}_2$.

This finishes the second iteration of the process.

Inductive step. In general, suppose that we are entering the i-th iteration of the process with the point $\hat{p}_i \in Q$, $i \geq 2$. We make the following inductive hypothesis for i.

1. We have points $\hat{p}_k \in Q$ for all $k = 1, 2, \ldots, i-1$ in the previous $i-1$ iterations such that $\hat{p}_i \neq \hat{p}_k$ for any $1 \leq k \leq i-1$
2. We have subsets A_k for all $k = 1, 2, \ldots, i-1$ such that $A_1 \subseteq A_2 \subseteq \cdots \subseteq A_{i-1} \subseteq P'_{\text{opt}}$, and $S(\hat{p}_k) \subseteq S(A_k) \cup S(\hat{p}_{k+1})$ holds for each $1 \leq k \leq i-1$.
3. For any $1 \leq k \leq i$, $\{\hat{p}_k\} \cup P'_{\text{opt}}$ is a feasible solution.
4. We have disks $\hat{s}_k \in S$ for $k = 1, 2, \ldots, i-1$ such that $\hat{s}_k$ is vertically below $\hat{p}_k$ and $\hat{p}_{k+1} \in \hat{s}_k$.

Our previous discussion already established the hypothesis for $i = 2$ and $i = 3$. Next, we proceed with the i-th iteration argument for any general $i \geq 4$. Our goal is to find a candidate point $\hat{p}_{i+1}$ such that $P'_{\text{opt}} \cup \{\hat{p}_{i+1}\}$ is a feasible solution and the inductive hypothesis still holds for $i+1$.

Since $\hat{p}_i \in Q$, by Observation 6, there is a disk $\hat{s}_i$ vertically below $\hat{p}_i$ such that $P(\hat{s}_i)$ has a point left of $\hat{p}_i$, denoted by $\hat{p}_i^l$, and a point right of $\hat{p}_i$, denoted by $\hat{p}_i^r$. Furthermore, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(\hat{p}_i^r)$. Depending on whether $\hat{s}_i$ is in $S(A_{i-1})$, there are two cases.

1. If $\hat{s}_i \notin S(A_{i-1})$, then since $\hat{s}_i$ does not contain $\hat{p}_i$ and $P'_{\text{opt}} \cup \{\hat{p}_i\}$ is a feasible solution, P'_{opt} must have a point p that hits $\hat{s}_i$. Clearly, p is to the left or right of $\hat{p}_i$. Without loss of generality, we assume that p is to the right of $\hat{p}_i$. Since $\hat{p}_i^r$ refers to an arbitrary point to the right of $\hat{p}_i$ that hits $\hat{s}_i$ and p is also a point to the right of $\hat{p}_i$ that hits $\hat{p}_i$, for notational convenience, we let $\hat{p}_i^r$ refer to p. As such, $\hat{p}_i^r$ is in P'_{opt}.
   We let $\hat{p}_{i+1}$ be $\hat{p}_i^l$ and define $A_i = A_{i-1} \cup \{\hat{p}_i^r\}$. In the following, we argue that the inductive hypothesis holds.
   - First of all, by definition, $\hat{p}_i$ is vertically above $\hat{s}_i$ and $\hat{p}_{i+1} \in \hat{s}_i$. Hence, the fourth statement of the hypothesis holds.
   - Since $\{\hat{p}_i\} \cup P'_{\text{opt}}$ is a feasible solution, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(\hat{p}_i^r)$, $\hat{p}_i^r \in P'_{\text{opt}}$, and $\hat{p}_{i+1} = \hat{p}_i^l$, we obtain that $\{\hat{p}_{i+1}\} \cup P'_{\text{opt}}$ is a feasible solution. This proves the third statement of the hypothesis.
   - Since $A_i = A_{i-1} \cup \{\hat{p}_i^r\}$, $A_{i-1} \subseteq P'_{\text{opt}}$ by inductive hypothesis, and $\hat{p}_i^r \in P'_{\text{opt}}$, we obtain $A_i \subseteq P'_{\text{opt}}$. Furthermore, since $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(\hat{p}_i^r)$, $\hat{p}_i^r \in A_i$, and $\hat{p}_{i+1} = \hat{p}_i^l$, we have $S(\hat{p}_i) \subseteq S(A_i) \cup S(\hat{p}_{i+1})$. This proves the second statement of the hypothesis.
   - For any point $\hat{p}_k$ with $1 \leq k \leq i-1$, to prove the first statement of the hypothesis, we need to show that $\hat{p}_k \neq \hat{p}_{i+1}$. To this end, since $\hat{s}_i$ is hit by $\hat{p}_{i+1}$, it suffices to show that $\hat{s}_i$ is not hit by $\hat{p}_k$. Indeed, by the inductive hypothesis, $S(\hat{p}_k) \subseteq S(A_k) \cup S(\hat{p}_{k+1})$ and $S(\hat{p}_{k+1}) \subseteq S(A_{k+1}) \cup S(\hat{p}_{k+2})$. Hence, $S(\hat{p}_k) \subseteq S(A_k) \cup S(A_{k+1}) \cup S(\hat{p}_{k+2})$. As $S(A_k) \subseteq S(A_{k+1})$, we obtain $S(\hat{p}_k) \subseteq S(A_{k+1}) \cup S(\hat{p}_{k+2})$. Following the same argument, we can derive $S(\hat{p}_k) \subseteq S(A_{i-1}) \cup S(\hat{p}_i)$. Now that $\hat{s}_i \notin S(A_{i-1})$ and $\hat{s}_i$ is not hit by $\hat{p}_i$, we obtain that $\hat{s}_i$ is not hit by $\hat{p}_k$.

2. If $\hat{s}_i \in S(A_{i-1})$, then $\hat{s}_i$ is hit by a point of A_{i-1}, say p. As $\hat{p}_i \notin \hat{s}_i$, p is left or right of $\hat{p}_i$. Without loss of generality, we assume that p is to the right of $\hat{p}_i$.
   We let $\hat{p}_{i+1}$ be $\hat{p}_i^l$ and define $A_i = A_{i-1}$. We show in the following that the inductive hypothesis holds.
   - By definition, $\hat{p}_i$ is vertically above $\hat{s}_i$ and $\hat{p}_{i+1} \in \hat{s}_i$. Hence, the fourth statement of the hypothesis holds.
   - Since $\hat{s}_i$ is hit by both p and $\hat{p}_i^l$, $\hat{p}_i^l$ is to the left of $\hat{p}_i$, and p is to the right of $\hat{p}_i$, by Observation 6, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(p)$. Further, since $\{\hat{p}_i\} \cup P'_{\text{opt}}$ is a feasible solution, $p \in A_{i-1} \subseteq P'_{\text{opt}}$, and $\hat{p}_{i+1} = \hat{p}_i^l$, we obtain that $\{\hat{p}_{i+1}\} \cup P'_{\text{opt}}$ is also a feasible solution. This proves the third statement of the hypothesis.
   - Since $A_{i-1} \subseteq P'_{\text{opt}}$ by the inductive hypothesis and $A_i = A_{i-1}$, we have $A_i \subseteq P'_{\text{opt}}$. As discussed above, $S(\hat{p}_i) \subseteq S(\hat{p}_i^l) \cup S(p)$. Since $p \in A_{i-1} = A_i$ and $\hat{p}_{i+1} = \hat{p}_i^l$, we obtain $S(\hat{p}_i) \subseteq S(A_i) \cup S(\hat{p}_{i+1})$. This proves the second statement of the hypothesis.
   - For any point $\hat{p}_k$ with $1 \leq k \leq i-1$, to prove the first statement of the hypothesis, we need to show that $\hat{p}_k \neq \hat{p}_{i+1}$. Depending on whether $x(\hat{p}_i) < x(\hat{p}_k)$, there are two cases (note that $\hat{p}_i \neq \hat{p}_k$ by our hypothesis and thus $x(\hat{p}_i) \neq x(\hat{p}_k)$ due to our general position assumption).
     - If $x(\hat{p}_i) < x(\hat{p}_k)$, then since $\hat{p}_{i+1} = \hat{p}_i^l$, we have $x(\hat{p}_{i+1}) < x(\hat{p}_i) < x(\hat{p}_k)$. Hence, $\hat{p}_k \neq \hat{p}_{i+1}$.
     - If $x(\hat{p}_k) < x(\hat{p}_i)$, then we can prove $\hat{p}_k \notin \hat{s}_i$. This implies that $\hat{p}_k \neq \hat{p}_{i+1}$ as $\hat{p}_{i+1} \in \hat{s}_i$. The proof of $\hat{p}_k \notin \hat{s}_i$, which is quite technical and lengthy, is omitted due to the space limit.

     This proves the first statement of the hypothesis.

   This proves that the inductive hypothesis still holds for $i+1$.

According to the inductive hypothesis, each iteration of the process finds a new candidate point $\hat{p}_i$ such that $P'_{\text{opt}} \cup \{\hat{p}_i\}$ is a feasible solution. If $\hat{p}_i \notin Q$, then we can use $\hat{p}_i$ as our target point p^* and we are done with the process. Otherwise, we continue with the next iteration. Since each iteration finds a new candidate point (that was never used before) and $|Q|$ is finite, eventually we will find a candidate point $\hat{p}_i$ that is not in Q.

This completes the proof of the lemma. ◀

5 Algorithm implementation

In this section, we present the implementation of our algorithm. In particular, we describe how to implement the first two steps of the algorithm: (1) Compute $a(i)$ and $b(i)$ for all disks $s_i \in S$; (2) find the subset Q of all prunable points from P.

The following lemma gives the implementation for the first step of the algorithm.

▶ **Lemma 8.** *Computing $a(i)$ and $b(i)$ for all disks $s_i \in S$ can be done in $O(m \log^2 n + (m+n)\log(m+n))$ time.*

Proof. We only discuss how to compute $a(i)$ since computing $b(i)$ can be done analogously.

Recall that points of P are indexed in ascending order of their x-coordinates as $p_1, \ldots, p_n$. Let T be a complete binary search tree whose leaves from left to right correspond to points of P in their index order. Since $n = |P|$, the height of T is $O(\log n)$. For each node $v \in T$, let P_v denote the subset of points of P in the leaves of the subtree rooted at v. Our algorithm is based on the following observation: a disk $s_i \in S$ contains a point of P_v if and only if s_i contains the closest point of P_v to c_i, where c_i is the center of s_i. In light of this observation, we construct the Voronoi diagram for P_v, denoted by VD_v, and build a point location data structure on VD_v so that each point location query can be answered in $O(\log n)$ time [13, 20]. For time analysis, after VD_v is computed, building the point location data structure on VD_v takes $O(|P_v|)$ time [13, 20]. To compute VD_v, if we do so from scratch, then it takes $O(|P_v| \log |P_v|)$ time. However, using Kirkpatrick's algorithm [19], we can compute VD_v in $O(|P_v|)$ time by merging the Voronoi diagrams VD_u and VD_w for the two children u and w of v, since $P_v = P_u \cup P_w$. As such, if we construct the Voronoi diagrams for all nodes of T in a bottom-up manner, the total time is linear in $\sum_{v\in T} |P_v|$, which is $O(n \log n)$.

For each disk $s_i \in S$, we can compute $a(i)$ using T, as follows. Starting from the root of T, for each node v, we do the following. Let u and w be the left and right children of v, respectively. First, we determine whether s_i contains a point of P_u. To this end, using a point location query on VD_u, we find the point p of P_v closest to c_i. As discussed above, s_i contains a point of P_u if and only if $p \in s_i$. If $p \in s_i$, then we proceed with $v = u$; otherwise, we proceed with $v = w$. In this way, $a(i)$ can be computed after searching a root-to-leaf path of T, which has $O(\log n)$ nodes as the height of T is $O(\log n)$. Because we spend $O(\log n)$ time on each node, the total time to compute $a(i)$ is $O(\log^2 n)$. The time for computing $a(i)$ for all disks $s_i \in S$ is thus $O(m \log^2 n)$.

In summary, the overall time to compute $a(i)$ for all disks $s_i \in S$ is bounded by $O(m \log^2 n + (n + m) \log(n + m))$. ◀

With $a(i)$ and $b(i)$ computed in Lemma 8, Lemma 10 finds all prunable points of P. The algorithm of Lemma 10 relies on the following observation.

▶ **Observation 9.** *For any point $p_k \in P$, p_k is prunable if and only if there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) \le k \le b(i)$.*

Proof. If p_k is prunable, then by definition there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) < k < b(i)$.

On the other hand, suppose that there is a disk $s_i \in S$ such that $p_k \notin s_i$ and $a(i) \le k \le b(i)$. By the definition of a_i, s_i contains the point $p_{a(i)}$. Since $p_k \notin s_i$, we obtain $k \neq a(i)$. By a similar argument, we have $k \neq b(i)$. As such, since $a(i) \le k \le b(i)$, we can derive $a(i) < k < b(i)$. Therefore, p_k is prunable. ◀

▶ **Lemma 10.** *All prunable points of P can be found in $O((n + m) \log(n + m))$ time.*

Proof. Recall that ℓ denotes the x-axis. We define T as the standard segment tree [11, Section 10.3] on the n points of ℓ whose x-coordinates are equal to $1, 2, \ldots, n$, respectively. The height of T is $O(\log n)$. For each disk $s_i \in S$, let I_i denote the interval $[a(i), b(i)]$ of ℓ. Following the definition of the standard segment tree [11, Section 10.3], we store I_i in $O(\log n)$ nodes of T. For each node $v \in T$, let S_v denote the subset of disks s_i of S whose interval I_i is stored at v. As such, $\sum_{v\in T} |S_v| = O(m \log n)$.

Consider a point $p_k \in P$. The tree T has a leaf corresponding to a point of ℓ whose x-coordinate is equal to k, called *leaf-k*. Let π_k denote the path of T from the root to leaf-k. Following the definition of the segment tree, we have the following observation: $\bigcup_{v\in\pi_k} S_v = \{s_i \mid s_i \in S, a(i) \le k \le b(i)\}$. By Observation 9, to determine whether p_k is prunable, it suffices to determine whether there is a node $v \in \pi_k$ such that S_v has a disk s_i that does not contain p_k. Recall that all points of P are above ℓ while the centers of all disks of S are below ℓ. Let C_v denote the common intersection of all disks of S_v in the halfplane above ℓ. Observe that S_v has a disk s_i that does not contain p_k if and only if p_k is outside C_v. Based on this observation, for each node $v \in T$, we compute C_v and store it at v. Due to the single-intersection property that the upper arcs of every two disks of S intersect at most once, C_v has $O(|S_v|)$ vertices; in addition, by adapting Graham's scan, C_v can be computed in $O(|S_v|)$ time if the centers of all the disks of S_v are sorted by x-coordinate (due to the non-containment property, this is also the order of the disks sorted by the left or right endpoints of their upper arcs). Assuming that the sorted lists of S_v as above are available for all nodes $v \in T$, the total time for constructing C_v for all nodes $v \in T$ is linear in $\sum_{v \in T} |S_v|$, which is $O(m \log n)$. We show that the sorted lists of S_v for all nodes $v \in T$ can be computed in $O(m \log m + m \log n)$ time, as follows. At the start of the algorithm, we sort all disks of S by the x-coordinates of their centers in $O(m \log m)$ time. Then, for each disk s_i of S following this sorted order, we find the nodes v of T where the interval I_i should be stored, and add s_i to S_v, which can be done in $O(\log n)$ time [11, Section 7.4]. In this way, after all disks of S are processed as above, S_v for every node $v \in T$ is automatically sorted. As such, all processing work on T together takes $O((m+n)\log(m+n))$ time.

For each point $p_k \in P$, to determine whether p_k is prunable, following the above discussion, we determine whether p_k is outside C_v for each node $v \in \pi_k$. Deciding whether p_k is outside C_v can be done in $O(\log m)$ time. Indeed, since the centers of all disks are below ℓ, the boundary of C_v consists of a segment on ℓ bounding C_v from below and an x-monotone curve bounding C_v from above. The projections of the vertices of C_v onto ℓ partition ℓ into a set $\mathcal{I}_v$ of $O(|S_v|)$ intervals. If we know the interval of $\mathcal{I}_v$ that contains $x(p_k)$, the x-coordinate of p_k, then whether p_k is outside C_v can be determined in $O(1)$ time. Clearly, we can find the interval of $\mathcal{I}_v$ that contains $x(p_k)$ in $O(\log m)$ time by binary search. In this way, whether p_k is prunable can be determined in $O(\log m \log n)$ time as π_k has $O(\log n)$ nodes. The time can be improved to $O(\log m + \log n)$ using fractional cascading [7], as follows.

We construct a fractional cascading data structure on the intervals of $\mathcal{I}_v$ of all nodes $v \in T$, which takes $O(m \log n)$ time [7] since the total number of such intervals is $O(m \log n)$. With the fractional cascading data structure, for each point $p_k \in P$, we only need to do binary search on the set of the intervals stored at the root of T to find the interval containing $x(p_k)$, which takes $O(\log(m \log n))$ time. Subsequently, following the path π_k in a top-down manner, the interval of $\mathcal{I}_v$ containing $x(p_k)$ for each node $v \in \pi_k$ can be determined in $O(1)$ time [7]. As such, whether p_k is prunable can be determined in $O(\log n + \log m)$ time. Hence, the total time for checking all the points $p_k \in P$ is $O(n \log(m+n))$.

In summary, the time complexity of the overall algorithm for finding all prunable disks of S is bounded by $O((n+m)\log(n+m))$. ◀

With Lemmas 8 and 10, Theorem 4 is proved.

An algebraic decision tree algorithm. In the algebraic decision tree model, where only comparisons are counted towards time complexities, the problem can be solved in $O((n+m)\log(n+m))$ time, i.e., using $O((n+m)\log(n+m))$ comparisons. To this end, observe that the entire algorithm, with the exception of Lemma 8, takes $O((n+m)\log(n+m))$ time. As such, we only need to show that Lemma 8 can be solved using $O((n+m)\log(n+m))$ comparisons. For this, notice that the factor $O(m \log^2 n)$ in the algorithm of Lemma 8 is caused by the point location queries on the Voronoi diagrams VD_v. The number of point location queries is $O(m \log n)$. The total combinatorial complexity of the Voronoi diagrams VD_v of all nodes $v \in T$ is $O(n \log n)$. To answer these point location queries, we employ a method recently introduced by Chan and Zheng [6]. In particular, by applying [6, Theorem 7.2], all point location queries can be solved using $O((n+m)\log(n+m))$ comparisons (specifically, following the notation in [6, Theorem 7.2], we have $t = O(n)$, $L = O(n \log n)$, $M = O(m \log n)$, and $N = O(n+m)$ in our problem; according to the theorem, all point location queries can be answered using $O(L + M + N \log N)$ comparisons, which is $O((n+m)\log(n+m))$).

The unit-disk case. If all disks of S have the same radius (and the points of P are separated from the centers of all disks of S by the x-axis ℓ), then as discussed in Section 1 this problem can be solved in $O((n+m)\log(n+m))$ time by reducing it to a line-separable unit-disk coverage problem and then applying the algorithm in [23]. Here, we show that our approach can provide an alternative algorithm with the same runtime.

We apply the same algorithm as above. Observe that the algorithm, except for Lemma 8, runs in $O((n+m)\log(n+m))$ time. Hence, it suffices to show that Lemma 8 can be implemented in $O((n+m)\log(n+m))$ time for the unit-disk case, which is done in the following lemma.

▶ **Lemma 11.** *If all disks of S have the same radius, then $a(i)$ and $b(i)$ for all disks $s_i \in S$ can be computed in $O((n+m)\log(n+m))$ time.*

Proof. We only discuss how to compute $a(i)$ since the algorithm for $b(i)$ is similar. We modify the algorithm in the proof of Lemma 8 and follow the notation there.

For any disk $s_i \in S$, to compute $a(i)$, recall that a key subproblem is to determine whether s_i contains a point of P_v for a node $v \in T$. To solve the subproblem, the algorithm of Lemma 8 uses Voronoi diagrams. Here, we use a different approach by exploring the property that all disks of S have the same radius, say r. For any point $p \in P$, let D_p denote the disk of radius r and centered at p. Define $\mathcal{D}_v = \{D_p \mid p \in P_v\}$. For each point $p \in P$, since p is above the axis ℓ, the portion of the boundary of D_p below ℓ is an arc on the lower half circle of the boundary of D_p, and we call it the *lower arc* of D_p. Let $\mathcal{L}_v$ denote the lower envelope of ℓ and the lower arcs of all disks of $\mathcal{D}_v$. Our method is based on the observation that s_i contains a point of P_v if and only if c_i is above $\mathcal{L}_v$, where c_i is the center of s_i.

In light of the above discussion, we construct $\mathcal{L}_v$ for every node $v \in T$. Since all disks of $\mathcal{D}_v$ have the same radius and all their centers are above ℓ, the lower arcs of every two disks of $\mathcal{D}_v$ intersect at most once. Due to this single-intersection property, $\mathcal{L}_v$ has at most $O(|P_v|)$ vertices. To see this, we can view the lower envelope of each lower arc of $\mathcal{D}_v$ and ℓ as an extended arc. Every two such extended arcs still cross each other at most once and therefore their lower envelope has $O(|P_v|)$ vertices following the standard Davenport-Schinzel sequence argument [27] (see also [5, Lemma 3] for a similar problem). Notice that $\mathcal{L}_v$ is exactly the lower envelope of these extended arcs and thus $\mathcal{L}_v$ has $O(|P_v|)$ vertices. Note also that $\mathcal{L}_v$ is x-monotone. In addition, given $\mathcal{L}_u$ and $\mathcal{L}_w$, where u and w are the two children of v, $\mathcal{L}_v$ can be computed in $O(|P_v|)$ time by a straightforward line sweeping algorithm. As such, if we compute $\mathcal{L}_v$ for all nodes $v \in T$ in a bottom-up manner, the total time is linear in $\sum_{v\in T}|P_v|$, which is $O(n\log n)$.

For each disk $s_i \in S$, we now compute $a(i)$ using T, as follows. Starting from the root of T, for each node v, we do the following. Let u and w be the left and right children of v, respectively. We first determine whether c_i is above $\mathcal{L}_u$; since $|P_u| \leq n$, this can be done in $O(\log n)$ time by binary search. More specifically, the projections of the vertices of $\mathcal{L}_u$ onto ℓ partition ℓ into a set $\mathcal{I}_u$ of $O(P_u)$ intervals. If we know the interval of $\mathcal{I}_u$ that contains $x(c_i)$, the x-coordinate of c_i, then whether c_i is above $\mathcal{L}_u$ can be determined in $O(1)$ time. Clearly, finding the interval of $\mathcal{I}_u$ containing $x(c_i)$ can be done by binary search in $O(\log n)$ time. If c_i is above $\mathcal{L}_u$, then s_i must contain a point of P_u; in this case, we proceed with $v = u$. Otherwise, we proceed with $v = w$. In this way, $a(i)$ can be computed after searching a root-to-leaf path of T, which has $O(\log n)$ nodes as the height of T is $O(\log n)$. Because we spend $O(\log n)$ time on each node, the total time for computing $a(i)$ is $O(\log^2 n)$. As in Lemma 10, the time can be improved to $O(\log n)$ using fractional cascading [7], as follows.

We construct a fractional cascading data structure on the intervals of $\mathcal{I}_v$ of all nodes $v \in T$, which takes $O(n\log n)$ time [7] since the total number of such intervals is $O(n\log n)$. With the fractional cascading data structure, for each disk $s_i \in S$, we only need to do binary search on the set of the intervals stored at the root of T to find the interval containing $x(c_i)$, which takes $O(\log n)$ time. After that, the interval of $\mathcal{I}_u$ containing $x(c_i)$ for each node u in the algorithm as discussed above can be determined in $O(1)$ time [7]. As such, $a(i)$ can be computed in $O(\log n)$ time. Hence, computing $a(i)$ for all disks $s_i \in S$ takes $O(m \log n)$ time.

In summary, the total time to compute $a(i)$ for all disks $s_i \in S$ is bounded by $O((n + m)\log(n + m))$ time. ◀

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 *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), and the 10th International Conference on Randomization and Computation (RANDOM)*, pages 3–14, 2006. doi:10.1007/11830924_3.

2 Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86, 1983. doi:10.1145/800061.808735.

3 Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018. doi:10.1016/j.dam.2017.12.018.

4 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry: Theory and Applications*, 47:112–124, 2014. doi:10.1016/j.comgeo.2012.04.001.

5 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.24.

6 Timothy M. Chan and Da Wei Zheng. Hopcroft's problem, log-star shaving, 2D fractional cascading, and decision trees. *ACM Transactions on Algorithms*, 2023. doi:10.1145/3591357.

7 Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986. doi:10.1007/BF01840440.

8 Francisco Claude, Gautam K. Das, Reza Dorrigiv, Stephane Durocher, Robert Fraser, Alejandro López-Ortiz, Bradford G. Nickerson, and Alejandro Salinger. An improved line-separable algorithm for discrete unit disk cover. *Discrete Mathematics, Algorithms and Applications*, 2:77–88, 2010. doi:10.1142/S1793830910000486.

9 Gruia Călinescu, Ion I. Măndoiu, Peng-Jun Wan, and Alexander Z. Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *Mobile Networks and Applications*, 9:101–111, 2004. doi:10.1023/B:MONE.0000013622.63511.57.

10 Gautam K. Das, Sandip Das, and Subhas C. Nandy. Homogeneous 2-hop broadcast in 2D. *Computational Geometry: Theory and Applications*, 43:182–190, 2010. doi:10.1016/j.comgeo.2009.06.005.

11 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry — Algorithms and Applications.* Springer-Verlag, Berlin, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

12 Stephane Durocher and Robert Fraser. Duality for geometric set cover and geometric hitting set problems on pseudodisks. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015. URL: https://research.cs.queensu.ca/cccg2015/CCCG15-papers/10.pdf.

13 Herbert Edelsbrunner, Leonidas J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986. doi:10.1137/0215023.

14 Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9:66–104, 1990. doi:10.1145/77635.77639.

15 Guy Even, Dror Rawitz, and Shimon Shahar. Hitting sets when the VC-dimension is small. *Information Processing Letters*, 95:358–362, 2005. doi:10.1016/j.ipl.2005.03.010.

16 Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants.* PhD thesis, Duke University, 2011. URL: https://dukespace.lib.duke.edu/server/api/core/bitstreams/0d37dabc-42a5-4bc8-b4e9-e69b263a10ca/content.

17 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3:65–85, 2012. doi:10.20382/jocg.v3i1a4.

18 Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

19 David G. Kirkpatrick. Efficient computation of continuous skeletons. In *20th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 18–27, 1979. doi:10.1109/SFCS.1979.15.

20 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983. doi:10.1137/0212002.

21 Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 898–909, 2015. doi:10.1007/978-3-662-47672-7_73.

22 Gang Liu and Haitao Wang. Geometric hitting set for line-constrained disks. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 574–587, 2023. doi:10.1007/978-3-031-38906-1_38.

23 Gang Liu and Haitao Wang. On the line-separable unit-disk coverage and related problems. In *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC)*, pages 51:1–51:14, 2023. Full version available at arXiv:2309.03162.

24 El O. Mourad, Fohlin Helena, and Srivastav Anand. A randomised approximation algorithm for the hitting set problem. *Theoretical Computer Science*, 555:23–34, 2014. doi:10.1007/978-3-642-36065-7_11.

25 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010. doi:10.1007/s00454-010-9285-9.

26 Logan Pedersen and Haitao Wang. Algorithms for the line-constrained disk coverage and related problems. *Computational Geometry: Theory and Applications*, 105-106:101883:1–18, 2022. doi:10.1016/j.comgeo.2022.101883.

27 Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1996.

28 Haitao Wang and Jie Xue. Algorithms for halfplane coverage and related problems. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, pages 79:1–79:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.79.

Scheduling with Locality by Routing

Alison Hsiang-Hsuan Liu
Department of Information and Computing Sciences, Utrecht University, The Netherlands

Fu-Hong Liu
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Abstract

This work examines a strongly NP-hard routing problem on trees, in which multiple servers need to serve a given set of requests (on vertices), where the routes of the servers start from a common source and end at their respective terminals. Each server can travel free of cost on its source-to-terminal path but has to pay for travel on other edges. The objective is to minimize the maximum cost over all servers. As the servers may pay different costs for traveling through a common edge, balancing the loads of the servers can be difficult. We propose a polynomial-time 4-approximation algorithm that applies the parametric pruning framework but consists of two phases. The first phase of the algorithm partitions the requests into packets, and the second phase of the algorithm assigns the packets to the servers. Unlike the standard parametric pruning techniques, the challenge of our algorithm design and analysis is to harmoniously relate the quality of the partition in the first phase, the balances of the servers' loads in the second phase, and the hypothetical optimal values of the framework. For the problem in general graphs, we show that there is no algorithm better than 2-approximate unless $P = NP$. The problem is a generalization of unrelated machine scheduling and other classic scheduling problems. It also models scheduling problems where the job processing times depend on the machine serving the job and the other jobs served by that machine. This modeling provides a framework that physicalizes scheduling problems through the graph's point of view.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Makespan minimization, Approximation algorithms, Routing problems, Parametric pruning framework

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.69

Acknowledgements The authors would like to thank Sheng-Yin Chen and Chung-Shou Liao for their discussions in the early stages.

1 Introduction

We propose a generalization of the makespan minimization problem that concerns jobs with *locality* and machines with *preferences*. Consider two jobs sharing some common preprocessing subroutines. The time for serving these jobs is reduced if they are assigned to the same machine since the outcome of the preprocessing can be shared (instead of executing the preprocessing step from scratch again). Jobs are considered to be "close" to each other if they share more preprocessing. In other words, the *locality* of jobs indicates how much acceleration a schedule can achieve when these jobs are assigned to one machine.

On the other hand, the machines have *preferences* on jobs. That is, a machine may be better at serving some of the jobs. If the jobs have common preprocessing routines, the machine that has a preference for the routine can further accelerate the total processing time of the batch of jobs.

For example, consider assigning teaching tasks to faculty members in a department. The teaching tasks can be classified systematically as a rooted tree structure. For instance, an internal node representing Algorithm Design And Analysis course may have two children, Approximation Algorithms and Randomized Algorithms courses. Every faculty member has

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 69; pp. 69:1–69:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

their own research area and prefers teaching the courses that match their expertise, where each expertise is a path in the tree starting from the root. A course in an area further away from a teacher's research direction takes this teacher more time to prepare and also makes them less happy. The department's goal is to balance the workload people take to prepare for courses. That is, the objective of the assignment is to make the most unhappy colleague as happy as possible.

In a typical routing problem, there are servers that are able to move along paths on a given graph. A server may be required to stop at a terminal, which is a vertex that is associated with the server. Additionally, a server may also need to travel to some vertices to serve some requests or collect some packets located at the vertices.

To model the job localities and the machine preferences, we propose the *scheduling with locality* problem (Scheduling-with-Locality) and describe it as a routing problem on a graph with server-sensitive edge weights. Given a weighted graph, the jobs are represented by *requests* located at (some of) the vertices, and the distance between requests indicates how close the corresponding jobs are. The machines are represented by *servers* initially at a *source vertex* and aim to travel to their *terminals*. Serving a request by a server is analogous to assigning the corresponding job (of the request) to the corresponding machine (of the server). The traveling distance for a server to serve a request is analogous to the time for processing the corresponding job on the corresponding machine. A server may have some *discount* on the weights of some particular edges. That is, it pays a smaller traveling cost when traveling through these edges. The discount of servers on those edges indicates the *preference* of the corresponding machines serving particular types of jobs. The goal is to compute a schedule for all servers, which is a set of walks (where vertices/edges may repeat), each for one server. A walk corresponding to a server starts from the common source vertex and ends at the terminal of the server. To feasibly serve all requests, the walks should cover all of them. The traveling cost of a server on its walk is the total load (that is, the total processing time) of the corresponding machine. To minimize the makespan of a schedule, we want to find the walks where the maximum traveling cost (with regard to the corresponding server) is minimized. The modeling aims to provide a framework that is capable of transforming scheduling problems into a routing problem that has the potential to capture more properties of the scheduling problems from the graph's point of view.

Formal problem definition. An instance of Scheduling-with-Locality is given by an undirected graph $G = (V, E)$, a set of *servers* $\mathcal{S}$, a set of *requests* $\mathcal{R} \subseteq V$, and a *source* $\rho \in V$. Each server $s \in \mathcal{S}$ is associated with its *terminal* $\tau(s) \in V$. Each edge $e \in E$ is associated with server-sensitive weights $weight(e, s) \geq 0$ for $s \in \mathcal{S}$, where $weight(e, s)$ is the cost for server s traveling through edge e. In a feasible schedule, each server $s \in \mathcal{S}$ is assigned a walk[1] that starts from ρ and ends at $\tau(s)$, and the walks of all servers in $\mathcal{S}$ all together visit all requests $\mathcal{R}$. Let Z_s be the walk assigned to server s by schedule Z, the cost of a server s in this schedule, $cost(Z_s, s)$, is $\sum_{e \in Z_s} weight(e, s)$ by definition. The cost of a schedule Z is defined as the maximum cost of servers on their corresponding walk, $\max_{s \in \mathcal{S}}\{cost(Z_s, s)\}$. The objective is to find a schedule that minimizes the cost, i.e., the maximum traveling cost of server walks.

We consider a special case **Multi-server Routing with Free-paths (MRF) problem**, where the given graph is a weighted tree $\mathcal{T} = (V, E)$ rooted at the source vertex ρ. There is an integral weight function on edges $weight : E \to \mathbb{N}$. We call the path from ρ to $\tau(s)$ the

[1] A *walk* is defined by a sequence of vertices $W = \{v_0, v_1, v_2, \cdots, v_\ell\}$ where v_{i-1} and v_i are adjacent for any $1 \leq i \leq \ell$. It is possible that a vertex or an edge appears multiple times in a walk.

home path of server s, denoted as $P_H(s)$. A server can travel on the edges of its home path free of cost. Formally, $weight(e, s) = 0$ if $e \in P_H(s)$ and $weight(e, s) = weight(e)$ otherwise. We will generalize the edge weights from integers to real numbers later in the paper.

Scheduling-with-Locality can model many scheduling problems, such as Unrelated machine scheduling [21], Machine scheduling with setup times [22], etc. (see the full version for details). On the other hand, although the preferences of servers are limited in the MRF problem, the problem captures the locality-aware properties of scheduling problems, i.e., the cost of a server serving a set of requests depends on the locality of the requests and the server. The problem still preserves servers' preferences for serving batches of requests and accommodates complicated inter-request localities by the tree structure. On the other hand, the routing is tricky due to different edge costs for different servers. In fact, one can show that without wisely partitioning the requests, any algorithm is at least r-approximate, where r is the number of requests.

Our results

For positive results, We propose a two-phase Partition-and-Balancing (PnB) algorithm that first partitions the requests into packets properly and then assigns the packets to the servers while balancing the serving costs of servers. More specifically, we apply the classic *parametric pruning* framework that has been exploited to solve the k-center problem [23]. Intuitively, the framework keeps guessing the optimal value. Given a parameter ϑ as the guessed optimal value, if an algorithm ALG guarantees to return a solution with a cost of $\alpha \cdot \vartheta$ as long as ϑ is a correct guess of the optimal cost, then ALG is α-approximate. Afterward, the framework concentrates on finding a correct guess.

In the two-phase PnB algorithm, we use the parameter ϑ as a hypothesized value of the optimal cost. In the first phase, we partition the requests into packets according to the value of ϑ. The partition ensures that for each packet, the cost of traveling to the packet and the cost of traveling within the packet (to serve the requests) are balanced for any server. Eventually, the cost of serving each packet falls between ϑ and 2ϑ, with a bounded number of packets with serving costs less than ϑ (Lemma 3). The second phase delicately assigns the packets, consulting the value of ϑ and the topology of the server terminals. In the end, with a large enough ϑ, the algorithm guarantees that a server only reaches packets within a distance bounded by a constant factor of ϑ, and the cost of any server is at most 4ϑ (Theorem 5). To make sure that if the algorithm fails to generate an assignment with the cost at most 4ϑ, then the value ϑ must be strictly smaller than the optimal cost, the algorithm is designed such that the information of the instance (that will be used in the second phase) when packing the requests into packets in the first phase is preserved (Theorem 8). Finally, the algorithm is adjusted to solve the problem with real number edge weights via runtime trade-off.

For negative results, we provide the 2-inapproximability for the general scheduling with locality problem (Scheduling-with-Locality) to complete the study (Theorem 15).

Related work

Makespan minimization has been studied in various settings where machines have different speeds with respect to different jobs [7, 12, 13, 15, 17, 20, 21], machines with status and setup costs [1, 10, 18], machines with reconfiguration overheads [19, 25], etc. On the other hand, the scheduling of jobs with locality has raised attention recently [14, 16, 25]. That is, reusing configurations among different jobs increases warm-starts and reduces cold-start overheads.

Jobs scheduling and load balancing. The closest related problem is the jobs scheduling problem with the objective of minimizing the makespan. Lenstra et al. proposed a classic LP-based polynomial time 2-approximation algorithm [21]. In contrast, Gairing et al. provided a combinatorial 2-approximation algorithm [15]. Later, Arad et al. used a parametric pruning style technique [2]. They showed that given values L and T, either there exists no schedule of mean machine completion time L and makespan T, or a schedule of makespan at most $T + L/h < 2T$ can be found in polynomial time, where $h \in (0, 1]$ is the feasibility factor of a given instance. An important special case of job scheduling problem is the restricted assignment problem, where processing times are of the form $p_{i,j} \in \{p_j, \infty\}$. The best known result of the problem to date is a $(\frac{33}{17} + \varepsilon)$-approximation by Svensson [26].

Multiple traveling salespeople problem. Another related problem is the *multiple traveling salespeople problem* [24]. Given a set C of n cities, k salespeople, and a depot $d \in C$, the multiple TSP problem aims to find k tours that start and end at the depot such that all of the cities in C must be visited by at least one salesperson. The objective is to minimize the maximum tour length over the k tours. In Euclidean metric, Monroe and Mount [24] showed that there exists a randomized algorithm for the multiple TSP problem that runs in the expected time $O(n((\frac{1}{\varepsilon})\log(\frac{n}{\varepsilon}))^{O(1/\varepsilon)})$, where $\varepsilon > 0$ is an approximation parameter. For the multiple distinct depots case, Xu and Rodrigues proposed a $\frac{3}{2}$-approximation algorithm [27]. For more approximation algorithms for a variety of vehicle routing problems within graphs, please see [3]. Note that the MRF problem should not be confused with the *k-delivery traveling salesman problem* [4, 6], which is a routing problem of limited-capacity vehicles with pickup and delivery locations.

Note that in these previous works, the number of servers is considered to be constant, while in the MRF problem, the number of servers is part of the input since each server has its own terminal. Moreover, in the MRF problem, the servers are allowed to travel through one edge multiple times. To decrease the overall cost, an optimal solution may send multiple servers through one edge. If the metric space in the multiple TSP problem is a tree, and the servers are allowed to travel through an edge more than once, the multiple TSP problem can be seen as a special case of our problem.

Other related work. MRF is similar to the *school bus problem* with regret minimization [8], in which an algorithm is additionally able to determine the locations of terminals. Bock et al. [8] provide a 13.5-approximation algorithm for the school bus problem. For more similar problems in vehicle routing, readers may refer to the survey [9]. On the other hand, the min-max objective considered in this paper also has a significant impact on other combinatorial optimization problems in vehicle routing [5] and efficiency of allocations [11].

Paper organization. In Section 2, we propose our algorithm PnB. In Section 3, we analyze the correctness and approximation ratio of the PnB algorithm. Finally, in Section 4, we provide complexity results of the SCHEDULING-WITH-LOCALITY problem. Due to the page limit, we skip most of the proofs. The complete version with proofs and pseudocodes is attached in the appendix.

2 Partition-and-Balancing Algorithm

We propose the Partition-and-Balancing (PnB) algorithm for the MRF problem, where the input graph is a weighted tree $\mathcal{T}$ rooted at the source vertex ρ, and $weight(e, s) = 0$ if the edge $e \in P_H(s)$ (that is, e is on the home path of s, from ρ to $\tau(s)$) and $weight(e, s) = weight(e)$ otherwise. We first specify the terminologies that will be used in the algorithm and analysis.

The input tree $\mathcal{T}$ can be partitioned into two components, *skeleton* (denoted by $\mathcal{T}_s$) and *request forest* (denoted by $\mathcal{F}_{\mathcal{R}}$) according to the servers' terminals and the set of requests $\mathcal{R}$. The skeleton is the union of the home paths of all servers, and the request forest is defined by $E(\mathcal{T}) \setminus \mathcal{T}_s$. Each connected component in $\mathcal{F}_{\mathcal{R}}$ is a tree rooted at some vertex on the skeleton. We call the trees *request trees* In other words, the request forest is a set of request trees (see Figure 1). We say that a server s is *based* at a (sub)tree T if $\tau(s) \in T$. We also say that an edge e is a *detour* of server s if $e \in P_H(s') \setminus P_H(s)$, where $s' \neq s$. Conceptually, a detour of server s is an edge that s travels at a non-free cost while there is another server that can travel through it free of cost. Note that the edges in request forest are not detours of any server, as all servers have to pay for travel in request forest.

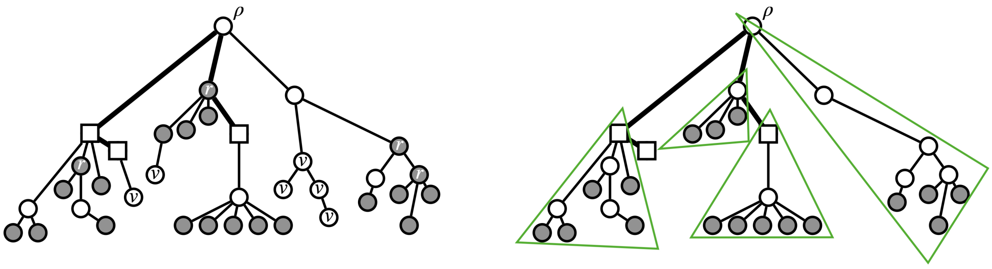
Given an edge-weighted graph, a *walk* is a sequence of vertices $W = \{v_0, v_1, v_2, \cdots, v_\ell\}$ such that v_{i-1} and v_i are adjacent for any $1 \leq i \leq \ell$. The walk W is called a *closed walk* if $v_0 = v_\ell$. The *cost* of walk W is the total weight of the edges and formally $\sum_{i=1}^{\ell} weight((v_{i-1}, v_i))$. Note that in a walk, the vertices or edges may be repeated, and a repeated edge contributes its weight multiple times in the cost of the walk. Given a weighted rooted tree $T = (V, E)$ with weight function $weight : E \to \mathbb{N}$, we denote the subtree rooted on vertex $v \in V$ by T_v. In T, the unique path between a pair of vertices u and v is called (u, v)*-path* and denoted by $P_{u,v}$. We further denote the distance between two vertices u and v by $dist(u, v)$, which is the sum of weights of the edges on $P_{u,v}$. For any connected component $C \subseteq E$, which can be a walk, a path, or even a subtree, we denote $weight(C)$ as the total weight of the edges in C.

For convenience, we define two operations, $\texttt{Walk}(P)$ and $\texttt{Merge}(C, W)$. The $\texttt{Walk}(P)$ operation makes the path P into a walk by going back and forth on the path. Formally, given that $P = [v_1, v_2, \cdots, v_\ell]$, $\texttt{Walk}(P)$ returns a walk $W = \{v_1, v_2, \cdots, v_{\ell-1}, v_\ell, v_{\ell-1}, v_{\ell-2}, \cdots, v_2, v_1\}$. The function $\texttt{Merge}(C, W)$ is to merge the closed walk W to the component C, which can be a path or a walk, by concatenating the two components properly. Note that the $\texttt{Merge}(C, W)$ function is feasible only when W is a closed walk, and there is at least a vertex that is in W and in C at the same time. More formally, assume that $C = \{v_1, v_2, \cdots, v_\ell\}$ is a walk and $W = \{u_1, u_2, \cdots, u_k, u_1\}$, where $v_x = u_y$ for some x and y, $\texttt{Merge}(C, W)$ returns a walk $\{v_1, v_2, \cdots, v_x = u_y, u_{y+1}, \cdots, u_k, u_1, \cdots, u_y = v_x, v_{x+1}, \cdots, v_\ell\}$. If there are multiple vertices that are in both C and W, we break ties arbitrarily.

Instance transformation. We first transform any input tree into a special form, where all requests are at the leaves, and the PnB algorithm works on the transformed instance. Formally, given the input tree $\mathcal{T}$, we first remove all vertices v such that there are no terminals or requests in T_v. Then, we remove the requests on any internal node of the resulting tree. (See Figure 1.)

One can prove by contradiction that in an optimal schedule, no server travels to any of the removed vertices. Moreover, since we only remove requests at the vertices with descendants that also contain requests or terminals, an optimal solution on the original input tree is a feasible schedule of the modified input tree. By a similar reasoning, we can argue that PnB($\mathcal{T}''$) is a feasible schedule of $\mathcal{T}$, where $\mathcal{T}''$ is the modified input tree. Therefore, we have the following lemma, and it is legal to restrict our discussion to the instance where all requests are at the leaves:

▶ **Lemma 1.** *If the PnB algorithm is α-approximate on the modified input $\mathcal{T}''$, PnB algorithm is α-approximate on input $\mathcal{T}$.*



(a) Original instance.

(b) Instance after transformation.

Figure 1 Instance transformation. The squares are terminals, and the gray circles are requests. The thick edges indicate the skeleton $\mathcal{T}_s$. The vertices labeled by v and requests at the vertices labeled by r in 1a are removed in 1b. In 1b, each of the green triangles indicates a request tree.

2.1 PnB algorithm

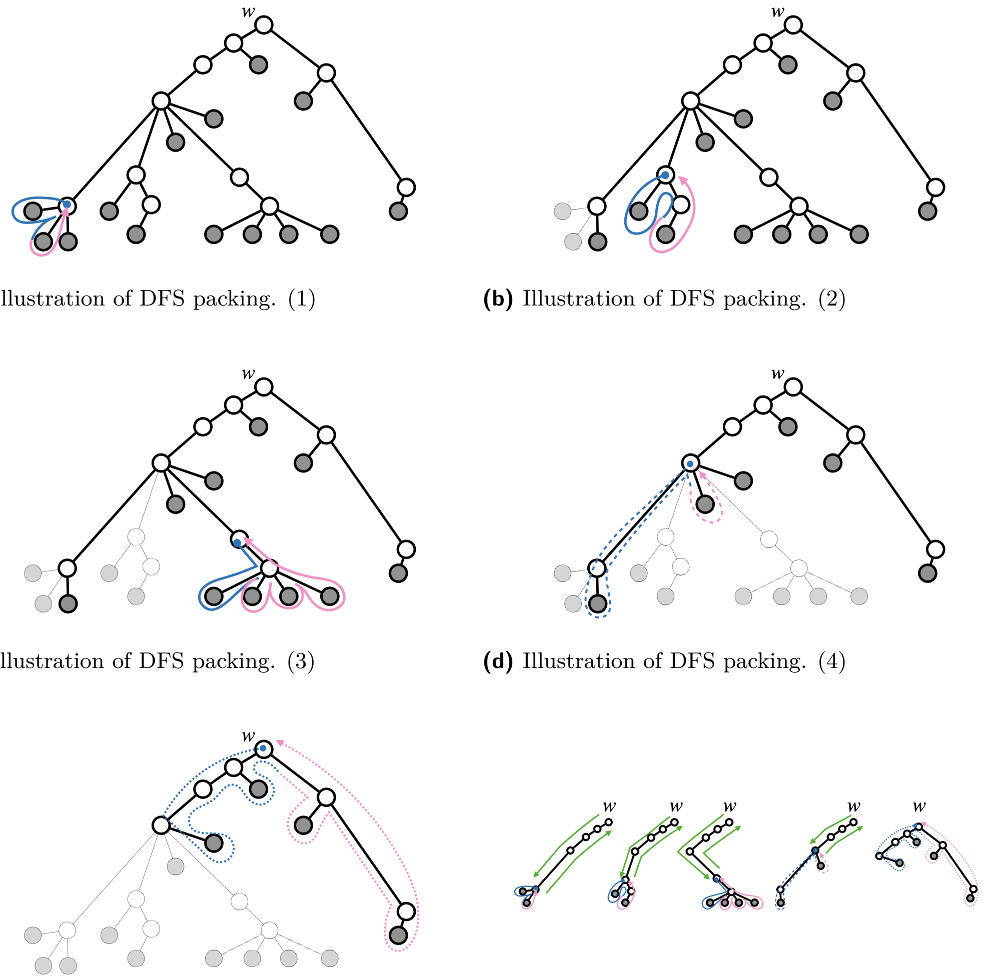
The PnB algorithm follows the parametric pruning framework and has two phases: `Partition` and `Assignment`. Taking parameter $\vartheta = 1, 2, 3, \cdots$, the algorithm PnB$(\mathcal{T}, \vartheta)$ first calls `Partition`$(\mathcal{T}, \vartheta)$. If `Partition`$(\mathcal{T}, \vartheta)$ does not fail, it returns an updated tree $\mathcal{T}'$ that only consists of the skeleton while each request tree T_w rooted at w is packed into packets stored at the vertex w. Next, the algorithm calls `Assignment`$(\mathcal{T}', \vartheta)$. If `Assignment`$(\mathcal{T}', \vartheta)$ does not fail, it returns a set of $|\mathcal{S}|$ walks, which is an assignment of the packets to the servers. Whenever a phase fails, we terminate the process of PnB$(\mathcal{T}, \vartheta)$ and move on to the next ϑ value.

2.1.1 `Partition`: Packing requests (Phase 1)

Given a value of ϑ, `Partition` deals with the request trees one by one and returns `fail` if there is a request tree with a depth (that is, the length of the longest path from w) larger than $\vartheta/2$. In a request tree T_w, the procedure keeps formulating packets of requests in a bottom-up manner, removing the walk that travels through the packed requests from the request tree, and finally, storing the formulated packets at the root of the request tree. After the `Partition` procedure, if it does not return `fail`, the updated tree $\mathcal{T}'$ has a special form, where the vertices in the request trees except the root are all deleted, and the vertices that were the roots of request trees now store packets. Moreover, each leaf is a terminal of some server in the updated tree $\mathcal{T}'$.

Checking the vertices in the request tree from bottom-top. For each request tree T_w rooted at w, the `Partition` procedure checks the vertices $v_1, v_2, \cdots, v_k$ in T_w in post order. If the subtree T_{v_i} has a weight of less than $\frac{\vartheta - dist(w, v_i)}{2}$, we finish checking v_i and move on to the next vertex v_{i+1} in T_w in the post order. Otherwise, T_{v_i} is "heavy" enough, and we proceed to the next step for partitioning the requests in T_{v_i}.

Partition the requests in a heavy-enough subtree T_{v_i} of request tree T_w. We travel T_{v_i} in the depth-first-search traversal, which is a walk $W_{v_i} = \{v_i, u_1, u_2, \cdots, u_{k_1}, v_i, u_{k_1+1}, u_{k_1+2}, \cdots, u_{k_2}, v_i, \cdots, v_i\}$. Let $\beta_{v_i} = \vartheta - 2 \cdot dist(w, v_i)$. We partition W_{v_i} into sub-walks, $B_1, B_2, \cdots, B_m$, such that each B_j starts and ends at v_i and is the walk with smallest total weight such that $weight(B_j) \geq \beta_{v_i}/2$. Then, each sub-walk B_j is merged to the walk `Walk`(P_{w,v_i}) and forms a packet. The involved vertices in B_j are then trimmed



(a) Illustration of DFS packing. (1)

(b) Illustration of DFS packing. (2)

(c) Illustration of DFS packing. (3)

(d) Illustration of DFS packing. (4)

(e) Illustration of DFS packing. (5)

(f) The five corresponding packets stored at vertex w.

Figure 2 An illustration of `Partition`(ϑ) on T_w. In 2a–2e, the blue trajectories show the walks paid by the "budget" $\beta_{\cdot} = \vartheta - 2 \cdot dist(w, \cdot)$ (where $\cdot$ is the vertex the corresponding DFS starts at), and the red trajectories show the remaining walks back to the vertex $\cdot$. Figure 2f shows the five corresponding packets stored at the vertex w. The green trajectories are the walks `Walk`($P_{w,\cdot}$).

from T_w (except v_i, which is trimmed only when all requests in T_{v_i} are packed, and w, which is never trimmed). Note that the last partition of W_{v_i} may have a weight of less than $\beta_{v_i}/2$. In this case, this sub-walk will be pended and dealt with again when the next vertex v_{i+1} in T_w (in the post-order) is checked. Finally, if at the end of packing on T_w, there are still vertices left in the (trimmed) T_w, these left-over vertices are packed into one packet and stored at w. (See Figure 2.)

2.1.2 `Assignment`: Assigning packets (Phase 2)

After the procedure `Partition` in Section 2.1.1, the modified input tree $\mathcal{T}'$ is exactly the skeleton $\mathcal{T}_s$ with each request tree T_w rooted at the vertex $w \in \mathcal{T}_s$ packed into packets and stored at w. Moreover, every leaf in $\mathcal{T}'$ is a terminal of some server. Let $\mathcal{B}$ denote the set of all packets returned by `Partition`, and $\mathcal{B}_w$ denotes the set of packets stored at vertex

w ($\mathcal{B}_v$ is *empty* if there is no request tree rooted at v). By the process of `Partition`, the packets in $\mathcal{B}_w$ cover all requests in the request tree T_w. In the second phase of our algorithm, `Assignment`, we visit the vertices in $\mathcal{T}'$ in a post-order. When a vertex v is visited, we assign the packets stored in the subtree T_v but "far" enough from v to a server, which has its terminal nearby, and balance the load among the servers. The algorithm returns `fail` if it fails to balance the load of the servers.

Initial assignment. In the round of visiting v, we check its children $c_1, c_2, \cdots, c_k$ one by one. When the child c_i is checked, for any vertex w that is in the subtree T_{c_i} and $dist(w, v) > \frac{\vartheta}{2}$, we *release* the packets in $\mathcal{B}_w$ and assign the packets to a server s based at w (that is, $\tau(s) \in T_w$)[2]. Ties are broken arbitrarily. Note that some servers may have already been assigned packets in previous rounds as we visit the vertices in $\mathcal{T}'$ in a post-order.

To describe how to reassign the packets among servers, we first define the terminologies. We call the total weight of the packets assigned to server s the *work* of s. A server is *heavy*, *light*, or *normal* if its work is strictly greater than 3ϑ, strictly smaller than ϑ, or otherwise, respectively. If a server s' is assigned to serve packets that are initially assigned to another server s, s' is called *helping* for s. Otherwise, a server that helps no other servers is *free*.

Re-assignment for load-balancing. After the initial assignment when checking a child c_i of v, the servers with terminals in T_{c_i} may be light, normal, or heavy. For a server s that becomes heavy in this round, we find some light servers based at T_{c_i} to help it with some of the packets as follows. We first check if there is a light server s' that is based at T_{c_i} and already helped s with some packets. If so, we re-assign some packets released in this round to s'. The re-assignment stops once s or s' becomes normal. If at this moment, s is still heavy, then we search for another light helping server for s based at T_{c_i} and repeat the re-assignment. If there is no light server that helped s before, we find a light free server with its terminal in T_{c_i} that has not helped any other server. If all light servers based at T_{c_i} are helping other servers, and s is still heavy, the procedure `Assignment` returns `fail`.

Wrap the packets into walks. Finally, let $W_s \subseteq V(\mathcal{T}')$ be the set of vertices that store packets assigned to server s and L_s be the set of vertices on the $(\rho, \tau(s))$-path. The packets assigned to s can be wrapped up into a walk from ρ to $\tau(s)$ by traversing the subtree induced by $W_s \cup L_s$.

Figure 3 is an example of the released packets when checking c and the servers' status after the initial assignment. The pseudocode of `Assignment` can be found in the full version.

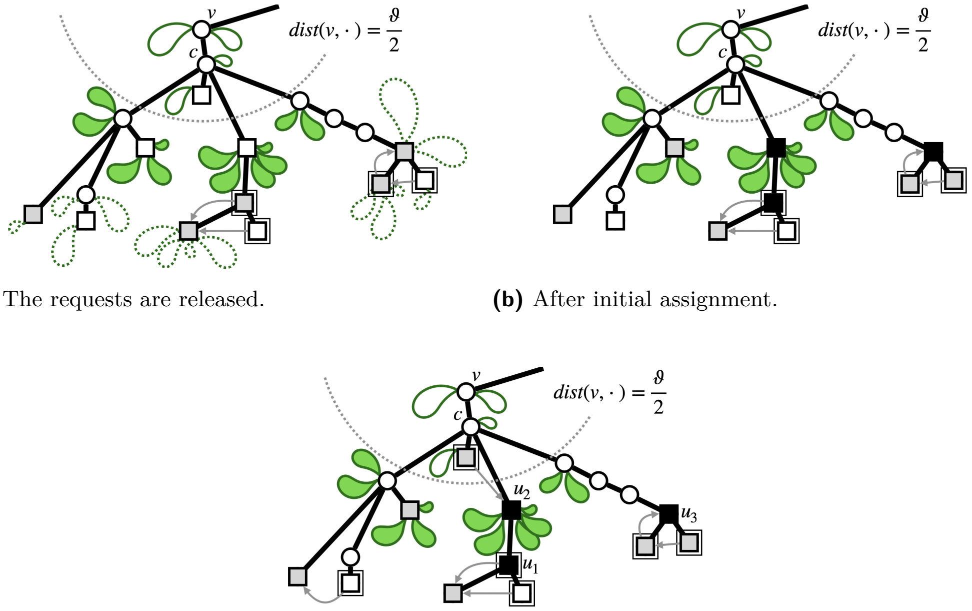
3 Analysis of PnB algorithm

Throughout the analysis, we use OPT to denote the optimal schedule. We also use OPT to denote the value of the optimal schedule when the context is clear.

Sizes of the packets successfully returned by `Partition` are bounded

Recall that the PnB algorithm is run on a modified input tree $\mathcal{T}$, where each leaf is a terminal of some server. With parameter ϑ, the procedure `Partition` either returns `fail` or modifies the input tree $\mathcal{T}$ into $\mathcal{T}'$ that contains the same vertices as $\mathcal{T}'$, and the packets are stored at

[2] Note that all these vertices on the path from v to c_i to w are all in $\mathcal{T}'$.


v
c
$dist(v,\cdot) = \frac{\vartheta}{2}$
v
c
$dist(v,\cdot) = \frac{\vartheta}{2}$
v
c
$dist(v,\cdot) = \frac{\vartheta}{2}$
u_2
u_3
u_1

(a) The requests are released.

(b) After initial assignment.

(c) After re-assignment.

Figure 3 Illustration of `Assignment`(ϑ) when v is visited and in the round of checking c. Each bean-shaped object is a packet from `Partition`(ϑ). The solid bean-shaped green objects are released when c is checked, as they are outside the range of $\frac{\vartheta}{2}$ from v. The hollow and dashed-lined ones are packets released in previous rounds, and the hollow and solid-lined ones have not been released. The squares indicate the servers' terminals. The square is white/black/gray if its corresponding server is light/heavy/normal. When a terminal is a double-layered square, its corresponding server is helping. The gray arrows from $\tau(s')$ to $\tau(s)$ indicate that s' is helping s with some packets. Note that in this example, after checking c, `Assignment`(ϑ) must return `fail` as the servers with the terminals at u_1, u_2 and u_3 are heavy, but there are no free light servers in T_c.

some of these vertices. Consider any request tree T_w rooted at $w \in \mathcal{T}'$. By construction, the procedure `Partition` packed each request in T_w into exactly one packet that is stored at the vertex w, as long as `Partition`(ϑ) does not return `fail`.

`Partition` fails when there is a path $P_{w,\ell}$, where $w \in \mathcal{T}'$ and ℓ is a leaf in T_w, such that $dist(w, \ell) > \frac{\vartheta}{2}$. Since we work on the modified input tree, the only vertex that can be a terminal in T_w is 2. Thus, any server that serves the request on ℓ has cost at least $2 \cdot dist(w, \ell)$. Therefore, we have the following lemma:

▶ **Lemma 2.** *If* `Partition`(ϑ) *returns* `fail`*, then* $OPT > \vartheta$*.*

On the other hand, if `Partition` returns a partition regarding ϑ, each packet B stored at vertex $w \in \mathcal{T}'$ is a closed walk containing w. For any packet B packed when $v_i \in T_w$ is visited, it was packed using a budget of $\vartheta - 2 \cdot dist(w, v_i)$. Since we only pack heavy enough request trees, and the vertices in the request tree are packed from the bottom up, the trip back to v_i after the budget is off is bounded by a function of ϑ and the distance between w and v_i. By careful calculations, we have the following lemma:

▶ **Lemma 3.** *If* `Partition`(ϑ) *does not return* `fail`*, for any packet* B*,* $weight(B) \leq 2\vartheta$*.*

`Assignment` feasibly returns a schedule with a cost of 4ϑ

After `Assignment`(ϑ), if it does not return `fail`, each server is assigned a set of packets, where each packet is a closed walk by the construction. Recall that a packet $B \in \mathcal{B}_w$ is assigned to a server s' either because B is assigned to s initially or s' is helping another server s with B. In the prior case, w is on $P_H(s') = P_{\rho,\tau(s')}$ and can be merged with $P_H(s')$ without incurring any detour edges. In the latter case, w is on $P_H(s) = P_{\rho,\tau(s)}$, and B was released when the algorithm visited some vertex $c \in P_{\rho,w}$, which is a common ancestor of $\tau(s')$ and $\tau(s)$. Thus, B can be merged with the walk `Walk`($P_{a,\tau(s)}$), and the merged walk can be merged with the path $P_H(s')$. Therefore, the packets assigned to server s' can be merged with $P_H(s')$ into a walk from the root to $\tau(s')$, and we have the following feasibility of the PnB algorithm (see the full version for a complete proof):

▶ **Theorem 4.** *Given instance $\mathcal{T}$, the transformed instance $\mathcal{T}'$, and a parameter of value ϑ, if* `Assignment`($\mathcal{T}', \vartheta$) *does not return* `fail`*,* PnB($\mathcal{T}, \vartheta$) *is feasible. Moreover, the cost of* PnB($\mathcal{T}, \vartheta$) *equals to the cost of* PnB($\mathcal{T}', \vartheta$)*.*

To bound the cost of any server, we first bound the cost spent on the server's detours. By the selection of helping servers, any server s' that helps another server s with packet $B \in \mathcal{B}_w$ is taking detours with a total distance of at most $2 \cdot dist(w, a)$, where a is the lowest common ancestor of $\tau(s')$ and $\tau(s)$. Since in each round, we only release "far-away" packets in terms of ϑ, $2 \cdot dist(w, a) \leq \vartheta$. Hence, the total detour length of s' is at most ϑ. Therefore, we have the following theorem:

▶ **Theorem 5.** *Given that* `Assignment`(ϑ) *does not return* `fail`*, the cost of any server on serving all packets assigned to it is at most 4ϑ.*

The lower bound of the optimal cost

There are two occasions that PnB returns `fail`; one is in the procedure `Partition`, and another one is in the procedure `Assignment`. In Lemma 2, we have shown that if `Partition` returns `fail`, the optimal cost must be larger than ϑ. Next, we consider the case when `Partition` successfully returns a partition of requests while `Assignment` returns `fail`.

We first argue that the partition returned by `Partition`(ϑ) is *sufficiently cost-efficient.* More specifically, if `Partition`(ϑ) does not return `fail`, and the optimal cost is at most ϑ, then total cost the optimal schedule has to pay is comparable to the total work of all packets.

▶ **Lemma 6.** *Given that* `Partition`(ϑ) *does not return* `fail`*, if $OPT \leq \vartheta$, then the sum of the cost of servers in the optimal schedule is at least $\sum_{packets\ B} weight(B)$. That is, $\sum_{s \in \mathcal{S}} cost^{OPT}(s) \geq \sum_{B \in \mathcal{B}} weight(B)$.*

Proof (Sketch). We denote n_e^* as the number of servers OPT sends over the edge e, and b_e as the number of packets constructed by `Partition`($\vartheta, \mathcal{T}$) that contain e. It is equivalent to prove that if $\text{OPT} \leq \vartheta$, $n_e^* \geq b_e$ for any edge $e \in \mathcal{F}_{\mathcal{R}}$. We prove this claim by induction on the edges with depths from the bottom up. Consider an edge $e = (u, v)$ in a request tree T_w, where u is v's parent. Denote $weight_{T_v}(B)$ the total weight of edges in packet B that are also in T_v. The total cost of servers in the optimal solution spent in the subtree T_v is $\sum_{e' \in T_v} n_{e'}^* \cdot weight(e') \geq \sum_{e' \in T_v} b_{e'} \cdot weight(e')$ by the inductive hypothesis. By construction, any packet that contains $e' \in T_v$ must contain the edge $e = (u, v)$. Therefore, the total weight $\sum_{e' \in T_v} b_{e'} \cdot weight(e') = \sum_{B: e \in B} weight_{T_v}(B)$.

Next, we bound $\sum_{B: e \in B} weight_{T_v}(B)$. By case distinction, any packet B containing the edge e that was packed when v or some descendent of v is visited has $weight_{T_v}(B) \geq \vartheta - 2 \cdot dist(w, v)$. Moreover, there is at most another packet containing the edge e that was packed when some ancestor of v is visited. Therefore, the total weight of the packets that contain e is at least $m \cdot (\vartheta - 2 \cdot dist(w, v)) + weight(\hat{B})$, where m is the total number of packets formed by visiting v or some descendent of v, and $\hat{B}$ is the packets packed when an ancestor of v is visited. Meanwhile, $b_e = m + \mathbb{1}[\text{there is at least one packet in } \hat{B}]$[3].

Finally, since $\text{OPT} \le \vartheta$, and v is in the request tree T_w, any server that travels to v has to finish its route in T_v with a cost of at most $\vartheta - 2 \cdot dist(w, v)$. Therefore, an optimal solution has to send at least $\frac{\sum_{B:e\in B} weight_{T_v}(B)}{\vartheta - 2\cdot dist(w,v)}$ servers over the edge e to serve all requests in T_v. Hence, $n_e^* \ge m + \mathbb{1}[\text{there is at least one packet in } \hat{B}] = b_e$, since n_e^* is integral. ◀

Next, we show that if `Assignment`(ϑ) returns `fail`, the average cost of the servers in the (failed) `Assignment`(ϑ) assignment on serving the packets must be strictly larger than ϑ.

▶ **Lemma 7.** *Given that* `Assignment`(ϑ) *returns* `fail`*, on average, each server in the PnB assignment is assigned by packets with total work strictly larger than* ϑ*.*

Proof (Sketch). Assume that `Assignment`(ϑ) returns `fail` when visiting v and checking the child c. Consider the set $\mathcal{S}'$ of servers that an optimal solution uses to serve the requests in T_c. There are two cases of the optimal solution: 1) there exists at least one server in $\mathcal{S}'$ who has its terminal outside T_c, or 2) all servers in $\mathcal{S}'$ have their terminals in T_c. It is easy to see in case 1, the optimal cost is at least ϑ due to any packets inside T_c are at least $\vartheta/2$ away from c.

For Case 2, we use a *helping forest* argument described below. Consider the servers' packets with their terminals in T_c at the moment when `Assignment`(ϑ) returns `fail`. We construct helping trees, which are rooted trees, where each node corresponds to a server by making s' a child of s if s' helps s with some packets.[4] In each helping tree, the root and the internal nodes are normal (or heavy if it is the server that triggers `Assignment`(ϑ) to return `fail`). Consider any internal node s and its children, which are sorted by the order that they help s. The servers s and all its children but the last one must be all normal. On the other hand, the last child can be normal or light. Furthermore, if the last child is light, it is a leaf, as it needs no one to help it. Hence, any internal node has at most one light leaf. The total work of an internal node s and its light leaf s_ℓ is at least 3ϑ, since $work(s) > 3\vartheta$ right before s_ℓ helps s. Therefore, the theorem is proven by the fact that the average work of the servers with terminals in T_c is strictly greater than $\frac{n\cdot\vartheta + \ell\cdot 3\vartheta}{n+2\ell} > \vartheta$, where ℓ is the number of internal node-light leaf pairs, and n is the number of other nodes in the helping forest. ◀

Now, we are ready to prove our main theorem.

▶ **Theorem 8.** *Given that* $\text{PnB}(\mathcal{T}, \vartheta)$ *returns* `fail`*,* $OPT > \vartheta$*.*

Proof. By the construction, `Partition`(ϑ) returns `fail`, the optimal cost must be strictly larger than ϑ. When `Assignment`(ϑ) returns `fail`, there are two cases of the optimal cost: 1) $\text{OPT} > \vartheta$, and 2) $\text{OPT} \le \vartheta$. Suppose on contrary that `Assignment`(ϑ) returns `fail` and $\text{OPT} \le \vartheta$. Then, $\text{OPT} \ge$ average cost of all servers $= \frac{\sum_{s\in\mathcal{S}} cost^{\text{OPT}}(s)}{|\mathcal{S}|} \ge \frac{\sum_{B\in\mathcal{B}} weight(B)}{|\mathcal{S}|}$, where the inequality is by Lemma 6. By Lemma 7, $\frac{\sum_{B\in\mathcal{B}} weight(B)}{|\mathcal{S}|} > \vartheta$. In result, $\text{OPT} > \vartheta$, which contradicts to the assumption that $\text{OPT} \le \vartheta$. ◀

By Theorem 5, Theorem 8, and the parametric pruning framework, we get the approximation ratio:

▶ **Theorem 9.** *PnB is a* 4*-approximation.*

[3] The function $\mathbb{1}[X]$ is 1 if statement X is true and is 0 otherwise.

[4] By the construction, a server s helped by another server will not help other servers, while a helping server s' may be helped by other servers later.

4 Runtime and complexity

4.1 Runtime

Based on the pseudocode, the runtime of PnB depends on a polynomial function of the number of vertices, the number of requests, the number of servers and the value of the optimal schedule. We can slightly improve it as follows. Previously, we search for ϑ^* through a linear scan of ϑ starting from 1, 2, and so on. The scan can be accelerated by binary search.

More specifically, we make ϑ starting from 1, double its value until PnB outputs a feasible schedule, and then binary search within the range up to the ϑ of the feasible schedule. Therefore, for the part of the value of optimal schedule, the runtime reduces to a logarithmic function of the value.

▶ **Theorem 10.** *PnB runs in $O(|V|^3 \log OPT(\mathcal{T}))$ time where $|V|$ is the number of vertices in the input tree $\mathcal{T}$.*

▶ **Corollary 11.** *PnB runs in polynomial time.*

By the binary search method, we can generalize the edge weights from integers to real numbers.

▶ **Theorem 12.** *For real number edge weights, PnB is $(4+\epsilon)$-approximation with runtime $O(|V|^3(4/\epsilon - 1 + \log\lceil OPT(\mathcal{T})\rceil))$ where $|V|$ is the number of vertices in the input tree $\mathcal{T}$.*

4.2 Complexity results

First, by reducing from 3-Partition, we have the following theorem for MRF:

▶ **Theorem 13.** *MRF is strongly NP-hard even if the input is a star.*

Then, we move on to the complexity of Scheduling-with-Locality. We show that the unrelated machine scheduling problem can be reduced to a special case of the scheduling with locality problem. Thus, the problem is more general than the unrelated machine scheduling problem in the sense of approximation ratios.

▶ **Theorem 14.** *For $\alpha < 1.5$, there does not exist a polynomial-time α-approximation algorithm for Scheduling-with-Locality even if we restrict that a server only gets discounts on the edges of the path from the source vertex to its terminal, unless $P = NP$.*

For the rest of the section, we show the following theorem by a reduction from 3-dimensional matching.

▶ **Theorem 15.** *For $\alpha < 2$, there does not exist a polynomial-time α-approximation algorithm for Scheduling-with-Locality even if the edge weights consist of only 0 and 1, unless $P = NP$.*

3-Dimensional matching. In the problem, there are disjoint vertex sets A, B and C with $|A| = |B| = |C| = n$. There is also a set of triples T. Each $t \in T$ is (a_t, b_t, c_t) for some $a_t \in A$, $b_t \in B$ and $c_t \in C$. There is a matching if we can find n triples that cover $A \cup B \cup C$. It is known that 3-dimensional matching is NP-hard.

Construction of the reduction. If $|T| < n$, we construct a trivial no-instance. Thus in the following, we assume $|T| \geq n$. For ease of the description, there is no discounts on any edge for each server by default. For each vertex $x \in A \cup B \cup C$, there is a corresponding request vertex r_x. There are additionally $|T| - n$ dummy request vertices D. For each triple $t = (a, b, c) \in T$, there are four vertices v_t, x_t, y_t and z_t. Each triple has a corresponding server. Let s be the corresponding server of t. Vertex v_t is the terminal of s. There are edges (v_t, x_t), (x_t, y_t) and (y_t, z_t). The weights of the three edges is 1 and the discounts of (x_t, y_t) and (y_t, z_t) for server s are 0. There are additionally edges (x_t, r_a), (y_t, r_b) and (z_t, r_c) all with weight 0. In addition, v_t is adjacent to all the dummy requests D, and these edges all have weight 1. Finally, v_t is also adjacent to the source vertex. The edge has weight 1, but the corresponding server s has a discount 0.

The construction has the following properties.

▶ **Observation 16.** *Consider two different servers s and s', and their corresponding triples $t = (a, b, c) \in T$ and $t' = (a', b', c') \in T$ respectively. Any walk starting from the terminal of s and ending at x'_t incurs a cost of at least 2 for server s if $a \neq a'$. The reversed walk (from x'_t to the terminal of s) also incurs a cost of at least 2 for server s if $a \neq a'$. Both statements are also true if we replace the vertices (a, a', x'_t) by (b, b', y'_t) or replace (a, a', x'_t) by (c, c', z'_t).*

▶ **Lemma 17.** *For Scheduling-with-Locality, the question of deciding if there exists a feasible schedule with makespan at most 2 is NP-hard.*

Proof of Theorem 15. By the construction and Observation 16, it is easy to see that any feasible schedule with a makespan strictly larger than 2 has a makespan larger than 4. Thus, the theorem follows. ◀

References

1 Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *Eur. J. Oper. Res.*, 246(2):345–378, 2015. doi:10.1016/j.ejor.2015.04.004.

2 Dor Arad, Yael Mordechai, and Hadas Shachnai. Tighter bounds for makespan minimization on unrelated machines. *CoRR*, abs/1405.2530, 2014. arXiv:1405.2530.

3 Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *J. Algorithms*, 59(1):1–18, 2006. doi:10.1016/j.jalgor.2005.01.007.

4 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 275–283. ACM, 1997. doi:10.1145/258533.258602.

5 Tolga Bektas and Adam N. Letchford. Using ℓ^p-norms for fairness in combinatorial optimisation. *Comput. Oper. Res.*, 120:104975, 2020. doi:10.1016/j.cor.2020.104975.

6 Binay K. Bhattacharya and Yuzhuang Hu. k-delivery traveling salesman problem on tree networks. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 325–336. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.325.

7 Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu. Traveling repairperson, unrelated machines, and other stories about average completion times. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 28:1–28:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.28.

8 Adrian Bock, Elyot Grant, Jochen Könemann, and Laura Sanità. The school bus problem on trees. *Algorithmica*, 67(1):49–64, 2013.

9 Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Comput. Ind. Eng.*, 99:300–313, 2016.

10 John L. Bruno and Peter J. Downey. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM J. Comput.*, 7(4):393–404, 1978. doi:10.1137/0207031.

11 Ioannis Caragiannis, Christos Kaklamanis, Panagiotis Kanellopoulos, and Maria Kyropoulou. The efficiency of fair division. *Theory Comput. Syst.*, 50(4):589–610, 2012.

12 L. Chen, K. Jansen, and G. Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *25th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–668, 2014.

13 Shichuan Deng, Jian Li, and Yuval Rabani. Generalized unrelated machine scheduling problem. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2898–2916. SIAM, 2023. doi:10.1137/1.9781611977554.ch110.

14 Alexander Fuerst and Prateek Sharma. Locality-aware load-balancing for serverless clusters. In Jon B. Weissman, Abhishek Chandra, Ada Gavrilovska, and Devesh Tiwari, editors, *HPDC '22: The 31st International Symposium on High-Performance Parallel and Distributed Computing, Minneapolis, MN, USA, 27 June 2022 – 1 July 2022*, pages 227–239. ACM, 2022. doi:10.1145/3502181.3531459.

15 Martin Gairing, Burkhard Monien, and Andreas Woclaw. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. *Theor. Comput. Sci.*, 380(1-2):87–99, 2007. doi:10.1016/j.tcs.2007.02.056.

16 Maxime Gonthier, Loris Marchal, and Samuel Thibault. Locality-aware scheduling of independent tasks for runtime systems. In Ricardo Chaves, Dora B. Heras, Aleksandar Ilic, Didem Unat, Rosa M. Badia, Andrea Bracciali, Patrick Diehl, Anshu Dubey, Oh Sangyoon, Stephen L. Scott, and Laura Ricci, editors, *Euro-Par 2021: Parallel Processing Workshops – Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30-31, 2021, Revised Selected Papers*, volume 13098 of *Lecture Notes in Computer Science*, pages 5–16. Springer, 2021. doi:10.1007/978-3-031-06156-1_1.

17 Sungjin Im and Shi Li. Improved approximations for unrelated machine scheduling. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2917–2946. SIAM, 2023. doi:10.1137/1.9781611977554.ch111.

18 Klaus Jansen, Marten Maack, and Alexander Mäcker. Scheduling on (un-)related machines with setup times. In *2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 145–154. IEEE, 2019. doi:10.1109/IPDPS.2019.00025.

19 Hessam Kooti and Eli Bozorgzadeh. Reconfiguration-aware task graph scheduling. In Eli Bozorgzadeh, João M. P. Cardoso, Rui Abreu, and Seda Ogrenci Memik, editors, *13th IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2013, Porto, Portugal, October 21-23, 2015*, pages 163–167. IEEE Computer Society, 2015. doi:10.1109/EUC.2015.33.

20 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 217–224. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.8.

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

22 Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, and Sören Riechers. Non-preemptive scheduling on machines with setup times. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures – 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 542–553. Springer, 2015. doi:10.1007/978-3-319-21840-3_45.

23 Jurij Mihelic and Borut Robic. Solving the k-center problem efficiently with a dominating set algorithm. *J. Comput. Inf. Technol.*, 13(3):225–234, 2005. doi:10.2498/cit.2005.03.05.

24 Mary Monroe and David M. Mount. A PTAS for the min-max euclidean multiple TSP. *CoRR*, abs/2112.04325, 2021. arXiv:2112.04325.

25 Javier Resano, Daniel Mozos, Diederik Verkest, Serge Vernalde, and Francky Catthoor. Run-time minimization of reconfiguration overhead in dynamically reconfigurable systems. In Peter Y. K. Cheung, George A. Constantinides, and José T. de Sousa, editors, *Field Programmable Logic and Application, 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003, Proceedings*, volume 2778 of *Lecture Notes in Computer Science*, pages 585–594. Springer, 2003. doi:10.1007/978-3-540-45234-8_57.

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

27 Zhou Xu and Brian Rodrigues. A 3/2-approximation algorithm for multiple depot multiple traveling salesman problem. In Haim Kaplan, editor, *Algorithm Theory – SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, volume 6139 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2010. doi:10.1007/978-3-642-13731-0_13.

On Line-Separable Weighted Unit-Disk Coverage and Related Problems

Gang Liu
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Haitao Wang
Kahlert School of Computing, University of Utah, Salt Lake City, UT, USA

Abstract

Given a set P of n points and a set S of n weighted disks in the plane, the disk coverage problem is to compute a subset of disks of smallest total weight such that the union of the disks in the subset covers all points of P. The problem is NP-hard. In this paper, we consider a line-separable unit-disk version of the problem where all disks have the same radius and their centers are separated from the points of P by a line ℓ. We present an $O(n^{3/2}\log^2 n)$ time algorithm for the problem. This improves the previously best work of $O(n^2\log n)$ time. Our result leads to an algorithm of $O(n^{7/2}\log^2 n)$ time for the halfplane coverage problem (i.e., using n weighted halfplanes to cover n points), an improvement over the previous $O(n^4\log n)$ time solution. If all halfplanes are lower ones, our algorithm runs in $O(n^{3/2}\log^2 n)$ time, while the previous best algorithm takes $O(n^2\log n)$ time. Using duality, the hitting set problems under the same settings can be solved with similar time complexities.

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

Keywords and phrases Line-separable, unit disks, halfplanes, geometric coverage, geometric hitting set

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.70

Related Version *Full Version*: http://arxiv.org/abs/2407.00329

Funding This research was supported in part by NSF under Grant CCF-2300356.

1 Introduction

Let P be a set of points, and S a set of disks in the plane such that each disk has a positive weight. The *disk coverage* problem asks for a subset of disks whose union covers all points and the total weight of the disks in the subset is minimized. The problem is NP-hard, even if all disks have the same radius and all disks have the same weight [12, 20]. Polynomial-time approximation algorithms have been proposed for the problem and many of its variants, e.g., [1, 7–9, 13, 17].

In this paper, we consider a *line-separable unit-disk* version of the problem where all disks have the same radius and their centers are separated from the points of P by a line ℓ (see Fig. 1). This version of the problem has been studied before. For the *unweighted case*, that is, all disks have the same weight, Ambühl, Erlebach, Mihalák, and Nunkesser [3] first solved the problem in $O(m^2n)$ time, where $n = |P|$ and $m = |S|$. An improved $O(nm + n\log n)$ time algorithm was later given in [11]. Liu and Wang [19] recently presented an $O((n + m)\log(n + m))$ time algorithm.[1] For the weighted case, Pederson and Wang [22] derived an algorithm of $O(nm\log(m + n))$ time or $O((m + n)\log(m + n) + \kappa\log m)$ time,

[1] The runtime of the algorithm in the conference paper of [19] was $m^{2/3}n^{2/3}2^{O(\log^*(m+n))} + O((n + m)\log(n + m))$, which has been improved to $O((n + m)\log(n + m))$ time in the latest arXiv version.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 70; pp. 70:1–70:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

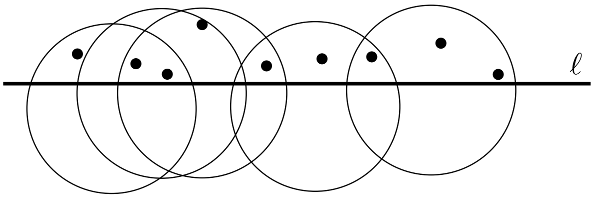


Figure 1 Illustrating the line-separable unit-disk case: All points of P are above ℓ while the centers of all disks are below ℓ.

where κ is the number of pairs of disks that intersect and $\kappa = O(m^2)$ in the worst case. In this paper, we propose an algorithm of $O(n\sqrt{m}\log^2 m + (m+n)\log(m+n))$ time for the weighted case. In addition to the improvement over the previous work, perhaps theoretically more interesting is that the runtime of our algorithm is subquadratic.

The halfplane coverage problem. If every disk of S is a halfplane, then the problem becomes the *halfplane coverage problem*. To solve the problem, Pedersen and Wang [22] showed that the problem can be reduced to $O(n^2)$ instances of the *lower-only halfplane coverage problem* in which all halfplanes are lower halfplanes; this reduction works for both the unweighted and the weighted cases. Consequently, if the lower-only problem can be solved in $O(T)$ time, then the general problem (i.e., S has both lower and upper halfplanes) is solvable in $O(n^2 \cdot T)$ time.

For the weighted lower-only problem, Chan and Grant [8] first gave an algorithm that runs in $O((m+n)^4)$ time. As observed in [22], the lower-only halfplane coverage problem is actually a special case of the line-separable unit-disk coverage problem. Indeed, let ℓ be a horizontal line below all the points of P. Then, since each halfplane of S is a lower halfplane, it can be considered a disk of infinite radius with center below ℓ. In this way, the lower-only halfplane coverage problem becomes an instance of the line-separable unit-disk coverage problem. As such, with their algorithm for the weighted line-separable unit-disk coverage problem, Pederson and Wang [22] solved the weighted lower-only halfplane coverage problem in $O(nm + n\log n)$ time or $O((m+n)\log(m+n) + m^2\log m)$ time. Using our new algorithm for the weighted line-separable unit-disk coverage problem, the weighted lower-only halfplane coverage problem can now be solved in $O(n\sqrt{m}\log^2 m + (m+n)\log(m+n))$ time.

The unweighted lower-only halfplane coverage problem can be solved faster. Indeed, since the problem is a special case of the unweighted line-separable unit-disk coverage problem, with the $O((n+m)\log(n+m))$ time algorithm of Liu and Wang [19] for the latter problem, the unweighted lower-only halfplane coverage problem is solvable in $O((n+m)\log(n+m))$ time. Wang and Xue [24] derived another $O((n+m)\log(n+m))$ time algorithm for the unweighted lower-only halfplane coverage problem with a different approach (which does not work for the unit-disk problem); they also solved the general unweighted halfplane coverage problem in $O(n^{4/3}\log^{5/3} n \log^{O(1)} \log n)$ time. In addition, by a reduction from the set equality problem [4], a lower bound of $\Omega((n+m)\log(n+m))$ is proved in [24] for the lower-only halfplane coverage problem under the algebraic decision tree model.

The hitting set problem. A related problem is the hitting set problem in which each point of P has a positive weight and we seek to find a subset of points with minimum total weight so that each disk of S contains at least one point of the subset. Since the disks of S have the same radius, the problem is actually "dual" to the disk coverage problem. More specifically, if we consider the set of unit disks centered at the points of P as a set of "dual disks" and consider the centers of the disks of S as a set of “dual points”, then the hitting set problem on P and S is equivalent to finding a minimum weight subset of dual disks whose union covers all dual points. Consequently, applying our new weighted case line-separable unit-disk coverage algorithm in this paper solves the weighted line-separable unit-disk hitting set problem in $O(m\sqrt{n}\log^2 n + (m+n)\log(m+n))$ time; applying the $O((m+n)\log(m+n))$ time algorithm in [19] for the unweighted line-separable unit-disk coverage algorithm solves the unweighted line-separable unit-disk hitting set problem in $O((m+n)\log(m+n))$ time.

If every disk of S is a halfplane, then the problem becomes the *halfplane hitting set* problem. Har-Peled and Lee [14] first solved the weighted problem in $O((m+n)^6)$ time. Liu and Wang [18] showed that the problem can be reduced to $O(n^2)$ instances of the *lower-only halfplane hitting set problem* in which all halfplanes are lower halfplanes; this reduction works for both the unweighted and the weighted cases. Consequently, if the lower-only problem can be solved in $O(T)$ time, then the general problem can be solved in $O(n^2 \cdot T)$ time. For the lower-only problem, as in the coverage problem, it is a special case of the line-separable unit-disk hitting set problem; consequently, the weighted and unweighted cases can be solved in $O(m\sqrt{n}\log^2 n + (m+n)\log(m+n))$ time using our new algorithm in this paper and $O((m+n)\log(m+n))$ time using the algorithm in [19], respectively.

Other related work. Pedersen and Wang [22] actually considered a *line-constrained* disk coverage problem, where disk centers are on the x -axis while the points of P can be anywhere in the plane, but the disks may have different radii. They solved the weighted case in $O(nm + n\log n)$ time or $O((m+n)\log(m+n) + \kappa\log m)$ time, where κ is the number of pairs of disks that intersect. For the unweighted case, Liu and Wang [19] gave an algorithm of $O((n+m)\log(m+n) + m\log m\log n)$ time. The line-constrained disk hitting set problem was also studied by Liu and Wang [18], where an $O((m+n)\log(m+n) + \kappa\log m)$ time algorithm was derived for the weighted case, matching the time complexity of the above line-constrained disk coverage problem. Other types of line-constrained problems have also been considered in the literature, e.g., [2,5,6,15,16,21,25].

Our approach. Our algorithm for the weighted line-separable unit-disk coverage problem is essentially a dynamic program. The algorithm description is quite simple and elegant. However, it is not straightforward to prove its correctness. To this end, we show that our algorithm is consistent with the algorithm in [22] for the same problem; one may view our algorithm as a different implementation of the algorithm in [22]. Another challenge of our approach lies in its implementation. More specifically, our algorithm has two key operations, and the efficiency of the algorithm hinges on how to perform these operations efficiently. For this, we construct a data structure based on building a cutting on the disks of S [10]. Although we do not have a good upper bound on the runtime of each individual operation of the algorithm, we manage to bound the total time of all operations in the algorithm by $O(n\sqrt{m}\log^2 m + (m+n)\log(m+n))$.

Outlines. The rest of the paper is structured as follows. After introducing some notation in Section 2, we describe our algorithm and prove its correctness in Section 3. The implementation of the algorithm is presented in Section 4.

2 Preliminaries

We follow the notation defined in Section 1, e.g., P, S, m, n, ℓ. All disks of S have the same radius, which we call *unit disks*. Without loss of generality, we assume that ℓ is the x-axis and all points of P are above ℓ while all centers of disks of S are below ℓ. Note that when we say that a point is above (or below) ℓ, we allow the case where the point is on ℓ.

We assume that each point of P is covered by at least one disk since otherwise there would be no solution. Our algorithm can check whether this assumption is met. For ease of discussion, we make a general position assumption that no point of P lies on the boundary of a disk and no two points of P have the same x-coordinate.

We call a subset S' of S a *feasible subset* if the union of all disks of S' covers all points of P. If S' is a feasible subset of minimum total weight, then S' is called an *optimal subset*. Let δ_{opt} denote the total weight of all disks in an optimal subset; we call δ_{opt} the *optimal objective value*.

For any point q in the plane, let $S_q \subseteq S$ denote the subset of disks containing q; define $\overline{S_q} = S \setminus S_q$. For each disk $s \in S$, let $w(s)$ denote its weight.

3 Algorithm description and correctness

We now present our algorithm. As mentioned above, the algorithm description is quite simple. The challenging part is to prove its correctness and implement it efficiently. In the following, we first describe the algorithm in Section 3.1, and then establish its correctness in Section 3.2. The algorithm implementation will be elaborated in Section 4.

3.1 Algorithm description

We first sort the points of P from left to right as $p_1, p_2, \ldots, p_n$. Our algorithm then processes the points of P in this order. For each point $p_i \in P$, the algorithm computes a value δ_i. The algorithm also maintains a value $cost(s)$ for each disk $s \in S$, which is initialized to its weight $w(s)$. The pseudocode of the algorithm is given in Algorithm 1.

Algorithm 1 The primal algorithm.

```
Input: The points of $P$ are sorted from left to right as $p_1, p_2, \ldots, p_n$
Output: The optimal objective value $\delta_{\text{opt}}$
$cost(s) \leftarrow w(s)$, for all disks $s \in S$;
for $i \leftarrow 1$ to $n$ do
    $\delta_i \leftarrow \min_{s \in S_{p_i}} cost(s)$;                  // FindMinCost Operation
    $cost(s) \leftarrow w(s) + \delta_i$ for all disks $s \in \overline{S_{p_i}}$;     // ResetCost Operation
end
return $\delta_n$;
```

The algorithm is essentially a dynamic program. We prove in Section 3.2 that the value δ_n returned by the algorithm is equal to δ_{opt}, the optimal objective value. To find an optimal subset, one could slightly modify the algorithm following the standard dynamic programming backtracking technique. More specifically, if δ_n is equal to $cost(s)$ for some disk $s \in S_{p_n}$, then s should be reported as a disk in the optimal subset. Suppose that $cost(s)$ is equal to $w(s) + \delta_i$ for some point p_i. Then δ_i is equal to $cost(s')$ for some disk $s' \in S_{p_i}$ and s' should be reported as a disk in the optimal subset. We continue this backtracking process until a disk whose cost is equal to its own weight is reported (in which case all points of P are covered by the reported disks).

For reference purposes, we use FindMinCost to refer to the operation in Line 3 and use ResetCost to refer to the operation in Line 4 of Algorithm 1. The efficiency of the algorithm hinges on how to implement these two *key operations*, which will be discussed in Section 4.

3.2 Correctness of Algorithm 1

We prove that Algorithm 1 is correct, i.e., prove $\delta_n = \delta_{\mathrm{opt}}$. To this end, we show that our algorithm is consistent with the algorithm of Pederson and Wang [22] for the same problem, or alternatively, our algorithm provides a different implementation of their algorithm. Their algorithm first reduces the problem to a 1D interval coverage problem and then solves the interval coverage problem by a dynamic programming algorithm. In the following, we first review their problem reduction in Section 3.2.1 and then explain their dynamic programming algorithm in Section 3.2.2. Finally, we show that our algorithm is essentially an implementation of their dynamic programming algorithm in Section 3.2.3.

3.2.1 Reducing the problem to an interval coverage problem

For convenience, let p_0 (resp., p_{n+1}) be a point to the left (resp., right) all the points of P and is not contained in any disk of S.

Consider a disk $s \in S$. We say that a subsequence $P[i,j]$ of P with $1 \leq i \leq j \leq n$ is a *maximal subsequence covered* by s if all points of $P[i,j]$ are covered by s but neither p_{i-1} nor p_{j+1} is (it is well defined due to p_0 and p_{n+1}). Define $F(s)$ as the set of all maximal subsequences covered by s. It is easy to see that the subsequences of $F(s)$ are pairwise disjoint.

For each point p_i of P, we vertically project it on the x-axis ℓ; let p_i^* denote the projection point. Let P^* denote the set of all such projection points. Due to our general position assumption that no two points of P have the same x-coordinate, all points of P^* are distinct. For any $1 \leq i \leq j \leq n$, we use $P^*[i,j]$ to denote the subsequence $p_i^*, p_{i+1}^*, \ldots, p_j^*$.

Next, we define a set S^* of line segments on ℓ as follows. For each disk $s \in S$ and each subsequence $P[i,j] \in F(s)$, we create a segment for S^*, denoted by $s^*[i,j]$, with the left endpoint at p_i^* and the right endpoint at p_j^*. As such, $s^*[i,j]$ covers all points of $P^*[i,j]$ and does not cover any point of $P^* \setminus P^*[i,j]$. We let the weight of $s^*[i,j]$ be equal to $w(s)$. We say that $s^*[i,j]$ is *defined* by the disk s.

Consider the following *interval coverage* problem (i.e., each segment of S^* can also be considered an interval of ℓ): Find a subset of segments of S^* of minimum total weight such that the union of the segments in the subset covers all points of P^*. Pederson and Wang [22] proved that an optimal solution to this interval coverage problem corresponds to an optimal solution to the original disk coverage problem on P and S. More specifically, suppose that S^*_{opt} is an optimal subset of the interval coverage problem. Then, we can obtain an optimal subset S_{opt} for the disk coverage problem as follows: For each segment $s^*[i,j] \in S^*_{\mathrm{opt}}$, we add the disk that defines $s^*[i,j]$ to S_{opt}. It is proved in [22] that S_{opt} thus obtained is an optimal subset of the disk coverage problem. Note that since a disk of S may define multiple segments of S^*, a potential issue with S_{opt} is that a disk may be included in S_{opt} multiple times (i.e., if multiple segments defined by the disk are in S^*_{opt}); but this is proved impossible [22].

3.2.2 Solving the interval coverage problem

We now explain the dynamic programming algorithm in [22] for the interval coverage problem.

Let p_0^* be the vertical projection of p_0 on ℓ. Note that $p_0^*, p_1^*, \ldots, p_n^*$ are sorted on ℓ from left to right. For each segment $s^* \in S^*$, let $w(s^*)$ denote the weight of s^*.

For each segment $s^* \in S^*$, define f_{s^*} as the index of the rightmost point of $P^* \cup \{p_0^*\}$ strictly to the left of the left endpoint of s^*. Note that f_{s^*} is well defined due to p_0^*.

For each $i \in [1, n]$, define δ_i^* as the minimum total weight of a subset of segments of S^* whose union covers all points of $P^*[1, i]$. The goal of the interval coverage problem is to compute δ_n^*, which is equal to δ_{opt} [22]. For convenience, we let $\delta_0^* = 0$. For each segment $s^* \in S^*$, define $cost(s^*) = w(s^*) + \delta^*_{f_{s^*}}$. One can verify that $\delta_i^* = \min_{s^* \in S^*_{p_i^*}} cost(s^*)$, where $S^*_{p_i^*} \subseteq S^*$ is the subset of segments that cover p_i^*. This is the recursive relation of the dynamic programming algorithm.

Assuming that the indices f_{s^*} for all disks $s^* \in S^*$ have been computed, the algorithm works as follows. We sweep a point q on ℓ from left to right. Initially, q is at p_0^*. During the sweeping, we maintain the subset $S_q^* \subseteq S^*$ of segments that cover q. The algorithm maintains the invariant that the cost of each segment of S_q^* is already known and the values δ_i^* for all points $p_i^* \in P^*$ to the left of q have been computed. An event happens when q encounters an endpoint of a segment of S^* or a point of P^*. If q encounters a point $p_i^* \in P^*$, then we find the segment of S_q^* with the minimum cost and assign the cost value to δ_i^*. If q encounters the left endpoint of a segment s^*, we set $cost(s^*) = w(s^*) + \delta^*_{f_{s^*}}$ and then insert s^* into S_q^*. If q encounters the right endpoint of a segment, we remove the segment from S_q^*. The algorithm finishes once q meets p_n^*, at which event δ_n^* is computed.

Remark. It was shown in [22] that the above dynamic programming algorithm for the interval coverage problem can be implemented in $O((|P^*| + |S^*|) \log(|P^*| + |S^*|))$. While $|P^*| = n$, $|S^*|$ may be relatively large. A straightforward upper bound for $|S^*|$ is $O(nm)$. Pederson and Wang [22] proved another bound $|S^*| = O(n + m + \kappa)$, where κ is the number of pairs of disks that intersect. This leads to their algorithm of $O(nm \log(m + n) + n \log n)$ time or $O((m + n) \log(m + n) + \kappa \log m)$ time for the original disk coverage problem on P and S.

3.2.3 Correctness of our algorithm

Next, we argue that $\delta_n = \delta_n^*$, which will establish the correctness of Algorithm 1 since $\delta_n^* = \delta_{\text{opt}}$. In fact, we will show that $\delta_i = \delta_i^*$ for all $1 \le i \le n$. We prove it by induction.

As the base case, we first argue $\delta_1 = \delta_1^*$. To see this, by definition, $\delta_1 = \min_{s \in S_{p_1}} w(s)$ because $cost(s) = w(s)$ initially for all disks $s \in S$. For δ_1^*, notice that $f_{s^*} = 0$ for every segment $s^* \in S^*_{p_1^*}$. Since $\delta_0^* = 0$, we have $\delta_1^* = \min_{s^* \in S^*_{p_1^*}} w(s^*)$. By definition, a segment $s^* \in S^*$ covers p_1^* only if the disk of S defining s^* covers p_1, and the segment has the same weight as the disk. Therefore, s^* is in $S^*_{p_1^*}$ only if the disk of S defining s^* is in S_{p_1}. On the other hand, if a disk s covers p_1, then s must define exactly one segment in S^* covering p_1^*. Hence, for each disk $s \in S_{p_1}$, it defines exactly one segment in $S^*_{p_1^*}$ with the same weight. This implies that δ_1^* is equal to the minimum weight of all disks of S covering p_1, and therefore, $\delta_1^* = \delta_1$ must hold.

Consider any i with $2 \le i \le n$. Assuming that $\delta_j = \delta_j^*$ for all $1 \le j < i$, we now prove $\delta_i = \delta_i^*$. Recall that $\delta_i = \min_{s \in S_{p_i}} cost(s)$ and $\delta_i^* = \min_{s^* \in S^*_{p_i^*}} cost(s^*)$. As argued above, each disk $s \in S_{p_i}$ defines a segment in $S^*_{p_i^*}$ with the same weight and each segment $s^* \in S^*_{p_i^*}$ is defined by a disk in S_{p_i} with the same weight. Let s^* be the segment of $S^*_{p_i^*}$ defined by a disk $s \in S_{p_i}$. To prove $\delta_i = \delta_i^*$, it suffices to show that $cost(s)$ of s is equal to $cost(s^*)$ of s^*. To see this, first note that $w(s) = w(s^*)$. By definition, $cost(s^*) = w(s^*) + \delta^*_{f_{s^*}}$. For notational convenience, let $j = f_{s^*}$. By definition, all points of $P^*[j + 1, i]$ are covered by the segment s^* but the point p_j^* is not. Therefore, all points of $P[j + 1, i]$ are covered by the disk s but p_j is not. As such, during the ResetCost operation of the j-th iteration of the for loop in Algorithm 1, $cost(s)$ will be set to $w(s) + \delta_j$; furthermore, $cost(s)$ will not be reset again during the i'-th iteration for all $j + 1 \leq i' \leq i$. Therefore, we have $cost(s) = w(s) + \delta_j$ at the beginning of the i-th iteration of the algorithm. Since $\delta_j = \delta_j^*$ holds by induction hypothesis and $w(s) = w(s^*)$, we obtain $cost(s) = cost(s^*)$. This proves $\delta_i = \delta_i^*$.

The correctness of Algorithm 1 is thus established.

Remark. The above proof for $\delta_n = \delta_{\text{opt}}$ also implies that $\delta_i = \delta_{\text{opt}}^i$ for all $1 \leq i \leq n-1$, where δ_{opt}^i is the minimum total weight of a subset of disks whose union covers all points of $P[1, i]$. Indeed, we can apply the same proof to the points of $P[1, i]$ only. Observe that δ_i will never change after the i-th iteration of Algorithm 1.

4 Algorithm implementation

In this section, we discuss the implementation of Algorithm 1. Specifically, we describe how to implement the two key operations FindMinCost and ResetCost. A straightforward method can implement each operation in $O(m)$ time, resulting in a total $O(mn + n \log n)$ time of the algorithm. Note that this is already a logarithmic factor improvement over the previous work of Pederson and Wang [22]. In the following, we present a faster approach of $O(n\sqrt{m} \log^2 m + (m + n) \log(m + n))$ time.

Duality. Recall that the points of P are sorted from left to right as $p_1, p_2, \ldots, p_n$. In fact, we consider the problem in the "dual" setting. For each point $p_i \in P$, let d_i denote the unit disk centered at p_i, and we call d_i the *dual disk* of p_i. For each disk $s \in S$, let q_s denote the center of s, and we call q_s the *dual point* of s. We define the weight of q_s to be equal to $w(s)$. We use D to denote the set of all dual disks and Q the set of all dual points. For each dual point $q \in Q$, let $w(q)$ denote its weight. Because all disks of S are unit disks, we have the following observation.

▶ **Observation 1.** *A disk $s \in S$ covers a point $p_i \in P$ if and only if the dual point q_s is covered by the dual disk d_i.*

For any disk $d_i \in D$, let Q_{d_i} denote the subset of dual points of Q that are covered by d_i, i.e., $Q_{d_i} = Q \cap d_i$. Define $\overline{Q_{d_i}} = Q \setminus Q_{d_i}$. In light of Observation 1, Algorithm 1 is equivalent to the following Algorithm 2.

Algorithm 2 An algorithm "dual" to Algorithm 1.

```
$cost(q) \leftarrow w(q)$, for all dual points $q \in Q$;
for $i \leftarrow 1$ to $n$ do
    $\delta_i \leftarrow \min_{q \in Q_{d_i}} cost(q)$;                        // FindMinCost Operation
    $cost(q) \leftarrow w(q) + \delta_i$ for all dual points $q \in \overline{Q_{d_i}}$;     // ResetCost Operation
end
return $\delta_n$;
```

In the following, we will present an implementation for Algorithm 2, and in particular, for the two operations FindMinCost and ResetCost.

For each disk $d_i \in D$, since its center is above the x-axis ℓ and all points of Q are below ℓ, only the portion of d_i below ℓ matters to Algorithm 2. We call the boundary portion of d_i below ℓ the *lower arc* of d_i. Let H denote the set of the lower arcs of all disks of D.

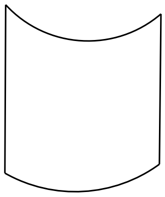


Figure 2 Illustrating a pseudo-trapezoid.

Cuttings. Our algorithm will need to construct a cutting on the arcs of H [10]. We explain this concept first. Note that $|H| = n$. For a parameter r with $1 \leq r \leq n$, a $(1/r)$-*cutting* Ξ of size $O(r^2)$ for H is a collection of $O(r^2)$ constant-complexity cells whose union covers the entire plane and whose interiors are pairwise disjoint such that the interior of each cell $\sigma \in \Xi$ is crossed by at most n/r arcs of H, i.e., $|H_\sigma| \leq n/r$, where H_σ is the subset of arcs of H that cross the interior of σ (H_σ is often called the *conflict list* in the literature). Let D_σ be the subset of disks of D whose lower arcs are in H_σ. Hence, we also have $|D_\sigma| \leq n/r$.

We actually need to construct a *hierarchical cutting* for H [10]. We say that a cutting Ξ' *c-refines* another cutting Ξ if each cell of Ξ' is completely inside a single cell of Ξ and each cell of Ξ contains at most c cells of Ξ'. Let Ξ_0 denote the cutting with a single cell that is the entire plane. We define cuttings $\{\Xi_0, \Xi_1, ..., \Xi_k\}$, in which each Ξ_i, $1 \leq i \leq k$, is a $(1/\rho^i)$-cutting of size $O(\rho^{2i})$ that c-refines Ξ_{i-1}, for two constants ρ and c. By setting $k = \lceil \log_\rho r \rceil$, the last cutting Ξ_k is a $(1/r)$-cutting. The sequence $\{\Xi_0, \Xi_1, ..., \Xi_k\}$ is called a *hierarchical* $(1/r)$-*cutting* for H. If a cell σ' of Ξ_{i-1}, $1 \leq i \leq k$, contains cell σ of Ξ_i, we say that σ' is the *parent* of σ and σ is a *child* of σ'. We can also define *ancestors* and *descendants* correspondingly. As such, the hierarchical $(1/r)$-cutting can be viewed as a tree structure with the single cell of Ξ_0 as the root. We often use Ξ to denote the set of all cells in all cuttings Ξ_i, $0 \leq i \leq k$. The total number of cells of Ξ is $O(r^2)$ [10].

A hierarchical $(1/r)$-cutting of H can be computed in $O(nr)$ time, e.g., by the algorithm in [23], which adapts Chazelle's algorithm [10] for hyperplanes. The algorithm also produces the conflict lists H_σ (and thus D_σ) for all cells $\sigma \in \Xi$, implying that the total size of these conflict lists is bounded by $O(nr)$. In particular, each cell of the cutting produced by the algorithm of [23] is a (possibly unbounded) *pseudo-trapezoid* that typically has two vertical line segments as left and right sides, a sub-arc of an arc of H as a top side (resp., bottom side) (see Fig. 2).

In what follows, we first discuss a preprocessing step in Section 4.1. The algorithms for handling the two key operations are described in the subsequent two subsections, respectively. Section 4.4 finally summarizes everything.

4.1 Preprocessing

In order to handle the two key operations, we first perform some preprocessing work before we run Algorithm 2. As discussed above, we first sort all points of P from left to right. In the following, we describe a data structure which will be used to support the two key operations.

We start by computing a hierarchical $(1/r)$-cutting $\{\Xi_0, \Xi_1, ..., \Xi_k\}$ for H in $O(nr)$ time [10,23], for a parameter $1 \leq r \leq n$ to be determined later. We follow the above notation, e.g., σ, H_σ, D_σ, Ξ. As discussed above, the cutting algorithm also produces the conflict lists H_σ (and thus D_σ) for all cells $\sigma \in \Xi$. Using the conflict lists, we compute a list $L(d_i)$ for each disk $d_i \in D$, where $L(d_i)$ comprises all cells $\sigma \in \Xi$ such that $d_i \in D_\sigma$. Computing $L(d_i)$ for all disks $d_i \in D$ can be done in $O(\sum_{\sigma \in \Xi} |H_\sigma|)$ time by simply traversing the conflict lists H_σ of all cells $\sigma \in \Xi$, which takes $O(nr)$ time as $\sum_{\sigma \in \Xi} |H_\sigma| = O(nr)$. Note that this also implies $\sum_{d_i \in D} |L(d_i)| = O(nr)$.

For any region R in the plane, let $Q(R)$ denote the subset of points of Q that are inside R, i.e., $Q(R) = Q \cap R$.

For simplicity, we assume that no point of Q is on the boundary of any cell of Ξ. This implies that each point of Q is in the interior of a single cell of Ξ_i, for all $0 \leq i \leq k$. We compute the subset $Q(\sigma)$ of all cells σ in the last cutting Ξ_k. This can be done by a point location procedure as follows. For each point $q \in Q$, starting from the only cell of Ξ_0, we find the cell σ_i of Ξ_i containing q, for each $0 \leq i \leq k$. More precisely, suppose that σ_i is known. To find σ_{i+1}, we simply check all $O(1)$ children of σ_i and find the one that contains q, which takes $O(1)$ time. As such, processing all points of Q takes $O(m \log r)$ time as $k = O(\log r)$, after which $Q(\sigma)$ for all cells $\sigma \in \Xi_k$ are computed. We explicitly store $Q(\sigma)$ for all cells $\sigma \in \Xi_k$. Note that the subsets $Q(\sigma)$ for all cells $\sigma \in \Xi_k$ form a partition of Q. Therefore, we have the following observation.

▶ **Observation 2.** $\sum_{\sigma \in \Xi_k} |Q(\sigma)| = m$.

Note that a cell $\sigma \in \Xi$ is the ancestor of another cell $\sigma' \in \Xi$ (alternatively, σ' is a descendant of σ) if and only if σ fully contains σ'. For convenience, we consider σ an ancestor of itself but not a descendant of itself. Let $A(\sigma)$ denote the set of all ancestors of σ and $B(\sigma)$ the set of all descendants of σ. Hence, σ is in $A(\sigma)$ but not in $B(\sigma)$. Let $C(\sigma)$ denote the set of all children of σ Clealy, $|A(\sigma)| = O(\log r)$ and $|C(\sigma)| = O(1)$.

Variables and algorithm invariants. For each point $q \in Q$, we associate with it a variable $\lambda(q)$. For each cell $\sigma \in \Xi$, we associate with it two variables: $minCost(\sigma)$ and $\lambda(\sigma)$. If $|Q(\sigma)| = \emptyset$, then $minCost(\sigma) = \infty$ and $\lambda(\sigma) = 0$ always hold during the algorithm. Our algorithm for handling the two key operations will maintain the following two invariants.

▶ **Algorithm Invariant 1.** *For any point $q \in Q$, $cost(q) = w(q) + \lambda(q) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma')$, where σ is the cell of Ξ_k that contains q.*

▶ **Algorithm Invariant 2.** *For each cell $\sigma \in \Xi$ with $Q(\sigma) \neq \emptyset$, if σ is a cell of Ξ_k, then $minCost(\sigma) = \min_{q \in Q(\sigma)}(w(q) + \lambda(q))$; otherwise, $minCost(\sigma) = \min_{\sigma' \in C(\sigma)}(minCost(\sigma') + \lambda(\sigma'))$.*

The above algorithm invariants further imply the following observation.

▶ **Observation 3.** *For each cell $\sigma \in \Xi$ with $Q(\sigma) \neq \emptyset$, $\min_{q \in Q(\sigma)} cost(q) = minCost(\sigma) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma')$.*

Proof. We prove the observation by induction. For the base case, consider a cell σ of the last cutting Ξ_k. By the two algorithm invariants, we have

$$\begin{aligned}
\min_{q \in Q(\sigma)} cost(q) &= \min_{q \in Q(\sigma)} \Big(w(q) + \lambda(q) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma') \Big) && \text{by Algorithm Invariant 1} \\
&= \min_{q \in Q(\sigma)} \Big(w(q) + \lambda(q) \Big) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma') \\
&= minCost(\sigma) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma'). && \text{by Algorithm Invariant 2}
\end{aligned}$$

This proves the observation for σ.

Now consider a cell $\sigma \in \Xi \setminus \Xi_k$. We assume that the observation holds for all children σ' of σ, i.e., $\min_{q \in Q(\sigma')} cost(q) = minCost(\sigma') + \sum_{\sigma'' \in A(\sigma')} \lambda(\sigma'')$. Then, we have

$$\begin{aligned}
\min_{q \in Q(\sigma)} cost(q) &= \min_{\sigma' \in C(\sigma)} \min_{q \in Q(\sigma')} cost(q) \\
&= \min_{\sigma' \in C(\sigma)} \Big(minCost(\sigma') + \sum_{\sigma'' \in A(\sigma')} \lambda(\sigma'') \Big) && \text{by induction hypothesis} \\
&= \min_{\sigma' \in C(\sigma)} \Big(minCost(\sigma') + \lambda(\sigma') \Big) + \sum_{\sigma'' \in A(\sigma)} \lambda(\sigma'') \\
&= minCost(\sigma) + \sum_{\sigma'' \in A(\sigma)} \lambda(\sigma''). && \text{by Algorithm Invariant 2}
\end{aligned}$$

This proves the observation. ◀

For each cell $\sigma \in \Xi$, we also maintain $\mathcal{L}(\sigma)$, a list comprising all descendant cells σ' of σ whose values $\lambda(\sigma')$ are not zero and all points $q \in Q(\sigma)$ whose values $\lambda(q)$ are not zero. As $\mathcal{L}(\sigma)$ has both cells of $B(\sigma)$ and points of $Q(\sigma)$, for convenience, we use an “element” to refer to either a cell or a point of $\mathcal{L}(\sigma)$. As such, for any element $e \in B(\sigma) \cup Q(\sigma)$ with $e \notin \mathcal{L}(\sigma)$, $\lambda(e) = 0$ must hold. As will be seen later, whenever the algorithm sets $\lambda(\sigma)$ to a nonzero value for a cell $\sigma \in \Xi$, σ will be added to $\mathcal{L}(\sigma')$ for every ancestor σ' of σ with $\sigma' \neq \sigma$. Similarly, whenever the algorithm sets $\lambda(q)$ to a nonzero value for a point q, then q will be added to $\mathcal{L}(\sigma')$ for every cell $\sigma' \in A(\sigma)$, where σ is the cell of Ξ_k containing q.

Initialization. The above describes the data structure. We now initialize the data structure, and in particular, initialize the variables $\mathcal{L}(\cdot)$, $\lambda(\cdot)$, $minCost(\cdot)$ so that the algorithm invariants hold.

First of all, for each cell $\sigma \in \Xi$, we set $\mathcal{L}(\sigma) = \emptyset$ and $\lambda(\sigma) = 0$. For each point $q \in Q$, we set $\lambda(q) = 0$. Since $cost(q) = w(q)$ initially, it is not difficult to see that Algorithm Invariant 1 holds.

We next set $minCost(\sigma)$ for all cells of $\sigma \in \Xi$ in a bottom-up manner following the tree structure of Ξ. Specifically, for each cell σ in the last cutting Ξ_k, we set $minCost(\sigma) = \min_{q \in Q(\sigma)} w(q)$ by simply checking every point of $Q(\sigma)$. If $Q(\sigma) = \emptyset$, we set $minCost(\sigma) = \infty$. This establishes the second algorithm invariant for all cells $\sigma \in \Xi_k$. Then, we set $minCost(\sigma)$ for all cells of $\sigma \in \Xi_{k-1}$ with $minCost(\sigma) = \min_{\sigma' \in C(\sigma)}(minCost(\sigma') + \lambda(\sigma'))$, after which the second algorithm invariant holds for all cells $\sigma \in \Xi_{k-1}$. We continue this process to set $minCost(\sigma)$ for cells in $\Xi_{k-2}, \Xi_{k-3}, \ldots, \Xi_0$. After that, the second algorithm invariant is established for all cells $\sigma \in \Xi$.

In addition, for each cell σ in the last cutting Ξ_k, in order to efficiently update $minCost(\sigma)$ once $\lambda(q)$ changes for a point $q \in Q(\sigma)$, we construct a min-heap $\mathcal{H}(\sigma)$ on all points q of $Q(\sigma)$ with the values $w(q) + \lambda(q)$ as “keys”. Using the heap, if $\lambda(q)$ changes for a point $q \in Q(\sigma)$, $minCost(\sigma)$ can be updated in $O(\log m)$ time as $|Q(\sigma)| \leq m$.

This finishes our preprocessing step for Algorithm 2. The following lemma analyzes the time complexity of the preprocessing.

▶ **Lemma 4.** *The preprocessing takes $O(n \log n + nr + m \log r)$ time.*

Proof. First of all, sorting P takes $O(n \log n)$ time. Constructing the hierarchical cutting Ξ takes $O(nr)$ time. Computing the lists $L(d_i)$ for all disks $d_i \in D$ also takes $O(nr)$ time. The point location procedure for computing the subsets $Q(\sigma)$ for all cells $\sigma \in \Xi_k$ runs in $O(m \log r)$ time. For the initialization step, setting $\mathcal{L}(Q) = \emptyset$ and $\lambda(\sigma) = 0$ for all cells $\sigma \in \Xi$

takes $O(r^2)$ time as $|\Xi| = O(r^2)$. Since Ξ_k has $O(r^2)$ cells, computing $minCost(\sigma)$ for all cells $\sigma \in \Xi_k$ can be done in $O(r^2 + \sum_{\sigma \in \Xi_k} |Q(\sigma)|)$ time, which is $O(r^2 + m)$ by Observation 2. Initializing $minCost(\sigma)$ for all other cells $\sigma \in \Xi \setminus \Xi_k$ takes $O(r^2)$ time since each cell has $O(1)$ children (and thus computing $minCost(\sigma)$ for each such cell σ takes $O(1)$ time). Finally, constructing a heap $\mathcal{H}(\sigma)$ for all cells $\sigma \in \Xi_k$ takes $O(\sum_{\sigma \in \Xi_k} |Q(\sigma)|)$ time, which is $O(m)$ by Observation 2. Since $r \leq n$, $r^2 \leq nr$. Therefore, the total time of the preprocessing is $O(n \log n + nr + m \log r)$. ◀

4.2 The FindMinCost operation

We now discuss how to perform the FindMinCost operation.

Consider a disk d_i in FindMinCost operation of the i-th iteration of Algorithm 2. The goal is to compute $\min_{q \in Q_{d_i}} cost(q)$, i.e., the minimum cost of all points of Q inside the disk d_i.

Recall that $L(d_i)$ is the list of all cells $\sigma \in \Xi$ such that $d_i \in D_\sigma$. Define $L_1(d_i)$ to be the set of all cells of $L(d_i)$ that are from Ξ_k and let $L_2(d_i) = L(d_i) \setminus L_1(d_i)$. Define $L_3(d_i)$ as the set of cells $\sigma \in \Xi$ such that σ's parent is in $L_2(d_i)$ and σ is completely contained in d_i. We first have the following observation following the definition of the hierarchical cutting.

▶ **Observation 5.** *Q_{d_i} is the union of $\bigcup_{\sigma \in L_1(d_i)} (Q(\sigma) \cap d_i)$ and $\bigcup_{\sigma \in L_3(d_i)} Q(\sigma)$.*

Proof. Consider a point $q \in \bigcup_{\sigma \in L_1(d_i)} (Q(\sigma) \cap d_i)$. Suppose that q is in $Q(\sigma) \cap d_i$ for some cell $\sigma \in L_1(d_i)$. Then, since $q \in d_i$, it is obvious true that $q \in Q_{d_i}$.

Consider a point $q \in \bigcup_{\sigma \in L_3(d_i)} Q(\sigma)$. Suppose that $q \in Q(\sigma)$ for some cell $\sigma \in L_3(d_i)$. By the definition of $L_3(d)$, σ is fully contained in d_i. Therefore, $q \in d_i$ holds. Hence, $q \in Q_{d_i}$.

On the other hand, consider a point $q \in Q_{d_i}$. By definition, d_i contains q. Let σ be the cell of Ξ_k containing q. Since both d_i and σ contain q, $d_i \cap \sigma \neq \emptyset$. Therefore, either $\sigma \subseteq d_i$ or the boundary of d_i crosses σ. In the latter case, we have $\sigma \in L_1(d_i)$ and thus $q \in \bigcup_{\sigma \in L_1(d_i)} (Q(\sigma) \cap d_i)$. In the former case, σ must have two ancestors σ_1 and σ_2 such that (1) σ_1 is the parent of σ_2; (2) σ_2 is fully contained in d_i; (3) the boundary of d_i crosses σ_1. This is true because σ is fully contained in d_i while the boundary of d_i crosses the only cell of Ξ_0, which is the entire plane and is an ancestor of σ. As such, σ_1 must be in $L_2(d_i)$ and σ_2 must be in $L_3(d_i)$. Therefore, q must be in $\bigcup_{\sigma \in L_3(d_i)} Q(\sigma)$.

This proves the observation. ◀

With Observation 5, we now describe our algorithm for FindMinCost. Let α be a variable, which is initialized to ∞. At the end of the algorithm, we will have $\alpha = \min_{q \in Q_{d_i}} cost(q)$. For each cell σ in the list $L(d_i)$, if it is from $\sigma \in \Xi_k$, i.e., $\sigma \in L_1(d_i)$, then we process σ as follows. For each point $q \in Q(\sigma)$, by Algorithm Invariant 1, we have $cost(q) = w(q) + \lambda(q) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma')$. If $q \in d_i$, we compute $cost(q)$ by visiting all cells of $A(\sigma)$, which takes $O(\log r)$ time, and then we update $\alpha = \min\{\alpha, cost(q)\}$.

If $\sigma \in L_2(d_i)$, then we process it as follows. For each child σ' of σ that is fully contained in d_i (i.e., $\sigma \in L_3(d_i)$), we compute $\alpha_{\sigma'} = minCost(\sigma') + \sum_{\sigma'' \in A(\sigma')} \lambda(\sigma'')$ by visiting all cells of $A(\sigma')$, which takes $O(\log r)$ time. By Observation 3, we have $\alpha_{\sigma'} = \min_{q \in Q(\sigma)} cost(q)$. Then we update $\alpha = \min\{\alpha, \alpha_{\sigma'}\}$. After processing every cell $\sigma \in L(d_i)$ as above, we return α, which is equal to $\min_{q \in Q_{d_i}} cost(q)$ according to our algorithm invariants as well as Observation 5. This finishes the FindMinCost operation. The following lemma analyzes the runtime of the operation.

▶ **Lemma 6.** *The total time of the FindMinCost operations in the entire Algorithm 2 is bounded by $O((nr + mn/r) \log r)$.*

Proof. Recall that in each operation we processes cells of $L_1(d_i)$ and cells of $L_2(d_i)$ in different ways. The total time of the operation is the sum of the time for processing $L_1(d_i)$ and the time for processing $L_2(d_i)$.

For the time for processing $L_1(d_i)$, for each cell $\sigma \in L_1(d_i)$, for each point $q \in Q(\sigma) \cap d_i$, we spend $O(\log r)$ time computing $cost(q)$. Hence, the time for processing cells of $L_1(d_i)$ for each d_i is bounded by $O(\sum_{\sigma \in L_1(d_i)} |Q(\sigma)| \log r)$. The total time of processing $L_1(d_i)$ in the entire algorithm is on the order of $\sum_{i=1}^{n} \sum_{\sigma \in L_1(d_i)} |Q(\sigma)| \cdot \log r = \sum_{\sigma \in \Xi_k} (|D_\sigma| \cdot |Q(\sigma)|) \cdot \log r$. Recall that $|D_\sigma| \leq n/r$ for each cell $\sigma \in \Xi_k$. Hence, $\sum_{\sigma \in \Xi_k} (|D_\sigma| \cdot |Q(\sigma)|) \leq n/r \cdot \sum_{\sigma \in \Xi_k} |Q(\sigma)|$, which is $O(mn/r)$ by Observation 2. Therefore, the total time for processing cells of $L_1(d_i)$ in the entire Algorithm 2 is $O(mn/r \cdot \log r)$.

For the time for processing $L_2(d_i)$, for each cell $\sigma \in L_2(d_i)$, for each child σ' of σ, it takes $O(\log r)$ time to compute $\alpha_{\sigma'}$. Since σ has $O(1)$ cells, the total time for processing all cells of $L_2(d_i)$ is $O(|L_2(d_i)| \cdot \log r)$. The total time of processing $L_2(d_i)$ in the entire algorithm is on the order of $\sum_{i=1}^{n} |L_2(d_i)| \cdot \log r$. Note that $\sum_{i=1}^{n} |L_2(d_i)| \leq \sum_{i=1}^{n} |L(d_i)| = O(nr)$. Therefore, the total time for processing cells of $L_2(d_i)$ in the entire Algorithm 2 is $O(nr \log r)$.

Summing up the time for processing $L_1(d_i)$ and $L_2(d_i)$ leads to the lemma. ◀

4.3 The ResetCost operation

We now discuss the ResetCost operation. Consider the ResetCost operation in the i-th iteration of Algorithm 2. The goal is to reset $cost(q) = w(q) + \delta_i$ for all points $q \in Q$ that are outside the disk d_i. To this end, we will update our data structure, and more specifically, update the $\lambda(\cdot)$ and $minCost(\cdot)$ values for certain cells of Ξ and points of Q so that the algorithm invariants still hold.

Define $L_4(d_i)$ as the set of cells $\sigma \in \Xi$ such that σ's parent is in $L_2(d_i)$ and σ is completely outside d_i. Let $\overline{d_i}$ denote the region of the plane outside the disk d_i. We have the following observation, which is analogous to Observation 5.

▶ **Observation 7.** *$\overline{Q_{d_i}}$ is the union of $\bigcup_{\sigma \in L_1(d_i)} (Q(\sigma) \cap \overline{d_i})$ and $\bigcup_{\sigma \in L_4(d_i)} Q(\sigma)$.*

Proof. The proof is the same as that of Observation 5 except that we use $\overline{d_i}$ to replace d_i and use $L_4(d_i)$ to replace $L_3(d_i)$. We omit the details. ◀

Our algorithm for ResetCost works as follows. Consider a cell $\sigma \in L(d_i)$. As for the FindMinCost operation, depending on whether σ is from $L_1(d_i)$ or $L_2(d_i)$, we process it in different ways.

If σ is from $L_1(d_i)$, we process σ as follows. For each point $q \in Q(\sigma)$, if $q \in \overline{d_j}$, then we are supposed to reset $cost(q)$ to $w(q) + \delta_i$. To achieve the effect and also maintain the algorithm invariants, we do the following. First, we set $\lambda(q) = \delta_i - \sum_{\sigma' \in A(\sigma)} \lambda(\sigma')$, which can be done in $O(\log r)$ time by visiting the ancestors of σ. As such, we have $w(q) + \lambda(q) + \sum_{\sigma' \in A(\sigma)} \lambda(\sigma') = w(q) + \delta_i$, which establishes the first algorithm invariant for q. For the second algorithm invariant, we first update $minCost(\sigma)$ using the heap $\mathcal{H}(\sigma)$, i.e., by updating the key of q to the new value $w(q) + \lambda(q)$. The heap operation takes $O(\log m)$ time. Next, we update $minCost(\sigma')$ for all ancestors σ' of σ in a bottom-up manner using the formula $minCost(\sigma') = \min_{\sigma'' \in C(\sigma')}(minCost(\sigma'') + \lambda(\sigma''))$. Since each cell has $O(1)$ children, updating all ancestors of σ takes $O(\log r)$ time. This establishes the second algorithm invariant. Finally, since $\lambda(q)$ has just been changed, if $\lambda(q) \neq 0$, then we add q to the list $\mathcal{L}(\sigma')$ for all cells $\sigma' \in A(\sigma)$. Note that for each such $\mathcal{L}(\sigma')$ it is possible that q was already in the list before; but we do not check this and simply add q to the end of the list (and thus the list may contain multiple copies of q). This finishes the processing of q, which takes $O(\log r + \log m)$ time. Processing all points of $q \in Q(\sigma)$ as above takes $O(|Q(\sigma)| \cdot (\log r + \log m))$ time.

If σ is from $L_2(d_i)$, then we process σ as follows. For each child σ' of σ, if σ' is completely outside d_i, then we process σ' as follows. We are supposed to reset $cost(q)$ to $w(q) + \delta_i$ for all points $q \in Q(\sigma')$. In other words, the first algorithm invariant does not hold any more and we need to update our data structure to restore it. Note that the second algorithm invariant still holds. To achieve the effect and also maintain the algorithm invariants, we do the following. For each element e in the list $\mathcal{L}(\sigma')$ (recall that e is either a cell of $B(\sigma')$ or a point of $Q(\sigma')$), we process e as follows. First, we remove e from $\mathcal{L}(\sigma')$. Then we reset $\lambda(e) = 0$. If e is a point of $Q(\sigma')$, then let σ_e be the cell of Ξ_k that contains e; otherwise, e is a cell of $B(\sigma')$ and let σ_e be the parent of e. Since $\lambda(e)$ is changed, we update $minCost(\sigma'')$ for all cells $\sigma'' \in A(\sigma_e)$ in the same way as above in the first case for processing $L_1(d_i)$, which takes $O(\log r + \log m)$ time. This finishes processing e, after which the second algorithm invariant still holds. After all elements of $\mathcal{L}(\sigma')$ are processed as above, $\mathcal{L}(\sigma')$ becomes $\emptyset$ and we reset $\lambda(\sigma') = \delta_i - \sum_{\sigma'' \in A(\sigma') \setminus \{\sigma'\}} \lambda(\sigma'')$. Since $\lambda(\sigma')$ has been changed, we update $minCost(\sigma'')$ for all cells $\sigma'' \in A(\sigma)$ in the same way as before (which takes $O(\log r)$ time), after which the second algorithm invariant still holds. In addition, if $\lambda(\sigma') \neq 0$, then we add σ' to the list $\mathcal{L}(\sigma'')$ for all cells $\sigma'' \in A(\sigma)$, which again takes $O(\log r)$ time. This finishes processing σ', which takes $O(|\mathcal{L}(\sigma')| \cdot (\log r + \log m))$ time. It remains to restore the first algorithm invariant, for which we have the following observation.

▶ **Observation 8.** *After σ' is processed, the first algorithm invariant is established for all points $q \in Q(\sigma')$.*

Proof. Consider a point $q \in Q(\sigma')$. It suffices to show $w(q) + \delta_i = w(q) + \lambda(q) + \sum_{\sigma'' \in A(\sigma_q)} \lambda(\sigma'')$, where σ_q is the cell of Ξ_k that contains q. After the elements of the list $\mathcal{L}(\sigma')$ are processed as above, we have $\lambda(q) = 0$ for all points $q \in Q(\sigma')$ and $\lambda(\sigma'') = 0$ for all descendants σ'' of σ'. Therefore, $w(q) + \lambda(q) + \sum_{\sigma'' \in A(\sigma_1)} \lambda(\sigma_1) = w(q) + \sum_{\sigma'' \in A(\sigma')} \lambda(\sigma'') = w(q) + \lambda(\sigma') + \sum_{\sigma'' \in A(\sigma') \setminus \{\sigma'\}} \lambda(\sigma'')$. Recall that $\lambda(\sigma') = \delta_i - \sum_{\sigma'' \in A(\sigma') \setminus \{\sigma'\}} \lambda(\sigma'')$. We thus obtain $w(q) + \lambda(q) + \sum_{\sigma'' \in A(\sigma_q)} \lambda(\sigma'') = w(q) + \delta_i$. ◀

This finishes the ResetCost operation. According to Observation 7, $cost(q)$ has been reset for all points $q \in Q$ that are outside d_i. The following lemma analyzes the runtime of the operation.

▶ **Lemma 9.** *The total time of the ResetCost operations in the entire Algorithm 2 is bounded by $O((nr + mn/r) \cdot \log r \cdot (\log r + \log m))$.*

Proof. Recall that we processes cells of $L_1(d_i)$ and cells of $L_2(d_i)$ in different ways. The total time of the operation is the sum of the time for processing $L_1(d_i)$ and the time for processing $L_2(d_i)$.

For the time for processing $L_1(d_i)$, for each cell $\sigma \in L_1(d_i)$, recall that processing all points of $Q(\sigma)$ takes $O(|Q(\sigma)| \cdot (\log r + \log m))$ time. Hence, the total time for processing cells of $L_1(d_i)$ is on the order of $\sum_{\sigma \in L_1(d_i)} |Q(\sigma)| \cdot (\log r + \log m)$. The total time for processing cells of $L_1(d_i)$ in the entire Algorithm 2 is thus on the order of $\sum_{i=1}^{n} \sum_{\sigma \in L_1(d_i)} |Q(\sigma)| \cdot (\log r + \log m)$. As analyzed in the proof of Lemma 6, $\sum_{i=1}^{n} \sum_{\sigma \in L_1(d_i)} |Q(\sigma)| = O(mn/r)$. Therefore, the total time for processing cells of $L_1(d_i)$ in the entire Algorithm 2 is $O(mn/r \cdot (\log r + \log m))$.

For the time for processing $L_2(d_i)$, for each cell $\sigma \in L_2(d_i)$, for each child σ' of σ, processing σ' takes $O(|\mathcal{L}(\sigma')| \cdot (\log r + \log m))$ time. Next, we give an upper bound for $|\mathcal{L}(\sigma')|$ for all such cells σ' in the entire algorithm. Recall that each element e of $\mathcal{L}(\sigma')$ is either a point $q \in Q(\sigma')$ or a descendant cell $\sigma'' \in B(\sigma')$. Let $\mathcal{L}_1(\sigma')$ denote the subset of elements of $\mathcal{L}(\sigma')$ in the former case and $\mathcal{L}_2(\sigma')$ the subset of elements in the latter case. In the following we provide an upper bound for each subset.

1. For $\mathcal{L}_1(\sigma')$, notice that a point q is in the list only if σ_q is crossed by the lower arc of a disk d_i, where σ_q is the cell of Ξ_k containing q. If q is outside d_i, then a copy of q will be added to $\mathcal{L}(\sigma'')$ for all $O(\log r)$ cells σ'' of $A(\sigma_q)$. As $|D_{\sigma_q}| \leq n/r$, the number of elements in $\mathcal{L}_1(\sigma')$ contributed by the points of $Q(\sigma_q)$ for all such cells σ' in the entire algorithm is bounded by $O(|Q(\sigma_q)| \cdot n/r \cdot \log r)$. In light of Observation 2, the total size of $\mathcal{L}_1(\sigma')$ of all such cells σ' in the entire Algorithm 2 is $O(mn/r \cdot \log r)$.
2. For $\mathcal{L}_2(\sigma')$, observe that a cell σ_1 is in the list only if σ_2 is crossed by the lower arc of a disk d_i, where σ_2 is the parent of σ_1. If σ_1 is completely outside d_i, then a copy of σ_1 is added to $\mathcal{L}(\sigma'')$ for all $O(\log r)$ cells σ'' of $A(\sigma_2)$. As such, the number of elements in $\mathcal{L}_2(\sigma')$ for all such cells σ' in the entire algorithm contributed by each cell $\sigma_1 \in \Xi$ is bounded by $O(|D_{\sigma_2}| \cdot \log r)$. Since every cell of Ξ has $O(1)$ children and $\sum_{\sigma_2 \in \Xi} |D_{\sigma_2}| = O(nr)$, the total size of $\mathcal{L}_2(\sigma')$ of all such cells σ' in the entire Algorithm 2 is $O(nr \cdot \log r)$.

Therefore, the total time for processing cells of $L_2(d_i)$ in the entire Algorithm 2 is $O((nr + mn/r) \cdot \log r \cdot (\log r + \log m))$.

Summing up the time for processing $L_1(d_i)$ and $L_2(d_i)$ leads to the lemma. ◀

4.4 Putting it all together

We summarize the time complexity of the overall algorithm. By Lemma 4, the preprocessing step takes $O(n \log n + nr + m \log r)$ time. By Lemma 6, the total time for performing the FindMinCost operations in the entire algorithm is $O((nr + mn/r) \cdot \log r)$. By Lemma 9, the total time for performing the ResetCost operations in the entire algorithm is $O((nr + mn/r) \cdot \log r \cdot (\log m + \log r))$. Therefore, the total time of the overall algorithm is $O(n \log n + m \log r + (nr + mn/r) \cdot \log r \cdot (\log m + \log r))$. Recall that $1 \leq r \leq n$. Setting $r = \min\{\sqrt{m}, n\}$ gives the upper bound $O(n\sqrt{m} \log^2 m + (n + m) \log(n + m))$ for the time complexity of the overall algorithm.

Note that we have assumed that each point of P is covered by at least one disk of S. In this is not the case, then no feasible subset exists (alternatively, one may consider the optimal objective value ∞); if we run our algorithm in this case, then one can check that the value δ_n returned by our algorithm is ∞. Hence, our algorithm can automatically determine whether a feasible subset exists.[2]

▶ **Theorem 10.** *Given a set of n points and a set of m weighted unit disks in the plane such that the points and the disk centers are separated by a line, there is an $O(n\sqrt{m} \log^2 m + (n + m) \log(n + m))$ time algorithm to compute a subset of disks of minimum total weight whose union covers all points.*

References

1 Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. *Discrete and Computational Geometry*, 63:460–482, 2020. doi:10.1007/s00454-019-00099-6.

2 Helmut Alt, Esther M. Arkin, Hervé Brönnimann, Jeff Erickson, Sándor P. Fekete, Christian Knauer, Jonathan Lenchner, Joseph S. B. Mitchell, and Kim Whittlesey. Minimum-cost coverage of point sets by disks. In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SoCG)*, pages 449–458, 2006. doi:10.1145/1137856.1137922.

[2] It is possible to determine whether a feasible subset exists in $O(n \log n)$ time. For example, one can first compute the upper envelope of the boundary portions of all disks above ℓ. Then, it suffices to determine whether every point of P is below the upper envelope. Note that the upper envelope is x-monotone.

3 Christoph Ambühl, Thomas Erlebach, Matúš Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), and the 10th International Conference on Randomization and Computation (RANDOM)*, pages 3–14, 2006. doi:10.1007/11830924_3.

4 Michael Ben-Or. Lower bounds for algebraic computation trees (preliminary report). In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86, 1983. doi:10.1145/800061.808735.

5 Vittorio Bilò, Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 460–471, 2005. doi:10.1007/11561071_42.

6 Ahmad Biniaz, Prosenjit Bose, Paz Carmi, Anil Maheshwari, J. Ian Munro, and Michiel Smid. Faster algorithms for some optimization problems on collinear points. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 8:1–8:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.8.

7 Norbert Bus, Nabil H. Mustafa, and Saurabh Ray. Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25–32, 2018. doi:10.1016/j.dam.2017.12.018.

8 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry: Theory and Applications*, 47:112–124, 2014. doi:10.1016/j.comgeo.2012.04.001.

9 Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. In *Proceedings of 36th International Symposium on Computational Geometry (SoCG)*, pages 27:1–27:14, 2020. doi:10.4230/LIPIcs.SoCG.2020.27.

10 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.

11 Francisco Claude, Gautam K. Das, Reza Dorrigiv, Stephane Durocher, Robert Fraser, Alejandro López-Ortiz, Bradford G. Nickerson, and Alejandro Salinger. An improved line-separable algorithm for discrete unit disk cover. *Discrete Mathematics, Algorithms and Applications*, 2:77–88, 2010. doi:10.1142/S1793830910000486.

12 Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 434–444, 1988. doi:10.1145/62212.62255.

13 Shashidhara K. Ganjugunte. *Geometric hitting sets and their variants*. PhD thesis, Duke University, 2011. URL: https://dukespace.lib.duke.edu/items/391e2278-74f0-408f-9be7-4c97cf72e352.

14 Sariel Har-Peled and Mira Lee. Weighted geometric set cover problems revisited. *Journal of Computational Geometry*, 3:65–85, 2012. doi:10.20382/jocg.v3i1a4.

15 Arindam Karmakar, Sandip Das, Subhas C. Nandy, and Binay K. Bhattacharya. Some variations on constrained minimum enclosing circle problem. *Journal of Combinatorial Optimization*, 25(2):176–190, 2013. doi:10.1007/s10878-012-9452-4.

16 Nissan Lev-Tov and David Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47:489–501, 2005. doi:10.1016/j.comnet.2004.08.012.

17 Jian Li and Yifei Jin. A PTAS for the weighted unit disk cover problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 898–909, 2015. doi:10.1007/978-3-662-47672-7_73.

18 Gang Liu and Haitao Wang. Geometric hitting set for line-constrained disks. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS)*, pages 574–587, 2023. doi:10.1007/978-3-031-38906-1_38.

19 Gang Liu and Haitao Wang. On the line-separable unit-disk coverage and related problems. In *Proceedings of the 34th International Symposium on Algorithms and Computation (ISAAC)*, pages 51:1–51:14, 2023. Full version available at https://arxiv.org/abs/2309.03162.

20 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895, 2010. doi:10.1007/s00454-010-9285-9.

21 Logan Pedersen and Haitao Wang. On the coverage of points in the plane by disks centered at a line. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG)*, pages 158–164, 2018. URL: https://home.cs.umanitoba.ca/~cccg2018/papers/session4A-p1.pdf.

22 Logan Pedersen and Haitao Wang. Algorithms for the line-constrained disk coverage and related problems. *Computational Geometry: Theory and Applications*, 105-106:101883:1–18, 2022. doi:10.1016/j.comgeo.2022.101883.

23 Haitao Wang. Unit-disk range searching and applications. *Journal of Computational Geometry*, 14:343–394, 2023. doi:10.20382/jocg.v14i1a13.

24 Haitao Wang and Jie Xue. Algorithms for halfplane coverage and related problems. In *Proceedings of the 40th International Symposium on Computational Geometry (SoCG)*, pages 79:1–79:15, 2024. doi:10.4230/LIPIcs.SoCG.2024.79.

25 Haitao Wang and Jingru Zhang. Line-constrained k-median, k-means, and k-center problems in the plane. *International Journal of Computational Geometry and Application*, 26:185–210, 2016. doi:10.1142/S0218195916600049.

Streaming in Graph Products

Markus Lohrey
Universität Siegen, Germany

Julio Xochitemol
Universität Siegen, Germany

Abstract

We investigate the streaming space complexity of word problems for groups. Using so-called distinguishers, we prove a transfer theorem for graph products of groups. Moreover, we use distinguishers to obtain a logspace streaming algorithm for the membership problem in a finitely generated subgroup of a free group.

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

Keywords and phrases word problems for groups, streaming algorithms, graph products

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.71

Related Version *Full Version*: `https://arxiv.org/abs/2202.04060` [17]

Funding *Julio Xochitemol*: Funded by DFG project LO 748/13-2.

1 Introduction

The word problem for a fixed finitely generated (f.g. for short) group G is the following computational problem: Fix a finite set of generators Σ for G, which means that every element of G can be written as a finite product of elements from Σ. The input for the word problem is a finite word $a_1 a_2 \cdots a_n$ over the alphabet Σ and the question is whether this word evaluates to the group identity of G. The word problem was introduced by Dehn in 1911 [5]. It is arguably the most important computational problem in group theory and has been studied by group theorists as well as computer scientists. In general, the word problem is undecidable [4, 20], but for many classes of groups (e.g. linear groups, metabelian groups, hyperbolic groups) efficient algorithms exist; see e.g. [15] for an overview.

In [16, 17] we started to investigate streaming algorithms for the word problem of a group G. The input stream is a word $w \in \Sigma^*$ over the generators and the algorithm has to decide whether $w = 1$ holds in the group G. This can be viewed as a streaming algorithm for the formal language $\{w \in \Sigma^* : w = 1 \text{ in } G\}$. Streaming algorithms for formal languages (mainly subclasses of context-free languages) have been studied in [1, 2, 7, 18]. In [16, 17] we consider the space complexity of streaming algorithms. For deterministic streaming algorithms it turnes out that the space complexity of the word problem for a f.g. group G is tightly related to the growth of G: if $\gamma(n)$ is the growth function of G (which is the number of different group elements that can be represented by words of length at most n over the generating set), then the space complexity of the best deterministic streaming algorithm for the word problem of G is roughly $\log \gamma(n/2)$. This result basically reduces the study of deterministic streaming algorithms for word problems to the study of growth in groups, which is an important research area in geometric group theory with many deep results.

In [16, 17] we therefore mainly focus on randomized streaming algorithms for word problems. For this it turns out to be useful to consider so called distinguishers for groups. Roughly speaking, a distinguisher for a f.g. group G with finite generating set Σ is a randomized streaming algorithm $\mathcal{A}$ such that for all words $u, v \in \Sigma^*$ of length at most n we have that: (i) if u and v evaluate to the same element of G then with high probability, u and

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 71; pp. 71:1–71:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

v lead to the same memory state of $\mathcal{A}$, and (ii) if u and v evaluate to different elements of G then with high probability, u and v lead to different memory states of $\mathcal{A}$; see Section 4. It is easy to obtain from a distinguisher $\mathcal{R}$ for the group G a randomized streaming algorithm $\mathcal{S}$ for the word problem of G (with low error probability). Moreover, the space complexity of $\mathcal{S}$ is only twice the space complexity of $\mathcal{R}$; see Lemma 3.

We showed in [16, 17] that for many important f.g. groups G there exist logspace distinguishers with error probability $1/n^c$ for any constant $c > 1$, where n is the input length. This is in particular the case for f.g. linear groups (matrix groups). In general, the growth of a linear group can be $2^{\Theta(n)}$ (take for instance a free group of rank 2), and therefore its deterministic streaming space complexity can be $\Theta(n)$, which is the worst case (the streaming algorithms can always store the whole prefix of the input stream). We proved in [17] also the following transfer theorem for wreath products: If G is a f.g. group having a distinguisher with space complexity $s(n)$ and error probability $\epsilon(n)$ and A is a f.g. abelian group then there is a distinguisher for the wreath product $A \wr G$ having space complexity $\mathcal{O}(s(n) + \log n)$ and error probability roughly $n^2\epsilon(n)$. Interestingly, if H is any non-abelian group then any randomized streaming algorithm with error at most $1/2 - \epsilon$ for the word problem of $H \wr G$ must have the worst-case space complexity $\Theta(n)$; see [16, Theorem 21].

The first main result of the paper states a similar transfer theorem for *graph products*. This is an important construction in group theory that generalizes the direct product as well as the free product. It can be seen as a partially commutative version of the free product, where some of the factors G_i in a free product $G_1 * G_2 * \cdots * G_k$ are allowed to commute. Which of G_i commute is specified by a graph on the index set $\{1, \dots, k\}$. We show that if every group G_i ($1 \leq i \leq k$) has a distinguisher with space complexity at most $s(n)$ and error probability $\epsilon(n)$, then every graph product of the groups G_i has a distinguisher with space complexity $\mathcal{O}(s(n) + \log n)$ and error probability roughly $n^2\epsilon(n)$ (Theorem 9). As a corollary we obtain for instance a randomized streaming algorithm with logarithmic space complexity for the word problem of a graph product of linear groups. Theorem 9 is similar to the following result from [6]: If the word problem for every group G_i can be solved in deterministic logspace on a Turing machine then the same is true for every graph product of the G_i. Kausch in his thesis [13] strengthened this result by showing that the word problem of the graph product is AC_0-Turing-reducible to the word problems of the G_i ($1 \leq i \leq k$) and the free group of rank two.

Our second main contribution deals with randomized streaming algorithms for subgroup membership problems. In a subgroup membership problem one has a subgroup H of a f.g. group G. For an input word $w \in \Sigma^*$ (Σ is again a finite set of generators for G) one has to determine whether w represents an element of H. The word problem is the special case where $H = 1$. We present a randomized streaming algorithm with logarithmic space complexity for the case where G is a f.g. free group and H is a f.g. subgroup of G (Theorem 14). Moreover, we show that this result extends neither to the case where H is not finitely generated (Theorem 15) nor the case where H is a finitely generated subgroup of a direct product of two free groups of rank two (Theorem 16).

2 Preliminaries

For integers $a < b$ let $[a, b]$ be the integer interval $\{a, a + 1, \dots, b\}$. We write $[0, 1]_{\mathbb{R}}$ for the set $\{r \in \mathbb{R} \colon 0 \leq r \leq 1\}$ of all probabilities.

Let Σ be a finite alphabet. As usual we write Σ^* for the set of all finite words over the alphabet w. The empty word is denoted with ε. For a word $w = a_1 a_2 \cdots a_n$ ($a_1, a_2, \dots, a_n \in \Sigma$) let $|w| = n$ be its length and $w[i] = a_i$ (for $1 \leq i \leq n$) the symbol at position i.

A prefix of a word w is a word u such that $w = uv$ for some word v. We denote with $\mathcal{P}(w)$ the set of all prefixes of w. Let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ be the set of non-empty words and $\Sigma^{\leq n} = \{w \in \Sigma^* \colon |w| \leq n\}$ be the set of all words of length at most n. For a subalphabet $\Theta \subseteq \Sigma$ we denote with $\pi_\Theta : \Sigma^* \to \Theta^*$ the projection homomorphism that deletes all symbols from $\Sigma \setminus \Theta$ in a word: $\pi_\Theta(a) = a$ for $a \in \Theta$ and $\pi_\Theta(a) = \varepsilon$ for $a \in \Sigma \setminus \Theta$.

2.1 Sequential transducer

In Section 5 we make use of (left-)sequential transducers, see e.g. [3] for more details. A sequential transducer is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, q_0, \delta)$, where Q is a finite set of states, Σ is the input alphabet, Γ is the output alphabet, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to Q \times \Gamma^*$ is the transition function. The meaning of $\delta(q, a) = (p, u)$ is that if $\mathcal{T}$ is in state q and the next input symbol is a then it moves to state p and outputs the word u. We extend δ to a mapping $\delta : Q \times \Sigma^* \to Q \times \Gamma^*$ as follows, where $q \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$:

- $\delta(q, \varepsilon) = (q, \varepsilon)$ for all $q \in Q$, and
- if $\delta(q, a) = (p, u)$ and $\delta(p, w) = (r, v)$ then $\delta(q, aw) = (r, uv)$.

We define the function $f_\mathcal{T} : \Sigma^* \to \Gamma^*$ computed by $\mathcal{T}$ by $f_\mathcal{T}(w) = x$ if and only if $\delta(q_0, w) = (q, x)$ for some $q \in Q$. Intuitively, in order compute $f_\mathcal{T}(w)$, $\mathcal{T}$ reads the word w starting in the initial state q_0 and thereby concatenates all the outputs produced in the transitions.

2.2 Probabilistic finite automata

In the following we introduce probabilistic finite automata [21, 22] as a model for randomized streaming algorithms. A *probabilistic finite automaton* (PFA) $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ consists of a finite set of states Q, a finite alphabet Σ, an *initial state distribution* $\iota\colon Q \to [0,1]_\mathbb{R}$, a *transition probability function* $\rho\colon Q \times \Sigma \times Q \to [0,1]_\mathbb{R}$ and a set of final states $F \subseteq Q$ such that $\sum_{q \in Q} \iota(q) = 1$ and $\sum_{q \in Q} \rho(p, a, q) = 1$ for all $p \in Q$, $a \in \Sigma$. If ρ is required to map into $\{0, 1\}$, then $\mathcal{A}$ is a *semi-probabilitistic finite automaton* (semiPFA). This means that after choosing the initial state according to the distribution ι, $\mathcal{A}$ proceeds deterministically.

Let $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ be a PFA. For a random variable X with values from Q and $a \in \Sigma$ we define the random variable $X \cdot a$ (which also takes values from Q) by

$$\mathsf{Prob}[X \cdot a = q] = \sum_{p \in Q} \mathsf{Prob}[X = p] \cdot \rho(p, a, q).$$

For a word $w \in \Sigma^*$ we define a random variable $\mathcal{A}(w)$ with values from Q inductively as follows: the random variable $\mathcal{A}(\varepsilon)$ is defined such that $\mathsf{Prob}[\mathcal{A}(\varepsilon) = q] = \iota(q)$ for all $q \in Q$. Moreover, $\mathcal{A}(wa) = \mathcal{A}(w) \cdot a$ for all $w \in \Sigma^*$ and $a \in \Sigma$. Thus, $\mathsf{Prob}[\mathcal{A}(w) = q]$ is the probability that $\mathcal{A}$ is in state q after reading w. For a language $L \subseteq \Sigma^*$, the *error probability* of $\mathcal{A}$ on $w \in \Sigma^*$ for L is

$$\epsilon(\mathcal{A}, w, L) = \begin{cases} \mathsf{Prob}[\mathcal{A}(w) \notin F] & \text{if } w \in L, \\ \mathsf{Prob}[\mathcal{A}(w) \in F] & \text{if } w \notin L. \end{cases}$$

If $\mathcal{A}$ is a semiPFA then we can identify ρ with a mapping $\rho : Q \times \Sigma \to Q$, where $\rho(p, a)$ is the unique state q with $\rho(p, a, q) = 1$. This mapping ρ is extended to a mapping $\rho : Q \times \Sigma^* \to Q$ in the usual way: $\rho(p, \varepsilon) = p$ and $\rho(p, aw) = \rho(\rho(p, a), w)$. We then obtain

$$\mathsf{Prob}[\mathcal{A}(w) = q] = \sum \{\iota(p) : p \in Q, \rho(p, w) = q\}.$$

For a semiPFA $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ and a boolean condition $\mathcal{E} : Q \to \{0, 1\}$ we define the probability $\mathsf{Prob}_{q \in Q}[\mathcal{E}(q)] = \sum\{\iota(q) : q \in Q, \mathcal{E}(q) = 1\}$. SemiPFAs are needed in Section 4 for the notion of a distinguisher.

2.3 Randomized streaming algorithms

In this section we define our model of randomized streaming algorithms. It is a non-uniform model in the sense that for every input length n we have a separate algorithm that handles inputs of length at most n. Formally, a (non-uniform) *randomized streaming algorithm* is a sequence $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ of PFA $\mathcal{A}_n$ over the same alphabet Σ. If every $\mathcal{A}_n$ is a semiPFA, we speak of a *semi-randomized streaming algorithm.*

Let $\epsilon_0, \epsilon_1 : \mathbb{N} \to [0,1]_{\mathbb{R}}$ be monotonically decreasing functions. A randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ is (ϵ_0, ϵ_1)*-correct* for a language $L \subseteq \Sigma^*$ if for every large enough $n \geq 0$ and every word $w \in \Sigma^{\leq n}$ we have the following:

- if $w \in L$ then $\epsilon(\mathcal{A}_n, w, L) \leq \epsilon_1(n)$, and
- if $w \notin L$ then $\epsilon(\mathcal{A}_n, w, L) \leq \epsilon_0(n)$.

If $\epsilon_0 = \epsilon_1 =: \epsilon$ then we also say that $\mathcal{R}$ is ϵ-correct for L. We say that $\mathcal{R}$ is a randomized streaming algorithm for L if it is $1/3$-correct for L. The choice of $1/3$ for the error probability is not important. Using a standard application of the Chernoff bound, one can make the error probability an arbitrarily small constant; see [17, Theorem 4.1].

The *space complexity* of the randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ is the function $s(\mathcal{R}, n) = \lceil \log_2 |Q_n| \rceil$, where Q_n is the state set of $\mathcal{A}_n$. The motivation for this definition is that states of Q_n can be encoded by bit strings of length at most $\lceil \log_2 |Q_n| \rceil$. The *randomized streaming space complexity of the language* L is the smallest possible function $s(\mathcal{R}, n)$, where $\mathcal{R}$ is a randomized streaming algorithm for L. It is always bounded by $\mathcal{O}(n)$ since even a deterministic streaming algorithm can store the whole input word $w \in \Sigma^{\leq n}$ using $\mathcal{O}(n)$ bits.

As remarked before, our model of randomized streaming algorithms is non-uniform in the sense that for every input length n we have a separate streaming algorithm $\mathcal{A}_n$. This makes lower bounds of course stronger. On the other hand, the randomized streaming algorithms that we construct for concrete groups will be mostly uniform in the sense that there is an efficient algorithm that constructs from a given n the PFA $\mathcal{A}_n$. An exception is the following theorem (see [17, Theorem 4.3]), which uses non-uniformity in a crucial way.

▶ **Theorem 1.** *Let $\mathcal{R}$ be a randomized streaming algorithm such that $s(\mathcal{R}, n) \geq \Omega(\log n)$ and $\mathcal{R}$ is ϵ-correct for a language L. Then there exists a semi-randomized streaming algorithm $\mathcal{S}$ such that $s(\mathcal{S}, n) = \Theta(s(\mathcal{R}, n))$ and $\mathcal{S}$ is 2ϵ-correct for the language L.*

3 Groups and word problems

Let G be a group. The identity element will be always denoted with 1. For a subset $\Sigma \subseteq G$, we denote with $\langle \Sigma \rangle$ the subgroup of G generated by Σ. It is the set of all products of elements from $\Sigma \cup \Sigma^{-1}$. It can be also defined as the smallest (w.r.t. inclusion) subgroup of G that contains Σ. Similarly, the *normal closure* $N(\Sigma)$ of Σ is smallest normal subgroup of G that contains Σ. In this case, one gets the quotient group $G/N(\Sigma)$.

We only consider *finitely generated* (f.g.) groups G, for which there is a finite set $\Sigma \subseteq G$ with $G = \langle \Sigma \rangle$; such a set Σ is called a *finite generating set* for G. If $\Sigma = \Sigma^{-1}$ then we say that Σ is a *finite symmetric generating set* for G. In the following we assume that all finite generating sets are symmetric. Every word $w \in \Sigma^*$ evaluates to a group element $\pi_G(w)$ in the natural way; here $\pi_G : \Sigma^* \to G$ is the canonical morphism from the free monoid Σ^* to G that is the identity on Σ. Instead of $\pi_G(u) = \pi_G(v)$ we also write $u \equiv_G v$. Let $\mathsf{WP}(G, \Sigma) = \{w \in \Sigma^* \mid \pi_G(w) = 1\}$ be the *word problem for* G with respect to the generating set Σ.

We are interested in streaming algorithms for words problems $\mathsf{WP}(G, \Sigma)$. By the following lemma (see [17, Lemma 5.1]) the randomized streaming space complexity for a word problem only changes by a constant when the generating set is changed.

▶ **Lemma 2.** *Let Σ_1 and Σ_2 be finite symmetric generating sets for the group G and let $s_i(n)$ be the randomized streaming space complexity of $\mathsf{WP}(G, \Sigma_i)$. Then there exists a constant c that depends on G, Σ_1 and Σ_2 such that $s_1(n) \leq s_2(c \cdot n)$.*

4 Distinguishers for groups

Let G be a f.g. group G with the finite generating set Σ. Moreover, let $\epsilon_0, \epsilon_1 : \mathbb{N} \to [0,1]_{\mathbb{R}}$ be monotonically decreasing functions. A *semi-randomized* streaming algorithm $(\mathcal{A}_n)_{n\geq 0}$ with $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n, F_n)$ is called an (ϵ_0, ϵ_1)*-distinguisher* for G (with respect to Σ), if the following properties hold for all large enough $n \geq 0$ and all words $u, v \in \Sigma^{\leq n}$:

- If $u \equiv_G v$ then $\mathsf{Prob}_{q\in Q_n}[\rho_n(q,u) = \rho_n(q,v)] \geq 1 - \epsilon_1(n)$. In other words: for a randomly chosen initial state, the semiPFA $\mathcal{A}_n$ arrives with probability at least $1 - \epsilon_1(n)$ in the same state after reading u and v.
- If $u \not\equiv_G v$ then $\mathsf{Prob}_{q\in Q_n}[\rho_n(q,u) \neq \rho_n(q,v)] \geq 1 - \epsilon_0(n)$. In other words: for a randomly chosen initial state, the semiPFA $\mathcal{A}_n$ arrives with probability at least $1 - \epsilon_0(n)$ in different states after reading u and v.

Note that the set F_n of final states of $\mathcal{A}_n$ is not important for a distinguisher and we will just write $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n)$ in the following if we talk about an (ϵ_0, ϵ_1)-distinguisher $(\mathcal{A}_n)_{n\geq 0}$. The following result is shown in [17].

▶ **Lemma 3.** *Let $\mathcal{R}$ be an (ϵ_0, ϵ_1)-distinguisher for G with respect to Σ. Then $\mathsf{WP}(G, \Sigma)$ has an (ϵ_0, ϵ_1)-correct semi-randomized streaming algorithm with space complexity $2 \cdot s(\mathcal{R}, n)$.*

Due to Lemma 3, our goal in the rest of the paper will be the construction of space efficient (ϵ_0, ϵ_1)-distinguishers for groups. In the next section we will need some further observations on (ϵ_0, ϵ_1)-distinguishers that we introduce in the rest of this section.

For equivalence relations $\equiv_1$ and $\equiv_2$ on a set A and a subset $S \subseteq A$ we say that:

- $\equiv_1$ refines $\equiv_2$ on S if for all $a, b \in S$ we have: if $a \equiv_1 b$ then $a \equiv_2 b$;
- $\equiv_1$ equals $\equiv_2$ on S if for all $a, b \in S$ we have: $a \equiv_1 b$ if and only if $a \equiv_2 b$.

For a semiPFA $\mathcal{A} = (Q, \Sigma, \iota, \rho)$ and a state $q \in Q$ we define the equivalence relation $\equiv_{\mathcal{A},q}$ on Σ^* as follows: $u \equiv_{\mathcal{A},q} v$ if and only if $\rho(q,u) = \rho(q,v)$. Whenever $\mathcal{A}$ is clear from the context, we just write $\equiv_q$ instead of $\equiv_{\mathcal{A},q}$. The statements from the following lemma can be shown by straightforward applications of the union bound; see [17].

▶ **Lemma 4.** *Let $(\mathcal{A}_n)_{n\geq 0}$ be an (ϵ_0, ϵ_1)-distinguisher for the f.g. group G with respect to the generating set Σ. Let $\mathcal{A}_n = (Q_n, \Sigma, \iota, \rho)$. Consider a set $S \subseteq \Sigma^{\leq n}$. Then, the following statements hold, where $\equiv_q$ refers to $\mathcal{A}_n$:*

- $\mathsf{Prob}_{q\in Q_n}[\equiv_G$ *equals* $\equiv_q$ *on* $S] \geq 1 - \max\{\epsilon_0(n), \epsilon_1(n)\}\binom{|S|}{2}$,
- $\mathsf{Prob}_{q\in Q_n}[\equiv_G$ *refines* $\equiv_q$ *on* $S] \geq 1 - \epsilon_1(n)\binom{|S|}{2}$,
- $\mathsf{Prob}_{q\in Q_n}[\equiv_q$ *refines* $\equiv_G$ *on* $S] \geq 1 - \epsilon_0(n)\binom{|S|}{2}$.

The following two simple lemmas are needed in Section 5; their proofs can be found in [17]. Recall that for a word w we write $\mathcal{P}(w)$ for the set of all prefixes of w.

▶ **Lemma 5.** *Let G be a f.g. group with the finite generating set Σ and let $\mathcal{A} = (Q, \Sigma, \iota, \rho)$ be a semiPFA with $q \in Q$. Consider $u, v \in \Sigma^*$ such that $\equiv_G$ refines $\equiv_q$ on $\mathcal{P}(u) \cup \mathcal{P}(v)$ and let $u = xyz$ with $y \equiv_G 1$. Then $\equiv_G$ refines $\equiv_q$ on $\mathcal{P}(xz) \cup \mathcal{P}(v)$.*

▶ **Lemma 6.** *Let G, $\mathcal{A}$, and q be as in Lemma 5. Consider $u, v \in \Sigma^*$ such that $\equiv_q$ refines $\equiv_G$ on $\mathcal{P}(u) \cup \mathcal{P}(v)$ and let $u = xyz$ with $\rho(q, x) = \rho(q, xy)$. Then $\equiv_q$ refines $\equiv_G$ on $\mathcal{P}(xz) \cup \mathcal{P}(v)$.*

Finally we state the following result from [17, Theorem 9.1]. Recall that a linear group is a group of matrices over some field.

▶ **Theorem 7.** *For every f.g. linear group G and every $c > 0$ there exists a $(1/n^c, 0)$-distinguisher with space complexity $\mathcal{O}(\log n)$.*

Theorem 7 can serve as a basis for the construction of further distinguishers. Note that in the positive case (where $u \equiv_G v$) the error probability is zero.

5 Randomized streaming algorithms for graph products

In this section we investigate a common generalization of the free product and direct product, which is known as the graph product of groups.

Let us first define the free product of two groups G and H. Let $A = G \setminus \{1\}$ and $B = H \setminus \{1\}$. W.l.o.g. we assume that $A \cap B = \emptyset$. The *free product* $G * H$ consists of all alternating sequences $a_1 a_2 \cdots a_n$ where $n \geq 0$, $a_i \in A \cup B$ for all $i \in [1, n]$ and $a_i \in A \Leftrightarrow a_{i+1} \in B$ for all $i \in [1, n-1]$. The identity element is of course the empty sequence ε. The product $u \cdot v$ of two elements $u, v \in G * H$ is obtained by concatenating u and v and then making the obvious simplifications according to the multiplication tables of G and H. More precisely, let $u = a_n a_{n-1} \cdots a_1$ and $v = b_1 b_2 \cdots b_m$. If $n = 0$ then $u \cdot v = v$ and if $m = 0$ then $u \cdot v = u$. Now assume that $n > 0$ and $m > 0$. If $a_1 \in A \Leftrightarrow b_1 \in B$ then $u \cdot v = a_n a_{n-1} \cdots a_1 b_1 b_2 \cdots b_m$. Otherwise choose $k \geq 0$ maximal such that $a_i = b_i^{-1}$ (in either G or H) holds for all $i \in [1, k]$. If $k = n$ then $u \cdot v = b_{k+1} \cdots b_m$ and if $k = m$ then $u \cdot v = a_n \cdots a_{k+1}$. Finally, if $k < n$ and $k < m$ then $u \cdot v = a_n \cdots a_{k+2} (a_{k+1} \cdot b_{k+1}) b_{k+2} \cdots b_m$. Note that $a_{k+1} \cdot b_{k+1}$ is a nontrivial element (either in G or H) by the choice of k. The free product of several groups $G_1, \ldots, G_c$ can be simply defined as $\ast_{i \in [1,c]} G_i = (\cdots((G_1 * G_2) * G_3) * \cdots * G_c)$.

A graph product is specified by a list of groups $G_1, \ldots, G_c$ and a symmetric and irreflexive relation $I \subseteq [1, c] \times [1, c]$. The corresponding *graph product* $G = \mathsf{GP}(G_1, \ldots, G_c, I)$ is the quotient $(\ast_{i \in [1,c]} G_i)/N$ of the free product $\ast_{i \in [1,c]} G_i$ modulo the normal closure N of all commutators $aba^{-1}b^{-1}$, where $a \in G_i$, $b \in G_j$ and $(i, j) \in I$. In other words, we take the free product $\ast_{i \in [1,c]} G_i$ but allow elements from groups G_i and G_j with $(i, j) \in I$ to commute. Graph products interpolate in a natural way between free products ($I = \emptyset$) and direct products ($I = \{(i, j) : i, j \in [1, c], i \neq j\}$). Graph products were introduced by Green in her thesis [8]. Graph products $\mathsf{GP}(G_1, \ldots, G_c, I)$, where every G_i is isomorphic to $\mathbb{Z}$, are also known as *graph groups* (or right-angled Artin groups). We will make use of the fact that every graph group is linear [11].

Let Σ_i be a finite symmetric generating set for G_i, where w.l.o.g. $1 \notin \Sigma_i$ and $\Sigma_i \cap \Sigma_j = \emptyset$ for $i \neq j$. Then, $\Sigma = \bigcup_{i=1}^{c} \Sigma_i$ generates G. For a word $u \in \Sigma^*$, the *block factorization* of u is the unique factorization $u = u_1 u_2 \cdots u_l$ such that $l \geq 0$, $u_1, \ldots, u_l \in \bigcup_{i \in [1,c]} \Sigma_i^+$ and $u_j u_{j+1} \notin \bigcup_{i \in [1,c]} \Sigma_i^+$ for all $j \in [1, l-1]$. The factors $u_1, u_2, \ldots, u_l$ are called the *blocks* of u.

We define several rewrite relations on words from Σ^* as follows: take $u, v \in \Sigma^*$ and let $u = u_1 u_2 \cdots u_l$ be the block factorization of u.

- We write $u \leftrightarrow_s v$ (s for swap) if there is $i \in [1, l-1]$ and $(j, k) \in I$ such that $u_i \in \Sigma_j^+$, $u_{i+1} \in \Sigma_k^+$ and $v = u_1 u_2 \cdots u_{i-1} u_{i+1} u_i u_{i+2} \cdots u_l$. In other words, we swap consecutive commuting blocks. Note that $\leftrightarrow_s$ is a symmetric relation.

- We write $u \to_d v$ (d for delete) if there is $i \in [1,l]$ and $j \in [1,c]$ such that $u_i \in \Sigma_j^+$, $u_i \equiv_{G_j} 1$ and $v = u_1 u_2 \cdots u_{i-1} u_{i+1} u_{i+2} \cdots u_l$. In other words, we delete a block that is trivial in its group.
- We write $u \leftrightarrow_r v$ (r for replace) if there is $i \in [1,l]$ and $j \in [1,c]$ such that $u_i, u_i' \in \Sigma_j^+$, $u_i \equiv_{G_j} u_i'$ and $v = u_1 u_2 \cdots u_{i-1} u_i' u_{i+1} u_{i+2} \cdots u_l$. In other words, we replace a block by an equivalent non-empty word. Note that $\leftrightarrow_r$ is a symmetric relation.

Clearly, in all three cases we have $u \equiv_G v$. If $u \to_d v$, then the number of blocks of v is smaller than the number of blocks of u and if $u \leftrightarrow_s v$ then the number of blocks of v can be smaller than the number of blocks of u (since two blocks can be merged into a single block). We write $u \leftrightarrow_{sr} v$ if $u \leftrightarrow_s v$ or $u \leftrightarrow_r v$ and we write $u \to_{sd} v$ if $u \leftrightarrow_s v$ or $u \to_d v$.

Let us say that a word $u \in \Sigma^*$ with l blocks is *reduced*, if there is no $v \in \Sigma^*$ such that $u \to_{sd}^* v$ and v has at most $l-1$ blocks. Clearly, for every word $u \in \Sigma^*$ there is a reduced word $u' \in \Sigma^*$ such that $u \to_{sd}^* u'$. The following result can be found in [8, Theorem 3.9] and [10] in slightly different notations.

▶ **Lemma 8.** *Let G be a graph product as above and $u, v \in \Sigma^*$. The following are equivalent:*

- $u \equiv_G v$
- *There are reduced words u', v' such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, and $u' \leftrightarrow_r^* v'$.*

Consider a word $u \in \Sigma^*$ and its block factorization $u = u_1 u_2 \ldots u_l$. A *pure prefix* of u is a word $u_{k_1} u_{k_2} \cdots u_{k_m}$ such that for some $i \in [1,c]$ we have

- $1 \le k_1 < k_2 < \cdots < k_m \le l$,
- $u_{k_1}, u_{k_2}, \ldots, u_{k_m} \in \Sigma_i^+$ and
- if $k_j < p < k_{j+1}$ for some $j \in [1, m-1]$ or $1 \le p < k_1$ then $u_p \notin \Sigma_i^+$.

▶ **Theorem 9.** *Let $G = \mathsf{GP}(G_1, \ldots, G_c, I)$ be a graph product as above and let $\mathcal{R}_i = (\mathcal{A}_{i,n})_{n \ge 0}$ be an (ϵ_0, ϵ_1)-distinguisher for G_i. Let $d \ge 1$ and define $\zeta_0(n) = 2\epsilon_0(n) c n^2 + 1/n^d$ and $\zeta_1(n) = 2\epsilon_1(n) c n^2$. Then, there exists a (ζ_0, ζ_1)-distinguisher for G with space complexity $\mathcal{O}(\sum_{i=1}^c s(\mathcal{R}_i, n) + \log n)$.*

Proof. Let us fix an input length n and let $\mathcal{A}_{i,n} = (Q_{i,n}, \Sigma_i, \iota_{i,n}, \rho_{i,n})$, where w.l.o.g. $Q_{i,n} = [0, |Q_{i,n}| - 1]$. To simplify the notation, we will omit the second subscript n in the following, i.e., we write $\mathcal{A}_i = (Q_i, \Sigma_i, \iota_i, \rho_i)$ with $Q_i = [0, |Q_i| - 1]$ for the semiPFA $\mathcal{A}_{i,n}$. For a state $q \in Q_i$, we will use the equivalence relation $\equiv_q \; = \; \equiv_{\mathcal{A}_i, q}$ defined in Section 4. For a word $w \in \Sigma^*$, we write $\pi_i(w)$ for the projection $\pi_{\Sigma_i}(w)$.

For every $i \in [1,c]$ we choose a new symbol a_i and consider the infinite cyclic group $\langle a_i \rangle \cong \mathbb{Z}$. Let $\Gamma = \{a_1, a_1^{-1}, \ldots, a_c, a_c^{-1}\}$ and consider the graph group $H = \mathsf{GP}(\langle a_1 \rangle, \ldots, \langle a_c \rangle, I)$. Since every graph group is linear, there is a $(1/m^d, 0)$-distinguisher $(\mathcal{B}_m)_{m \ge 0}$ with space complexity $\mathcal{O}(\log m)$ for H by Theorem 7. Let $\mathcal{B}_m = (R_m, \Gamma, \lambda_m, \sigma_m)$.

We build from the semiPFA $\mathcal{A}_i$ and a state $q \in Q_i$ a sequential transducer $\mathcal{T}_{i,q} = (Q_i, \Sigma_i, \{a_i, a_i^{-1}\}, q, \delta_i)$, where for all $a \in \Sigma_i$ and $p \in Q_i$ we define (recall that $Q_i \subseteq \mathbb{N}$):

$$\delta_i(p, a) = (\rho_i(p, a), a_i^{-p} a_i^{\rho_i(p,a)}).$$

Let $f_{i,q} := f_{\mathcal{T}_{i,q}} : \Sigma_i^* \to \{a_i, a_i^{-1}\}^*$ be the function computed by $\mathcal{T}_{i,q}$. For a tuple $\bar{q} = (q_1, \ldots, q_c) \in \prod_{i \in [1,c]} Q_i$ of states from the semiPFAs $\mathcal{A}_i$ we define the sequential transducer $\mathcal{T}_{\bar{q}}$ by taking the direct product of the $\mathcal{T}_{i,q_i}$ ($i \in [1,c]$). Formally, it is defined as follows:

$$\mathcal{T}_{\bar{q}} = (\prod_{i \in [1,c]} Q_i, \Sigma, \Gamma, \bar{q}, \delta)$$

Algorithm 1 (ζ_0,ζ_1)-distinguisher for $\mathsf{GP}(G_1,\dots,G_c,I)$.

```
global variables: $q_i \in Q_i$ for all $i \in [1,c]$, $r \in R_m$
initialization:
guess $q_i \in Q_i = [0, |Q_i|-1]$ according to the input distribution $\iota_i$ of $\mathcal{A}_i$ ;
guess $r \in R_m$ according to the input distribution $\lambda_m$ of $\mathcal{B}_m$ ;
next input letter: $a \in \Sigma$
let $i \in [1,c]$ such that $a \in \Sigma_i$ ;
$r := \sigma_m(r, a_i^{-q_i} a_i^{\rho_i(q_i,a)})$ ; $q_i := \rho_i(q_i,a)$ ;
```

where for every $i \in [1,c]$, $a \in \Sigma_i$, and $(p_1,\dots,p_c) \in \prod_{i\in[1,c]} Q_i$ we have

$$\delta((p_1,\dots,p_c),a) = \big((p_1,\dots,p_{i-1},\rho_i(p_i,a),p_{i+1},\dots,p_c), a_i^{-p_i} a_i^{\rho_i(p_i,a)}\big).$$

Let $f_{\bar q} := f_{\mathcal{T}_{\bar q}} : \Sigma^* \to \Gamma^*$ be the function computed by $\mathcal{T}_{\bar q}$. Moreover, define

$$m = 2 \cdot n \cdot \max\{|Q_i| \colon i \in [1,c]\} \le n \cdot 2^{1+\max\{s(\mathcal{R}_i,n)\colon i\in[1,c]\}}.$$

Note that $|f_{\bar q}(w)| \le m$ if $|w| \le n$.

Our randomized streaming algorithm for G and input length n uses the semiPFA $\mathcal{B}_m$. States of $\mathcal{B}_m$ can be stored with $\mathcal{O}(\log m) \le \mathcal{O}(\max\{s(\mathcal{R}_i,n)\colon i \in [1,c]\} + \log n)$ bits. Basically, for an input word $w \in \Sigma^{\le n}$ the algorithm simulates $\mathcal{A}_i$ ($i \in [1,c]$) on the projections $w_i = \pi_i(w)$ and feeds the word $f_{\bar q}(w)$ into the semiPFA $\mathcal{B}_m$. Here, the state tuple $\bar q$ is randomly guessed in the beginning according to the distributions ι_i. The complete streaming algorithm is Algorithm 1. It stores at most $\sum_{i=1}^{c} s(\mathcal{R}_i,n) + \mathcal{O}(\max\{s(\mathcal{R}_i,n)\colon i \in [1,c]\} + \log n)$ bits.

Before we analyze the error probability of the algorithm we need some preparations. For $i \in [1,c]$ and a word $w \in \Sigma^*$ let $\mathcal{P}_i(w) = \mathcal{P}(\pi_i(w))$ be the set of all prefixes of the projection $\pi_i(w)$. Assume that $y \in \Sigma_i^+$ is a block of w and write $w = xyz$. We then have $f_{\bar q} = f_{\bar q}(x) f_{\bar r}(y) f_{\bar s}(z)$, where $\delta(\bar q, x) = (\bar r, f_{\bar q}(x))$ and $\delta(\bar r, y) = (\bar s, f_{\bar r}(y))$. The word $f_{\bar r}(y)$ is also a block of $f_{\bar q}$ (for this it is important that the transducers $\mathcal{T}_{j,q}$ translate non-empty words into non-empty words). Since $y \in \Sigma_i^+$ we have $r_j = s_j$ for all $j \in [1,c] \setminus \{i\}$ and $f_{\bar r}(y) = f_{i,r_i}(y)$. In addition, the definition of the transducer $\mathcal{T}_{i,r_i}$ implies that $f_{\bar r}(y) \equiv_{\langle a_i \rangle} a_i^{-r_i} a_i^{s_i}$.

Consider now two input words $u, v \in \Sigma^{\le n}$ and let $\mathcal{S}_i = \mathcal{P}_i(u) \cup \mathcal{P}_i(v)$ for $i \in [1,c]$, so that $|\mathcal{S}_i| \le 2n$. By Lemma 4 we have for all $i \in [1,c]$:

- $\mathsf{Prob}_{q\in Q_i}[\equiv_q \text{ refines } \equiv_{G_i} \text{ on } \mathcal{S}_i] \ge 1 - \epsilon_0(n)\binom{|\mathcal{S}_i|}{2} \ge 1 - 2\epsilon_0(n)n^2$,
- $\mathsf{Prob}_{q\in Q_i}[\equiv_{G_i} \text{ refines } \equiv_q \text{ on } \mathcal{S}_i] \ge 1 - \epsilon_1(n)\binom{|\mathcal{S}_i|}{2} \ge 1 - 2\epsilon_1(n)n^2$.

Our error analysis of Algorithm 1 is based on the following two claims.

▷ Claim 10. Assume that $\bar q = (q_1,\dots,q_c)$ is such that $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ for every $i \in [1,c]$. If $u \to_{sd}^* u'$ and $v \to_{sd}^* v'$, then $f_{\bar q}(u) \to_{sd}^* f_{\bar q}(u')$, $f_{\bar q}(v) \to_{sd}^* f_{\bar q}(v')$ and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$.

Proof. It suffices to show the following: If $u \to_{sd} u'$ holds, then $f_{\bar q}(u) \to_{sd} f_{\bar q}(u')$ and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v)$ for every $i \in [1,c]$. From this (and the symmetric statement where $v \to_{sd} v'$ and $u = u'$) we obtain the general statement by induction on the number of $\to_{sd}$-steps. We distinguish two cases.

Case 1. $u \leftrightarrow_s u'$. We must have $u = xy_1y_2z$ and $u' = xy_2y_1z$ for blocks y_1, y_2 such that $y_1 \in \Sigma_i^+$, $y_2 \in \Sigma_j^+$ and $(i,j) \in I$ (in particular $i \neq j$). We obtain

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x) f_{\bar{p}}(y_1) f_{\bar{r}}(y_2) f_{\bar{s}}(z) \text{ and} \\ f_{\bar{q}}(u') &= f_{\bar{q}}(x) f_{\bar{p}}(y_2) f_{\bar{r}'}(y_1) f_{\bar{s}}(z), \end{aligned}$$

where $\delta(\bar{q}, x) = (\bar{p}, f_{\bar{q}}(x))$, $\delta(\bar{p}, y_1) = (\bar{r}, f_{\bar{p}}(y_1))$, $\delta(\bar{r}, y_2) = (\bar{s}, f_{\bar{r}}(y_2))$, $\delta(\bar{p}, y_2) = (\bar{r}', f_{\bar{p}}(y_2))$, and $\delta(\bar{r}', y_1) = (\bar{s}, f_{\bar{r}'}(y_1))$. If we write $\bar{p} = (p_1, \ldots, p_c)$, then there are states $r_i \in Q_i$ and $r_j \in Q_j$ such that

$$\begin{aligned} \bar{r} &= (p_1, \ldots, p_{i-1}, r_i, p_{i+1}, \ldots, p_c), && (1) \\ \bar{r}' &= (p_1, \ldots, p_{j-1}, r_j, p_{j+1}, \ldots, p_c), \text{ and} && (2) \\ \bar{s} &= (p_1, \ldots, p_{i-1}, r_i, p_{i+1}, \ldots, p_{j-1}, r_j, p_{j+1}, \ldots, p_c) && (3) \end{aligned}$$

(we assume w.l.o.g. that $i < j$). Moreover, $f_{\bar{p}}(y_1) = f_{i,p_i}(y_1) = f_{\bar{r}'}(y_1) \in \{a_i, a_i^{-1}\}^+$ and $f_{\bar{r}}(y_2) = f_{j,p_j}(y_2) = f_{\bar{p}}(y_2) \in \{a_j, a_j^{-1}\}^+$. Thus, we have

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x) f_{\bar{p}}(y_1) f_{\bar{r}}(y_2) f_{\bar{s}}(z) \\ &= f_{\bar{q}}(x) f_{i,p_i}(y_1) f_{j,p_j}(y_2) f_{\bar{s}}(z) \\ &\leftrightarrow_s f_{\bar{q}}(x) f_{j,p_j}(y_2) f_{i,p_i}(y_1) f_{\bar{s}}(z) \\ &= f_{\bar{q}}(x) f_{\bar{p}}(y_2) f_{\bar{r}'}(y_1) f_{\bar{s}}(z) \\ &= f_{\bar{q}}(u'). \end{aligned}$$

Moreover, since $\mathcal{P}_i(u') = \mathcal{P}_i(u)$ and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$, it follows that $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v)$ for all $i \in [1,c]$.

Case 2. $u \to_d u'$. Then we obtain a factorization $u = xyz$, where $y \in \Sigma_i^+$ is a block, $y \equiv_{G_i} 1$, and $u' = xz$. We obtain a factorization

$$f_{\bar{q}}(u) = f_{\bar{q}}(x) f_{\bar{r}}(y) f_{\bar{s}}(z),$$

where $\delta(\bar{q}, x) = (\bar{r}, f_{\bar{q}}(x))$ and $\delta(\bar{r}, y) = (\bar{s}, f_{\bar{r}}(y))$. The word $f_{\bar{r}}(y)$ is a block of $f_{\bar{q}}(u)$. For the projection $\pi_i(u)$ we have $\pi_i(u) = \pi_i(x) y \pi_i(z)$. Since $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ and $\pi_i(x) \equiv_{G_i} \pi_i(x)y$, we obtain $\pi_i(x) \equiv_{q_i} \pi_i(x)y$. Since r_i (resp., s_i) is the state reached from q_i by the automaton $\mathcal{A}_i$ after reading $\pi_i(x)$ (resp., $\pi_i(x)y$), we obtain $r_i = s_i$ and hence $\bar{r} = \bar{s}$. This implies

$$f_{\bar{r}}(y) \equiv_{\langle a_i \rangle} a_i^{-r_i} a_i^{s_i} \equiv_{\langle a_i \rangle} 1.$$

Moreover, we have

$$f_{\bar{q}}(u') = f_{\bar{q}}(xz) = f_{\bar{q}}(x) f_{\bar{r}}(z) = f_{\bar{q}}(x) f_{\bar{s}}(z).$$

We therefore get $f_{\bar{q}}(u) \to_d f_{\bar{q}}(u')$.

It remains to show that $\equiv_{G_j}$ refines $\equiv_{q_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v)$ for every $j \in [1,c]$. For $j \neq i$ this is clear since $\mathcal{P}_j(u') \cup \mathcal{P}_j(v) = \mathcal{S}_j$. For $j = i$ we can use Lemma 5 for the words $\pi_i(u) = \pi_i(x) y \pi_i(z)$ and $\pi_i(v)$. ◁

▷ Claim 11. Assume that $\bar{q} = (q_1, \ldots, q_c)$ is such that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ for every $i \in [1,c]$. If $f_{\bar{q}}(u) \to_{sd}^* \tilde{u}$ and $f_{\bar{q}}(v) \to_{sd}^* \tilde{v}$, then there are $u', v' \in \Sigma^*$ such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, $f_{\bar{q}}(u') = \tilde{u}$, $f_{\bar{q}}(v') = \tilde{v}$ and $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$.

Proof. The proof is very similar to the above proof of Claim 10. As in the above proof of Claim 10, it suffices to consider the case where $f_{\bar{q}}(u) \to_{sd} \tilde{u}$ and $\tilde{v} = f_{\bar{q}}(v)$.

Case 1. $f_{\bar{q}}(u) \leftrightarrow_s \tilde{u}$. Since the blocks of u are translated into the blocks of $f_{\bar{q}}(u)$ by the transducer $\mathcal{T}_{\bar{q}}$, we obtain a factorization $u = xy_1y_2z$ for blocks $y_1 \in \Sigma_i^+, y_2 \in \Sigma_j^+$ of u such that $(i,j) \in I$ (in particular $i \neq j$) and

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x)f_{\bar{p}}(y_1)f_{\bar{r}}(y_2)f_{\bar{s}}(z), \\ \tilde{u} &= f_{\bar{q}}(x)f_{\bar{r}}(y_2)f_{\bar{p}}(y_1)f_{\bar{s}}(z). \end{aligned}$$

Here, the state tuples $\bar{p} = (p_1, \ldots, p_c)$, $\bar{r}$, and $\bar{s}$ are as in the above proof of Claim 10, see in particular (1) and (3). We can then define the tuple $\bar{r}'$ as in (2) and get $f_{\bar{p}}(y_1) = f_{i,p_i}(y_1) = f_{\bar{r}'}(y_1) \in \{a_i, a_i^{-1}\}^+$ and $f_{\bar{r}}(y_2) = f_{j,p_j}(y_2) = f_{\bar{p}}(y_2) \in \{a_j, a_j^{-1}\}^+$. We thus have

$$\tilde{u} = f_{\bar{q}}(x)f_{\bar{r}}(y_2)f_{\bar{p}}(y_1)f_{\bar{s}}(z) = f_{\bar{q}}(x)f_{\bar{p}}(y_2)f_{\bar{r}'}(y_1)f_{\bar{s}}(z) = f_{\bar{q}}(xy_2y_1z).$$

Clearly, we also have $u = xy_1y_2z \rightarrow_s xy_2y_1z$. So, we can set $u' = xy_2y_1z$. Since $\mathcal{P}_i(u') = \mathcal{P}_i(u)$ for all $i \in [1,c]$, it follows that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v)$ for all $i \in [1,c]$.

Case 2. $f_{\bar{q}}(u) \rightarrow_d \tilde{u}$. Then we obtain a factorization $u = xyz$, where $y \in \Sigma_i^+$ is a block of u,

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x)f_{\bar{r}}(y)f_{\bar{s}}(z), \text{ and} \\ \tilde{u} &= f_{\bar{q}}(x)f_{\bar{s}}(z). \end{aligned}$$

The state tuples $\bar{r}$ and $\bar{s}$ are such that $\delta(\bar{q}, x) = (\bar{r}, f_{\bar{q}}(x))$ and $\delta(\bar{r}, y) = (\bar{s}, f_{\bar{r}}(y))$. Moreover, the word $f_{\bar{r}}(y)$ is a block of $f_{\bar{q}}(u)$ with

$$a_i^{-r_i}a_i^{s_i} \equiv_{\langle a_i \rangle} f_{\bar{r}}(y) \equiv_{\langle a_i \rangle} 1.$$

This implies that $r_i = s_i$ and hence $\bar{r} = \bar{s}$. We therefore have

$$\rho_i(q_i, \pi_i(x)) = r_i = s_i = \rho_i(q_i, \pi_i(x)y).$$

Since $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ and $\pi_i(x), \pi_i(x)y \in \mathcal{S}_i$, we get $\pi_i(x) \equiv_{G_i} \pi_i(x)y$, i.e., $y \equiv_{G_i} 1$. If we set $u' = xz$ we get $u \rightarrow_d u'$ and

$$\tilde{u} = f_{\bar{q}}(x)f_{\bar{s}}(z) = f_{\bar{q}}(x)f_{\bar{r}}(z) = f_{\bar{q}}(xz) = f_{\bar{q}}(u').$$

It remains to show that $\equiv_{q_j}$ refines $\equiv_{G_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v)$ for every $j \in [1,c]$. For $j \neq i$ this is clear since $\mathcal{P}_j(u') \cup \mathcal{P}_j(v) = \mathcal{S}_j$. For $j = i$ we can use Lemma 6 for the words $\pi_i(u) = \pi_i(x)y\pi_i(z)$ and $\pi_i(v)$. ◁

We now estimate the error for the input words u and v. There are two cases to consider:

Case 1. $u \equiv_G v$. We will show that Algorithm 1 reaches with probability at least $1 - 2\epsilon_1(n)cn^2$ the same state when running on u and v, respectively. For this, assume that the randomly selected initial states $q_i \in Q_i$ are such that $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$. This happens with probability at least $1 - 2\epsilon_1(n)cn^2$.

First note that $u \equiv_G v$ implies $\pi_i(u) \equiv_{G_i} \pi_i(v)$ for all $i \in [1,c]$. Since $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$, we obtain $\rho_i(q_i, \pi_i(u)) = \rho_i(q_i, \pi_i(v))$. It remains to show that after reading u and v, also the states of $\mathcal{B}_m$ are the same. For this we show that $f_{\bar{q}}(u) \equiv_H f_{\bar{q}}(v)$ in the graph group H.

From Lemma 8 it follows that there are reduced words $u', v' \in \Sigma^*$ such that $u \rightarrow_{sd}^* u'$, $v \rightarrow_{sd}^* v'$, and $u' \leftrightarrow_r^* v'$. Claim 10 implies $f_{\bar{q}}(u) \rightarrow_{sd}^* f_{\bar{q}}(u')$, $f_{\bar{q}}(v) \rightarrow_{sd}^* f_{\bar{q}}(v')$, and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$. Since $u' \leftrightarrow_r^* v'$ we can write the block factorizations of u' and v' as $u' = u_1u_2 \cdots u_l$ and $v' = v_1v_2 \cdots v_l$ with $u_i, v_i \in \Sigma_{j_i}^+$ for some $j_i \in [1,c]$ and $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$. The block factorizations of $f_{\bar{q}}(u')$ and $f_{\bar{q}}(v')$ can be written as $f_{\bar{q}}(u') = \tilde{u}_1\tilde{u}_2 \cdots \tilde{u}_l$ and $f_{\bar{q}}(v') = \tilde{v}_1\tilde{v}_2 \cdots \tilde{v}_l$ with $\tilde{u}_i, \tilde{v}_i \in \{a_{j_i}, a_{j_i}^{-1}\}^+$.

We claim that $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$, which implies $f_{\bar{q}}(u') \equiv_H f_{\bar{q}}(v')$. Since $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$, we get the following: if $u'' = u_{k_1} u_{k_2} \cdots u_{k_e} \in \Sigma_j^*$ is a pure prefix of u' for some $j \in [1,c]$ then $v'' = v_{k_1} v_{k_2} \cdots v_{k_e} \in \Sigma_j^*$ is a pure prefix of v' such that $u'' \equiv_{G_j} v''$. Since $\equiv_{G_j}$ refines $\equiv_{q_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v')$ and $u'', v'' \in \mathcal{P}_j(u') \cup \mathcal{P}_j(v')$, we obtain $\rho_j(q_j, u'') = \rho_j(q_j, v'')$. This implies $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$ and hence $f_{\bar{q}}(u') \equiv_H f_{\bar{q}}(v')$. From this, we finally get $f_{\bar{q}}(u) \equiv_H f_{\bar{q}}(u') \equiv_H f_{\bar{q}}(v') \equiv_H f_{\bar{q}}(v)$.

Recall that $f_{\bar{q}}(u)$ (resp., $f_{\bar{q}}(v)$ is the word fed into the semiPFA $\mathcal{B}_m$ on input u (resp., v). Since $(\mathcal{B}_n)_{n \geq 0}$ is a $(1/n^d, 0)$-distinguisher for H, it follows that $f_{\bar{q}}(u)$ and $f_{\bar{q}}(v)$ lead in $\mathcal{B}_m$ with probability one to the same state. Hence, Algorithm 1 reaches with probability at least $1 - 2\epsilon_1(n)cn^2$ the same state when running on u and v, respectively.

Case 2. $u \not\equiv_G v$. We will show that Algorithm 1 reaches with probability at least $1 - (2\epsilon_0(n)cn^2 + 1/n^d)$ different states when running on u and v, respectively. To show this, assume that the randomly selected initial states $q_i \in Q_i$ are such that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$. This happens with probability at least $1 - 2\epsilon_0(n)cn^2$.

We claim that $f_{\bar{q}}(u) \not\equiv_H f_{\bar{q}}(v)$. In order to get a contradiction, assume that $f_{\bar{q}}(u) \equiv_H f_{\bar{q}}(v)$. From Lemma 8 it follows that there are reduced words $\tilde{u}, \tilde{v} \in \Gamma^*$ such that $f_{\bar{q}}(u) \to_{sd}^* \tilde{u}$, $f_{\bar{q}}(v) \to_{sd} \tilde{v}$ and $\tilde{u} \leftrightarrow_r^* \tilde{v}$. By Claim 11 there are $u', v' \in \Sigma^*$ such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, $f_{\bar{q}}(u') = \tilde{u}$, $f_{\bar{q}}(v') = \tilde{v}$ and $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$.

Since $f_{\bar{q}}(u') \leftrightarrow_r^* f_{\bar{q}}(v')$ we can write the block factorizations of $f_{\bar{q}}(u')$ and $f_{\bar{q}}(v')$ as $f_{\bar{q}}(u') = \tilde{u}_1 \tilde{u}_2 \cdots \tilde{u}_l$ and $f_{\bar{q}}(v') = \tilde{v}_1 \tilde{v}_2 \cdots \tilde{v}_l$ with $\tilde{u}_i, \tilde{v}_i \in \{a_{j_i}, a_{j_i}^{-1}\}^+$ for some $j_i \in [1,c]$ and $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$. Clearly, the block factorizations of u' and v' can then be written as $u' = u_1 u_2 \cdots u_l$ and $v' = v_1 v_2 \cdots v_l$, where the block $u_i \in \Sigma_{j_i}^+$ (resp., $v_i \in \Sigma_{j_i}^+$) is translated into the block $\tilde{u}_i$ (resp., $\tilde{v}_i$) by the sequential transducer $\mathcal{T}_{\bar{q}}$.

We claim that $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$. Since $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$ we have the following: if $\tilde{u}'' = \tilde{u}_{k_1} \tilde{u}_{k_2} \cdots \tilde{u}_{k_e} \in \{a_j, a_j^{-1}\}^*$ is a pure prefix of $f_{\bar{q}}(u')$ for some $j \in [1,c]$ then $\tilde{v}'' = \tilde{v}_{k_1} \tilde{v}_{k_2} \cdots \tilde{v}_{k_e} \in \{a_j, a_j^{-1}\}^*$ is a pure prefix of $f_{\bar{q}}(v')$ and $\tilde{u}'' \equiv_{\langle a_j \rangle} \tilde{v}''$. Let $p_j = \rho_j(q_j, u_{k_1} u_{k_2} \cdots u_{k_e})$ and $r_j = \rho_j(q_j, v_{k_1} v_{k_2} \cdots v_{k_e})$. We therefore have

$$a_j^{-q_j} a_j^{p_j} \equiv_{\langle a_j \rangle} \tilde{u}'' \equiv_{\langle a_j \rangle} \tilde{v}'' \equiv_{\langle a_j \rangle} a_j^{-q_j} a_j^{r_j},$$

i.e., $p_j = r_j$. Since $\equiv_{q_j}$ refines $\equiv_{G_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v')$ and $u_{k_1} u_{k_2} \cdots u_{k_e}$ as well as $v_{k_1} v_{k_2} \cdots v_{k_e}$ belong to $\mathcal{P}_j(u') \cup \mathcal{P}_j(v')$, we obtain $u_{k_1} u_{k_2} \cdots u_{k_e} \equiv_{G_j} v_{k_1} v_{k_2} \cdots v_{k_e}$. This holds for all pure prefixes of u'. We therefore have $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$, which implies $u' \equiv_G v'$. Finally, we get $u \equiv_G u' \equiv_G v' \equiv_G v$, which is a contradiction. Hence, we must have $f_{\bar{q}}(u) \not\equiv_H f_{\bar{q}}(v)$.

Since the algorithm feeds $f_{\bar{q}}(u)$ (resp., $f_{\bar{q}}(v)$) into the semiPFA $\mathcal{B}_m$, the latter reaches different states with probability at least $1 - 1/m^d \geq 1 - 1/n^d$ (under the assumption that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$). Hence, the probability that Algorithm 1 reaches different states when running on u and v is at least $(1 - 2\epsilon_0(n)cn^2)(1 - 1/n^d) \geq 1 - (2\epsilon_0(n)cn^2 + 1/n^d)$. ◀

Theorem 9 only makes sense if $\epsilon_0(n), \epsilon_1(n) < 1/2cn^2$. Most of the distinguishers from [17] have an error probability of $1/n^c$ for any chosen $c > 0$; see Theorem 7 for the case of f.g. linear groups. Moreover, notice that if $\epsilon_1 = 0$ then also $\zeta_1 = 0$ in Theorem 9. Combined with Theorem 7, Lemma 3, and the transfer theorems from [17] mentioned in Section 1, we get a logspace randomized streaming algorithm with a one-sided error (no error if $w \equiv_G 1$) for the word problem of any f.g. group G that can be constructed from f.g. linear groups using finite extensions, wreath products with a f.g. abelian left factor, and graph products. These groups are in general not linear; see the characterization of linear wreath products in [23].

6 Randomized streaming algorithms for subgroup membership

Let G be a f.g. group with a finite symmetric generating set Σ and let H be a subgroup of G. As before, $\pi_G : \Sigma^* \to G$ is the morphism that maps a word $w \in \Sigma^*$ to the group element represented by w. We define the language $\mathsf{GWP}(G, H, \Sigma) = \{w \in \Sigma^* : \pi_G(w) \in H\}$. GWP stands for generalized word problem which is another common name for the subgroup membership problem. Note that $\mathsf{GWP}(G, 1, \Sigma) = \mathsf{WP}(G, \Sigma)$. In the following we are interested in randomized streaming algorithms for $\mathsf{GWP}(G, H, \Sigma)$. One can easily show a statement for $\mathsf{GWP}(G, H, \Sigma)$ analogously to Lemma 2, which allows us to skip the generating set Σ in combination with the $\mathcal{O}$-notation and just write $\mathsf{GWP}(G, H)$ in the following.

For this section we fix a finite alphabet Σ and take a copy $\Sigma^{-1} = \{a^{-1} : a \in \Sigma\}$ of formal inverses. Let $\Gamma = \Sigma \cup \Sigma^{-1}$. We extend the mapping $a \mapsto a^{-1}$ $(a \in \Sigma)$ to the whole alphabet Γ by setting $(a^{-1})^{-1} = a$. Moreover, for a word $w = a_1 a_2 \cdots a_n$ with $a_i \in \Gamma$ we define $w^{-1} = a_n^{-1} \cdots a_2^{-1} a_1^{-1}$. A word $w \in \Gamma^*$ is called reduced if it contains no factor of the form aa^{-1} for $a \in \Gamma$. Let $\mathsf{Red}(\Gamma) \subseteq \Gamma^*$ be the set of reduced words. It is convenient to identify the free group $F(\Sigma)$ with the set $\mathsf{Red}(\Gamma)$ of reduced words and the following multiplication operation: Let $u, v \in \mathsf{Red}(\Gamma)$. Then one can uniquely write u and v as $u = xy$ and $v = y^{-1}z$ such that $xz \in \mathsf{Red}(\Gamma)$ and define the product of u and v in the free group $F(\Sigma)$ as xz. For every word $w \in \Gamma^*$ we can define a unique reduced word $\mathsf{red}(w)$ as follows: if $w \in \mathsf{Red}(\Gamma)$ then $\mathsf{red}(w) = w$ and if $w = uaa^{-1}v$ for $u, v \in \Gamma^*$ and $a \in \Gamma$ then $\mathsf{red}(w) = \mathsf{red}(uv)$. It is important that this definition does not depend on which factor aa^{-1} is deleted in w. The reduction relation $uaa^{-1}v \to uv$ for all $u, v \in \Gamma^*$ and $a \in \Gamma$ is a so-called confluent relation. The reduction mapping $w \mapsto \mathsf{red}(w)$ then becomes the canonical morphism mapping a word $w \in \Gamma^*$ to the element of the free group represented by w.

In the following we have to deal with a special class of finite automata over the alphabet Γ. A partial deterministic finite automaton (partial DFA) is defined as an ordinary DFA except that the transition function $\delta : Q \times \Gamma \to Q$ is only partially defined. As for (total) DFAs we extend the partial transition function $\delta : Q \times \Gamma \to Q$ to a partial function $\delta : Q \times \Gamma^* \to Q$. For $q \in Q$ and $w \in \Gamma^*$ we write $\delta(q, w) = \perp$ if $\delta(q, w)$ is undefined, which means that one cannot read the word w into the automaton $\mathcal{A}$ starting from state q. A *partial inverse automaton* $\mathcal{A} = (Q, \Gamma, q_0, \delta, q_f)$ over the alphabet Γ is a partial DFA with a single final state q_f and such that for all $p, q \in Q$ and $a \in \Gamma$, $\delta(p, a) = q$ implies $\delta(q, a^{-1}) = p$.

The main technique to deal with f.g. subgroups of a free group is Stallings folding [12]. For our purpose it suffices to know that for every f.g. subgroup $G \leq F(\Sigma)$ there exists a partial inverse automaton $\mathcal{A}_G$ over the alphabet Γ such that for every $w \in \mathsf{Red}(\Gamma)$, $w \in G$ if and only if $w \in L(\mathcal{A}_G)$. We call $\mathcal{A}_G$ the Stallings automaton for G. It has the additional property that the unique final state is the initial state. The Stallings automaton for G can be constructed quite efficiently from a given set of generators for G, but we do not need this fact since G will be fixed and not considered to be part of the input in our main result, Theorem 14 below.

Let G be a fixed f.g. subgroup of $F(\Sigma)$ and let $\mathcal{A}_G = (Q, \Gamma, q_0, \delta, q_0)$ be its Stallings automaton in the following. An important property of $\mathcal{A}_G$ is the following: If $q, q' \in Q$ and $u \in \Gamma^*$ (u is not necessarily reduced) are such that $\delta(q, u) = q'$ then also $\delta(q, \mathsf{red}(u)) = q'$. This follows from the fact that $\delta(q, aa^{-1}) = q$ for every $q \in Q$ and $a \in \Gamma$. In particular, if $\delta(q_0, u) \neq \perp$, then $u \in L(\mathcal{A}_G)$ if and only if $\mathsf{red}(u) \in L(\mathcal{A}_G)$ if and only if $\mathsf{red}(u) \in G$.

▶ **Definition 12.** *For a word $w \in \Gamma^*$ we define the $\mathcal{A}_G$-factorization of w uniquely as either*

(i) $w = w_0 a_1 u_1 \; w_1 a_2 u_2 \cdots w_{k-1} a_k u_k \; w_k$ *or*

(ii) $w = w_0 a_1 u_1 \; w_1 a_2 u_2 \cdots w_{k-1} a_k u_k \; w_k a_{k+1} v$

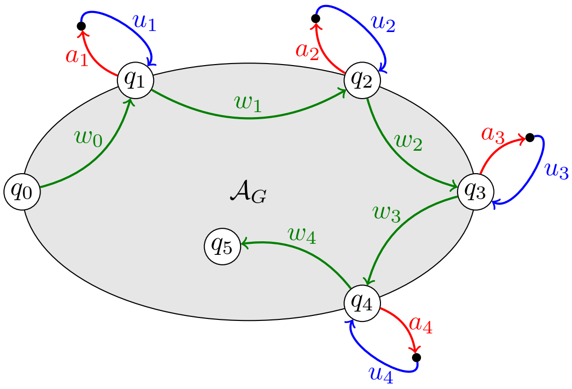


Figure 1 An $\mathcal{A}_G$-factorization of type (i) for $k = 4$. The red-blue loops outside of $\mathcal{A}_G$ are loops in the Cayley-graph of the free group $F(\Sigma)$.

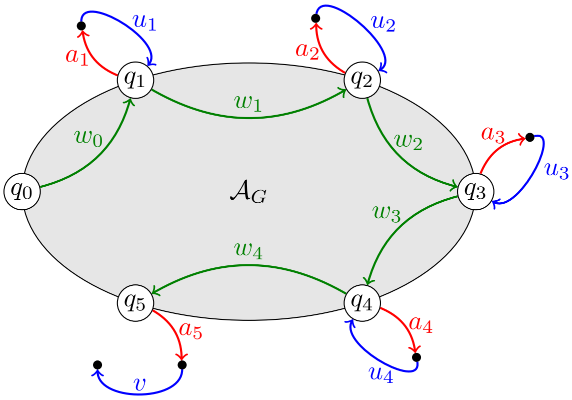


Figure 2 An $\mathcal{A}_G$-factorization of type (ii) for $k = 4$. The red-blue loops outside of $\mathcal{A}_G$ are loops in the Cayley-graph of the free group $F(\Sigma)$.

such that the following properties hold, where $\ell = k$ in case (i) and $\ell = k+1$ in case (ii):

- $w_0, \ldots, w_k, u_1, \ldots, u_k, v \in \Gamma^*$, $a_1, \ldots, a_\ell \in \Gamma$,
- *there are states $q_1, \ldots, q_{k+1} \in Q$ such that $\delta(q_i, w_i) = q_{i+1}$ for all $i \in [0, k]$ (recall that q_0 is the initial state of $\mathcal{A}_G$),*
- *$\delta(q_i, a_i) = \bot$ for all $i \in [1, \ell]$,*
- *for all $i \in [1, k]$, $\mathsf{red}(a_i u_i) = \varepsilon$ but there is no prefix $u \neq u_i$ of u_i with $\mathsf{red}(a_i u) = \varepsilon$, and*
- *in case (ii), v has no prefix x with $\mathsf{red}(a_{k+1} x) = \varepsilon$.*

Depending on which of the two cases (i) and (ii) in Definition 12 holds, we say that w has an $\mathcal{A}_G$-factorization of type (i) or type (ii).

Let us explain the intuition of the $\mathcal{A}_G$-factorization of w; see also Figures 1 and 2. We start reading the word w into the automaton $\mathcal{A}_G$, beginning at q_0, as long as possible. If it turns out that $\delta(q_0, w)$ is defined, then the $\mathcal{A}_G$-factorization of w consists of the single factor $w_0 = w$ and we obtain type (i). Otherwise, there is a shortest prefix w_0 of w (the first factor of the $\mathcal{A}_G$-factorization) such that after reading w_0 we reach the state $\delta(q_0, w_0) = q_1$ of $\mathcal{A}_G$ and $\delta(q_1, a_1) = \bot$, where a_1 is the symbol following w_0 in w. In other words, when trying to read a_1, we escape the automaton $\mathcal{A}_G$ for the first time. At this point let $w = w_0 a_1 x$.

Algorithm 2 $1/n^c$-correct randomized streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$.

```
   global variables: $q \in Q$, $p, r \in R_n$, $\beta \in \{0,1\}$
   initialization:
1  $q := q_0$ ; $\beta := 1$ ;
2  guess $r \in R_n$ according to the initial state distribution $\lambda_n$ of $\mathcal{B}_n$ ;
   next input letter: $a \in \Gamma$
3  if $\beta = 1$ and $\delta(q,a) = \bot$ then
4      $\beta := 0$ ; $p := r$
5  end
6  if $\beta = 1$ and $\delta(q,a) \neq \bot$ then
7      $q := \delta(q,a)$
8  end
9  $r := \sigma_n(r,a)$ ;
10 if $\beta = 0$ and $r = p$ then
11     $\beta := 1$
12 end
13 accept if $\beta = 1$ and $q = q_0$
```

We then take the shortest prefix u_1 of x such that $a_1 u_1 = 1$ in $F(\Sigma)$ (if such a prefix does not exist, we terminate in case (ii)). This yields a new factorization $w = w_0 a_1 u_1 y$. We then repeat this process with the word y starting from the state q_1 as long as possible. There are two possible terminations of the process: starting from state q_k we can read the whole remaining suffix into $\mathcal{A}_G$ (and arrive in state q_{k+1}). This suffix then yields the last factor w_k and we obtain (i). In the other case, we leave the automaton $\mathcal{A}_G$ with the symbol a_{k+1} from state q_{k+1} ($\delta(q_{k+1}, a_{k+1}) = \bot$) and the remaining suffix has no prefix x such that $a_{k+1}x$ evaluates to the identity in $F(\Sigma)$. The remaining suffix then yields the last factor v and we obtain (ii). The following lemma is shown in [17].

▶ **Lemma 13.** *Let $w \in \Gamma^*$ and assume that the $\mathcal{A}_G$-factorization of w and the states $q_1, \ldots, q_{k+1}$ are as in Definition 12.*

- *If the $\mathcal{A}_G$-factorization of w is of type (i) then $\mathsf{red}(w) \in G$ if and only if $q_{k+1} = q_0$.*
- *If the $\mathcal{A}_G$-factorization of w is of type (ii) then $\mathsf{red}(w) \notin G$.*

▶ **Theorem 14.** *Let G a fixed f.g. subgroup of $F(\Sigma)$. For every $c > 0$ there is a $1/n^c$-correct randomized streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$ with space complexity $\mathcal{O}(\log n)$.*

Proof. We would like to use the Stallings automaton $\mathcal{A}_G = (Q, \Gamma, q_0, \delta, q_0)$ as a streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$ (note that $|Q|$ is a constant since G is fixed). The problem is that the input word is not necessarily reduced. We solve this problem by using a $(1/n^{c+2}, 0)$-distinguisher $(\mathcal{B}_n)_{n \geq 0}$ for $F(\Sigma)$ with space complexity $\mathcal{O}(\log n)$. It exists by Theorem 7 since f.g. free groups are linear. Fix an input length n and let $\mathcal{B}_n = (R_n, \Gamma, \lambda_n, \sigma_n)$. Consider an input word $w \in \Gamma^{\leq n}$. Our randomized streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$ is Algorithm 2.

The space needed by Algorithm 2 is $\mathcal{O}(\log n)$: The variables q and β need constant space and p and r both need $\mathcal{O}(\log n)$ bits. Let us now show that the error probability of Algorithm 2 is bounded by $1/n^c$. For this let $S = \mathcal{P}(w)$ be the set of all prefixes of w. For the initially guessed state $r_0 \in R_n$ (line 3) we have by Lemma 4:

$$\mathop{\mathsf{Prob}}_{r_0 \in R_n}[\equiv_{F(\Sigma)} \text{ equals } \equiv_{r_0} \text{ on } S] \geq 1 - 1/n^{c+2}\binom{|S|}{2} \geq 1 - 1/n^c.$$

Assume for the further consideration that the guessed state r_0 is such that $\equiv_{F(\Sigma)}$ and $\equiv_{r_0}$ are equal on S. We claim that under this assumption, Algorithm 2 accepts in line 13 after reading w if and only if $\mathsf{red}(w) \in G$. For this, assume that the $\mathcal{A}_G$-factorization of w and the states $q_1, \ldots, q_{k+1} \in Q$ are as in Definition 12. By Lemma 13 it suffices to show the following:

(a) If the $\mathcal{A}_G$-factorization of w is of type (i) then after reading w we have $\beta = 1$ and $q = q_{k+1}$ in Algorithm 2.

(b) If the $\mathcal{A}_G$-factorization of w is of type (ii) then $\beta = 0$ after reading w in Algorithm 2.

To see this, observe that Algorithm 2 simulates $\mathcal{B}_n$ on w starting from r_0 (line 9). Moreover, initially we have $\beta = 1$ (line 1). This implies that Algorithm 2 simulates the Stallings automaton $\mathcal{A}_G$ on w as long as possible (line 7). If this possible for the whole input w (i.e., $\delta(q_0, w) \neq \bot$) then w has an $\mathcal{A}_G$-factorization of type (i) consisting of the single factor w (i.e., $k = 0$). Moreover, after processing w by Algorithm 2, we have $\beta = 1$ and the program variable q holds $\delta(q_0, w) = q_1 = q_{k+1}$. We obtain the above case (a).

Assume now that $k > 0$. The $\mathcal{A}_G$-factorization of w starts with $w_0 a_1$, where $\delta(q_0, w_0) = q_1$ and $\delta(q_1, a_1) = \bot$. After processing w_0 by Algorithm 2 we have $q = q_1$ and $r = \sigma_n(r_0, w_0)$. While processing the next letter a_1, Algorithm 2 sets β to 0 and saves the current state $r = \sigma_n(r_0, w_0)$ of $\mathcal{B}_n$ in the variable p (line 4). Let us write $w = w_0 a_1 v$. Since the flag β was set to 0, Algorithm 2 only continues the simulation of $\mathcal{B}_n$ on input $a_1 v$ starting from state $\sigma_n(r_0, w_0) = p$. Our assumption that $\equiv_{F(\Sigma)}$ and $\equiv_{r_0}$ are equal on the set S implies that for every prefix x of v we have: $\mathsf{red}(a_1 x) = \varepsilon$ if and only if $p = \sigma_n(r_0, w_0) = \sigma_n(r_0, w_0 a_1 x)$. In line 10, the algorithm checks the latter equality in each step (as long as $\beta = 0$). If there is no prefix x of v with $\mathsf{red}(a_1 x) = \varepsilon$ then the $\mathcal{A}_G$-factorization of w is of type (ii) (it is $w_0 a_1 v$) and the flag β is 0 after reading w. We then obtain the above case (b). Otherwise, u_1 is the shortest prefix of v with $\mathsf{red}(a_1 u_1) = \varepsilon$. Moreover, after processing u_1, the if-condition in line 10 is true for the first time. The algorithm then sets the flag β back to 1 (line 11) and resumes the simulation of the automaton $\mathcal{A}_G$ in state q_1 (which is still stored in the program variable q). This process now repeats and we see that the algorithm correctly locates the factors of the $\mathcal{A}_G$-factorization of w. This shows the above points (a) and (b). ◀

It is not possible to generalize Theorem 14 to subgroups of $F(\Sigma)$ that are not finitely generated. The proof of the following theorem (see [17]) uses the fact there is a finitely presented group whose word problem has randomized streaming space complexity $\Omega(n)$. A concrete example is Thompson's group F; see [16, Corollary 22].

▶ **Theorem 15.** *The free group F_2 of rank two has a normal subgroup N such that the randomized streaming space complexity of the language* $\mathsf{GWP}(F_2, N)$ *is in $\Theta(n)$.*

Applying Mihaĭlova's construction [19] to the normal subgroup N from Theorem 15 yields:

▶ **Theorem 16.** *There is a f.g. subgroup G of $F_2 \times F_2$ such that the randomized streaming space complexity of* $\mathsf{GWP}(F_2 \times F_2, G)$ *is in $\Theta(n)$.*

7 Open problems

A very important class of groups in geometric group theory is the class of hyperbolic groups; see [9] for background. Free groups are the simplest hyperbolic groups. Hyperbolic groups have some good algorithmic properties. For instance, their word problems can be decided in linear time by Dehn's algorithm. It would be interesting to know whether every hyperbolic group has an ϵ-distinguisher (say for $\epsilon = 1/3$) with space complexity $\mathcal{O}(\log n)$, which would imply that every hyperbolic group has randomized streaming space complexity $\mathcal{O}(\log n)$. One should remark that it is also open whether the word problem for a hyperbolic group belongs to randomized logspace (RL). The best known space upper bound for the word problem of a hyperbolic group is $\mathsf{DPSPACE}(\log^2 n)$. This follows from the fact that the word problem of a hyperbolic group belongs to LogCFL [14].

It would be also interesting to see whether a transfer theorem similar to Theorem 9 can be shown for certain fundamental groups of graphs of groups, e.g. when all edge groups are finite.

References

1 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013. doi:10.1016/j.tcs.2012.12.028.

2 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 119:1–119:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.119.

3 J. Berstel. *Transductions and context–free languages.* Teubner Studienbücher, Stuttgart, 1979. doi:10.1007/978-3-663-09367-1.

4 William W. Boone. The word problem. *Annals of Mathematics. Second Series*, 70:207–265, 1959. doi:10.2307/1970103.

5 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144, 1911. In German. doi:10.1007/BF01456932.

6 Volker Diekert and Jonathan Kausch. Logspace computations in graph products. *Journal of Symbolic Computation*, 75:94–109, 2016. doi:10.1016/j.jsc.2015.11.009.

7 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

8 Elisabeth R. Green. *Graph Products of Groups.* PhD thesis, The University of Leeds, 1990.

9 Derek F. Holt, Sarah Rees, and Claas E. Röver. *Groups, Languages and Automata*, volume 88 of *London Mathematical Society Student Texts.* Cambridge University Press, 2017. doi:10.1017/9781316588246.

10 Tim Hsu and Daniel T. Wise. On linear and residual properties of graph products. *Michigan Mathematical Journal*, 46(2):251–259, 1999. doi:10.1307/mmj/1030132408.

11 Stephen P. Humphries. On representations of Artin groups and the Tits conjecture. *Journal of Algebra*, 169(3):847–862, 1994. doi:10.1006/jabr.1994.1312.

12 Ilya Kapovich and Alexei Myasnikov. Stallings foldings and subgroups of free groups. *Journal of Algebra*, 248(2):608–668, 2002. doi:10.1006/jabr.2001.9033.

13 Jonathan Kausch. *The parallel complexity of certain algorithmic problems in group theory.* PhD thesis, University of Stuttgart, 2017. URL: 10.18419/opus-9152.

14 Markus Lohrey. Decidability and complexity in automatic monoids. *International Journal of Foundations of Computer Science*, 16(4):707–722, 2005. doi:10.1142/S0129054105003248.

15 Markus Lohrey. *The Compressed Word Problem for Groups.* SpringerBriefs in Mathematics. Springer, 2014. doi:10.1007/978-1-4939-0748-9.

16 Markus Lohrey and Lukas Lück. Streaming word problems. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022*, volume 241 of *LIPIcs*, pages 72:1–72:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.MFCS.2022.72.

17 Markus Lohrey, Lukas Lück, and Julio Xochitemol. Streaming word problems. *CoRR*, abs/2202.04060, 2022. URL: `https://arxiv.org/abs/2202.04060`.

18 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014. `doi:10.1137/130926122`.

19 K. A. Mihaĭlova. The occurrence problem for direct products of groups. *Math. USSR Sbornik*, 70:241–251, 1966. English translation.

20 Pjotr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *American Mathematical Society, Translations, II. Series*, 9:1–122, 1958. `doi:10.1090/trans2/009`.

21 Azaria Paz. *Introduction to Probabilistic Automata.* Academic Press, 1971. `doi:10.1016/C2013-0-11297-4`.

22 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. `doi:10.1016/S0019-9958(63)90290-0`.

23 B. A. F. Wehrfritz. Wreath products and chief factors of linear groups. *Journal of the London Mathematical Society (2)*, 4:671–681, 1972. `doi:10.1112/jlms/s2-4.4.671`.

Algorithmic Dimensions via Learning Functions

Jack H. Lutz
Department of Computer Science, Iowa State University, Ames, IA, USA

Andrei N. Migunov
Department of Mathematics and Computer Science, Drake University, Des Moines, IA, USA

Abstract

We characterize the algorithmic dimensions (i.e., the lower and upper asymptotic densities of information) of infinite binary sequences in terms of the inability of learning functions having an algorithmic constraint to detect patterns in them. Our pattern detection criterion is a quantitative extension of the criterion that Zaffora Blando used to characterize the algorithmically random (i.e., Martin-Löf random) sequences. Our proof uses Lutz's and Mayordomo's respective characterizations of algorithmic dimension in terms of gales and Kolmogorov complexity.

2012 ACM Subject Classification Theory of computation → Computability; Theory of computation → Models of learning

Keywords and phrases algorithmic dimensions, learning functions, randomness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.72

Funding *Jack H. Lutz*: This research was supported in part by National Science Foundation research grants 1545028 and 1900716.
Andrei N. Migunov: This research was supported in part by National Science Foundation research grants 1545028 and 1900716.

1 Introduction

Algorithmic dimension was first formulated as a Σ_1^0 effectivization of classical Hausdorff dimension [12, 14].[1] The algorithmic dimension $\mathrm{adim}(\Gamma)$ of a set Γ of infinite binary sequences is, in fact, an upper bound of the Hausdorff dimension $\dim_H(\Gamma)$ of this set. Since algorithmic dimension has the *absolute stability* property that $\mathrm{adim}(\Gamma)$ is always the supremum of all $\mathrm{adim}(\{X\})$ for $X \in \Gamma$, it was natural to define $\dim(X) = \mathrm{adim}(\{X\})$ for all infinite binary sequences X and to investigate algorithmic dimension entirely in terms of the dimensions $\dim(X)$ of individual sequences X. Mayordomo [23] proved that the dimension $\dim(X)$, originally defined in terms of algorithmic betting strategies called *gales*, can also be characterized as the lower asymptotic density of the algorithmic information content of X. The more recent *point-to-set principle* [15] uses relativization to give the characterization

$$\dim_H(\Gamma) = \min_{A \subseteq \mathbb{N}} \sup_{X \in \Gamma} \dim^A(X)$$

of classical Hausdorff dimension. This principle has enabled several recent uses of computability theory to prove new classical theorems about Hausdorff dimension[21, 19, 20, 27, 18, 4, 5, 16, 17].[2]

Algorithmic dimension has the same Σ_1^0 "level of effectivization" as algorithmic randomness (also called "Martin-Löf randomness" [22] or, simply, "randomness"). In fact, every algorithmically random sequence X satisfies $\dim(X) = 1$, although the converse does not hold [14].

[1] Algorithmic dimension has also been called "constructive dimension", "effective Hausdorff dimension", and "effective dimension" by various authors.

[2] A theorem is "classical" here if its statement does not involve computability or related aspects of mathematical logic. Hence "new classical theorem" is not an oxymoron.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 72; pp. 72:1–72:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Computable learning theory, as initiated by Gold in 1967 [7, 25, 10], has been used to shed new light on randomness notions. Specifically, in 2008, Osherson and Weinstein [24] characterized two randomness notions, called weak 1-randomness and weak 2-randomness, for sequences X in terms of the inability of computable learning functions to detect patterns in X. Even more compellingly, Zaffora Blando [32] recently formulated a clever variant of Osherson and Weinstein's pattern detection criteria, called uniform weak detection, and used this to give an exact characterization of algorithmic (i.e., Martin-Löf) randomness.[3]

In this paper we introduce a quantitative version of Zaffora Blando's uniform weak detection criterion, called *s-learnability*, and we use this to characterize algorithmic dimension in terms of learning functions. Our main theorem says that, for every infinite binary sequence X, $\dim(X)$ is the infimum of all nonnegative real numbers s for which some learning function s-learns X. Our proof of this result uses methods of Osherson, Weinstein, and Zaffora Blando, together with martingale and Kolmogorov complexity techniques of Mayordomo [23]. We also characterize both the classical *packing dimension* $\dim_P$ [29, 30] and the *algorithmic strong dimension* $\mathrm{Dim}(X)$ [1] of a sequence in terms of learning functions. Along the way, we show that algorithmic randomness can also be characterized by specifically *polynomial-time* computable learning functions.

2 Preliminaries

Let $\mathbb{N}$ represent the natural numbers $\{0, 1, 2, ...\}$, $\mathbb{Q}$ the rationals, and $\mathbb{R}$ the reals. We will often use the extended naturals $\mathbb{N} \cup \{\infty\}$ and the extended reals $\mathbb{R} \cup \{\infty\}$. More often, we will refer to the respectively half open and closed intervals, $[0, \infty)$ and $[0, \infty]$.

We denote by $\{0, 1\}^*$ the set of all (finite) binary strings, and by $\{0, 1\}^\infty$ the set of all infinite binary sequences, which we call the *Cantor space* $\mathbf{C}$. We denote the length in bits of a string or sequence w by $|w|$. The *empty string* is the unique string λ with $|\lambda| = 0$. If Z is an element of $\{0, 1\}^\infty$ or of $\{0, 1\}^*$, we write $Z \upharpoonright n$ for the first n bits of Z if $|Z| \geq n$, and the value is undefined otherwise . Note that for all $n \in \mathbb{N}$, and all $Z \in \{0, 1\}^\infty$, $|Z \upharpoonright n| = n$. We write $w \sqsubseteq Z$ if w is a *prefix* of Z, i.e., if $w = Z \upharpoonright |w|$. We write $w \sqsubset Z$ if w is a *proper prefix* of Z, i.e., if w is a prefix of Z and $w \neq Z$.

For any string $w \in \{0, 1\}^*$, the *cylinder* at w is

$$C_w = \{Z \in \mathbf{C} \mid w \sqsubseteq Z\}.$$

If A is a set of strings, we denote the union of cylinders at those strings by $[\![A]\!] = \cup_{w \in A} C_w$. A (Borel) probability measure on $\mathbf{C}$ is a function $\mu : \{0, 1\}^* \to [0, 1]$ such that $\mu(\lambda) = 1$ and $\mu(w) = \mu(w0) + \mu(w1)$ for all $w \in \{0, 1\}^*$. Intuitively, $\mu(w)$ is the probability that $Z \in C_w$ when $Z \in \mathbf{C}$ is "chosen according to μ". In this sense, $\mu(w)$ is an abbreviation for $\mu(C_w)$. Standard methods [2] extend μ from cylinders to a σ-algebra $\mathcal{F}$ on $\mathbf{C}$, so that $(\mathbf{C}, \mathcal{F}, \mu)$ is a probability space in the classical sense.

Most of our attention is on the uniform (Lebesgue) probability measure λ on $\mathbf{C}$ defined by $\lambda(w) = 2^{-|w|}$ for all $w \in \{0, 1\}^*$. We rely on context to distinguish Lebesgue measure from the empty string.

[3] Zaffora Blando's paper also characterized Schnorr randomness in terms of "computably uniform weak detection", but this is not germane to our work here.

▶ **Definition.** *A function $f : \{0,1\}^* \to [0,\infty)$ is lower semicomputable if there exists a computable function $g : \{0,1\}^* \times \mathbb{N} \to \mathbb{Q} \cap [0,\infty)$ such that for all $w \in \{0,1\}^*, t \in \mathbb{N}$,*

$$g(w,t) \leq g(w,t+1) \leq f(w)$$

and

$$\lim_{t\to\infty} g(w,t) = f(w).$$

We will say that a set S of real numbers is *uniformly left computably enumerable (uniformly left c.e.)* if there exists a lower-semicomputable function whose range is S.

3 Algorithmic randomness via learning functions

The algorithmic randomness of a sequence was originally defined in [22] in terms of algorithmic measure theory. In this view, an algorithmically random sequence is one which belongs to every algorithmically definable measure one set. Martin-Löf shows that all algorithmically nonrandom sequences belong to one universal algorithmically measure-zero set.

▶ **Definition** ([22]). *If μ is a probability measure on $\boldsymbol{C}$, then we say a set X has* algorithmic μ-measure zero *if there exists a computable function $g : \mathbb{N} \times \mathbb{N} \to \{0,1\}^*$ such that: for every $k \in \mathbb{N}$,*

$$X \subseteq \bigcup_{n=0}^{\infty} C_{g(k,n)}$$

and

$$\sum_{n=0}^{\infty} \mu(C_{g(k,n)}) \leq 2^{-k}.$$

▶ **Definition** ([22]). *We say a sequence S is* Martin-Löf μ-nonrandom *if $\{S\}$ has algorithmic μ-measure zero, and* Martin-Löf μ-random *otherwise.*

When the probability measure μ is the Lebesgue measure λ defined in section 2, we omit it from the terminology in the preceding two definitions. Randomness can also be characterized using gambling strategies called *gales*.

▶ **Definition** ([14]). *For $s \in [0,\infty)$, a* μ-s-gale *is a function $d : \{0,1\}^* \to [0,\infty)$ that satisfies the condition that*

$$d(w)\mu(w)^s = d(w0)\mu(w0)^s + d(w1)\mu(w1)^s,$$

for every $w \in \{0,1\}^*$.

Sometimes, when the probability distribution μ is clear from context, we will refer simply to s-gales. A *martingale* is a 1-gale. If not stated explicitly otherwise, we assume that the *initial capital* of a gale is $d(\lambda) = 1$.

▶ **Definition.** *A μ-s-gale d* succeeds *on a set Γ of sequences if*

$$\limsup_{n\to\infty} d(X \restriction n) = \infty$$

for every sequence $X \in \Gamma$.

Ville shows the following:

▶ **Theorem 1** ([31]). *Let $\lambda(E)$ denote the Lebesgue measure of a set $E \subseteq \{0,1\}^\infty$. The following are equivalent:*

(1) $\lambda(E) = 0$

(2) *There exists a martingale $d : \{0,1\}^* \to [0,\infty)$ that succeeds on E.*

Schnorr effectivizes Ville's theorem as follows:

▶ **Theorem 2** ([26]). *A set of sequences Γ is Martin-Löf random if and only if there exists no lower-semicomputable martingale that succeeds on Γ.*

We will revisit gales in a later section, when we discuss dimension.

▶ **Definition** ([25]). *A function $l : \{0,1\}^* \to \{\text{YES}, \text{NO}\}$ is called a* learning function.

YES and NO are simply aliases for 1 and 0, respectively.

One can impose resource bounds on learning functions or on properties of these functions such as their average answers "along" a string w. l may be computable (in which case we call it a computable learning function or *CLF*), or l may have lower-semicomputable averages at all points in $\{0,1\}^*$, or any number of other resource restrictions.

▶ **Definition** ([32]). *A learning function l is said to* uniformly weakly detect *that a sequence $X \in \{0,1\}^\infty$ is patterned if and only if*

1. $l(X \upharpoonright m) = \text{YES}$ *for infinitely many $m \in \mathbb{N}$, and*
2. $\lambda(\{Y \in \{0,1\}^\infty \mid \#\{m \in \mathbb{N} \mid l(Y \upharpoonright m) = \text{YES}\} \geq n\}) \leq 2^{-n}$
   for all $n \in \mathbb{N}$.

Zaffora Blando shows that computable learning functions and uniform weak detectability characterise Martin-Löf randomness:

▶ **Theorem 3** ([32]). *A sequence $X \in \{0,1\}^\infty$ is Martin-Löf random if and only if there is no computable learning function that uniformly weakly detects that X is patterned.*

▶ **Theorem 4.** *The following are equivalent:*

1. *There exists a computable learning function that uniformly weakly detects that X is patterned.*
2. *There exists a polynomial-time computable learning function that uniformly weakly detects that X is patterned.*

Proof. (2) $\implies$ (1) is immediate.

To see that (1) $\implies$ (2):

Let l be a computable learning function which uniformly weakly detects that X is patterned. Let $M_l(w)$ be a TM which computes $l(w)$. Algorithm 1 specifies a learning function with the following properties:

$\hat{l} = \text{YES}$ if there exist $w' \sqsubset w$ and $t_{w'} \in \mathbb{N}$ such that all of the following hold:

1. $|w| = |w'| + t_{w'}$
2. $l(w') = \text{YES}$
3. $t_{w'} = \min\{t \mid M_l(w') = \text{YES} \text{ after } t \text{ steps}\}$,

and $\hat{l} = \text{NO}$ otherwise.

If l says YES infinitely often on X, then there are infinitely many $w', t_{w'}$ where $M_l(w') = \text{YES}$ after exactly $t_{w'}$ time steps. Thus there are infinitely many w with $|w| = |w'| + t_{w'}$ where $\hat{l}$ says YES. Thus, $\hat{l}$ says YES infinitely often on X.

For all $Y \in \{0,1\}^\infty$, the number of YES given by $\hat{l}$ along Y remains the same as the number given by l, all such answers being "delayed" until later. Thus, the measure condition is satisfied. Thus, $\hat{l}$ is a learning function which uniformly weakly detects that X is patterned.

Algorithm 1 $M_{\hat{l}}$.

```
Input $w$:
for all $w' \sqsubset w$ do
    Run $M_l(w')$ for exactly $|w| - |w'|$ steps.
    if $M_l(w')$ prints YES for the first time after exactly $|w| - |w'|$ steps then
        return YES
    else
        Continue
    end if
end for
return NO
```

$M_{\hat{l}}$ always halts and terminates in time polynomial in $|w|$. Thus, $\hat{l}$ is polynomial-time computable. ◀

As a result, one can characterize algorithmic randomness in terms of polynomial-time computable learning functions:

▶ **Corollary 5.** *A sequence $X \in \{0,1\}^\infty$ is Martin-Löf random if and only if there is no polynomial-time computable learning function that uniformly weakly detects that X is patterned.*

We also note a useful fact about uniform weak detectability:

▶ **Observation 6.** *If l_1 uniformly weakly detects that Γ_1 is patterned, and l_2 uniformly weakly detects that Γ_2 is patterned, then there exists a learning function l_3 which uniformly weakly detects that $\Gamma_1 \cup \Gamma_2$ is patterned. This transformation preserves computability.*

Proof. Define l_3 by:

$l_3(v) =$ YES if either $l_1(v) =$ YES or $l_2(v) =$ YES, unless for all $v' \sqsubset v$ $l_1(v) =$ NO or for all $v' \sqsubset v$ $l_2(v) =$ NO.

That is, l_3 says YES whenever either l_1 or l_2 would say YES, except for the first time for each.

It is easy to verify that l_3 says YES infinitely often on every $X \in \Gamma_1 \cup \Gamma_2$ and that the measure property is satisfied. It is also easy to show that if l_1 and l_2 are computable, then l_3 is as well. ◀

As a result, uniform weak detectability by computable learning functions is closed under finite unions.

4 Classical and algorithmic dimensions

Next, we review the definitions of classical and algorithmic dimensions.

We say a set A *covers* a set of sequences Γ if for every $X \in \Gamma$ there is some $w \in A$, $w \sqsubseteq X$. Let $k \in \mathbb{N}$, and let $\mathcal{A}_k = \{A \mid A \text{ is a prefix set and } \forall x \in A, |x| \geq k\}$. Let

$$\mathcal{A}_k(\Gamma) = \{A \in \mathcal{A}_k \mid A \text{ covers } \Gamma\},$$

and let

$$H^s_k(\Gamma) = \inf_{A \in \mathcal{A}_k(\Gamma)} \sum_{w \in A} 2^{-s|w|}.$$

Note that this infimum is taken only over sets of cylinders, not over all possible covers. For that reason, the following function is a proxy for - and is within a constant multiplicative factor of - what is know as the *s-dimensional Hausdorff outer measure*, in which all covers are considered.

▶ **Definition** ([8]). $H^s(\Gamma) = \lim_{k\to\infty} H^s_k(\Gamma)$.

For any set Γ, there exists some $s \in [0, \infty)$ such that for every $a < s < b$,

1. $H^a(\Gamma) = \infty$, and
2. $H^b(\Gamma) = 0$.

The real number s is the Hausdorff dimension of Γ:

▶ **Definition** ([8]). *The* Hausdorff dimension *of a set* $\Gamma \subseteq \{0,1\}^\infty$ *is*

$$\dim_H(\Gamma) = \inf\{s \in [0,\infty) \mid H^s(\Gamma) = 0\}.$$

Hausdorff dimension can be characterized in terms of gales:

▶ **Theorem 7** ([13]).

$$\dim_H(\Gamma) = \inf\{s \in [0,\infty)| \textit{ there exists an s-gale that succeeds on } \Gamma\}.$$

Lutz also showed that by effectivizing gales[4] at various levels, one can obtain various *effective* dimension notions, including the *algorithmic dimension*:

▶ **Definition** ([14]). *The* algorithmic dimension *of a set* $\Gamma \subseteq \{0,1\}^\infty$ *is*

$$\text{adim}(\Gamma) = \inf\{s \in [0,\infty)| \textit{ there exists}$$

$$\textit{a lower semicomputable s-gale that succeeds on } \Gamma\}.$$

We also note the following theorem:

▶ **Theorem 8** ([14]). *Let μ be a probability measure on $\{0,1\}^\infty$, let $s, s' \in [0,\infty)$, and let $d, d' : \{0,1\}^* \to [0,\infty)$. Assume that*

$$d(w)\mu(w)^s = d'(w)\mu(w)^{s'}$$

for all $w \in \{0,1\}^*$. Then, d is a μ-s-gale if and only if d' is a μ-s'-gale.

A corollary of this theorem, which we will make use of in the main proof of the next section, is the following:

▶ **Proposition 9.** *If d is an s-gale that succeeds on X then there exists a martingale d' that succeeds on X against order $h(w) = 2^{(1-s)|w|}$. That is,*

$$\limsup_{n\to\infty} \frac{d(X \upharpoonright n)}{h(n)} = \infty.$$

[4] In Lutz' original proof, *supergales* are used. These satisfy the gale definition with $\leq$ in place of equality. [9] shows that gales suffice for the characterization of algorithmic dimension.

Another important classical dimension notion is the packing dimension [30, 29]. As with the Hausdorff dimension, it is easier to define it in terms of gales than in terms of its original conception via coverings. First we define a notion of *strong success* for gales:

▶ **Definition** ([1]). *A μ-s-gale d* succeeds strongly *on a set Γ of sequences if*

$$\liminf_{n\to\infty} d(X \restriction n) = \infty$$

for every sequence $X \in \Gamma$.

▶ **Definition** ([1]). *The* packing dimension *of a set $\Gamma \subseteq \{0,1\}^\infty$ is*

$$\dim_P(\Gamma) = \inf\{s \in [0,\infty) | \text{ there exists an } s\text{-gale that succeeds strongly on } \Gamma\}.$$

The packing dimension can be effectivized as follows:

▶ **Definition** ([1]). *The* algorithmic packing dimension *or* algorithmic strong dimension *of a set $\Gamma \subseteq \{0,1\}^\infty$ is*

$$\mathrm{aDim}(\Gamma) = \inf\{s \in [0,\infty) | \text{ there exists a lower semi-computable } s\text{-gale}$$

$$\text{that succeeds strongly on } \Gamma\}.$$

We use the above notations adim and aDim when describing the algorithmic dimensions of sets in order to distinguish these from their classical counterparts. When applied to individual sequences, we often use dim and Dim, respectively, as there is no ambiguity.

5 Hausdorff dimension via learning functions

Learning functions can be used to characterize the classical (Hausdorff) dimension. Once such a characterization is in place, one can impose further restrictions on the computability of learning functions and thereby use the notion of learning to characterize dimension notions at various levels of effectivity.

We refine the definition of uniform weak detectability, by adding a requirement on the frequency of YES answers, in order to characterize dimension. Recall that we identify YES with 1 and NO with 0, thus the sum in the following definition is well-defined.

▶ **Definition.** *If l is a learning function, the* path average of l up to w *is denoted*

$$\mathrm{AVG}_l(w) = \frac{\sum_{i=0}^{i=|w|} l(w \restriction i)}{|w|}.$$

Let Δ be a computability restriction such as “lower semi-computable”, “computable”, or “all” (the absence of a restriction).

▶ **Definition.** *A sequence X is (Δ)-s-*learnable *if and only if there exists a function l : $\{0,1\}^* \to \{\mathrm{YES}, \mathrm{NO}\}$ such that the following three conditions are satisfied.*

1. *For every $w \in \{0,1\}^*$, the path averages $\mathrm{AVG}_l(w)$ are uniformly Δ-computable (in w).*
2. *For all $n \in \mathbb{N}$,*

$$\lambda(\{Y \in \{0,1\}^\infty \mid \#\{i \mid l(Y \restriction i) = \mathrm{YES}\} \geq n)\}) \leq 2^{-n}.$$

3. $\limsup_{n\to\infty} \mathrm{AVG}_l(X \restriction n) \geq 1 - s.$

We say that a sequence X is *strongly* (Δ)*-s-learnable* if it satisfies (1) and (2) above, and satisfies condition (3) with a $\liminf$ rather than a $\limsup$.

Specifically, we will say that *algorithmic* s-learnability corresponds to $\Delta = \Sigma^0_1$, *computable* s-learnability corresponds to $\Delta = \Delta^0_1$ and s-learnability as such corresponds to no resource restriction (in other words, any learning functions at all can be used). Note that the restriction on computability is not applied to the learning function itself, but to its path averages on the elements of $\{0,1\}^*$.

We often refer to a learning function which satisfies these properties as an *s-learner*, and we say it *s-learns* a sequence or set of sequences.

This definition is in the spirit of [28], emphasizing learnability criteria based on frequencies of YES answers, rather than restrictions on the measure conditions.

A function $l : \{0,1\}^* \to \{\text{NO}, \text{YES}\}$ is associated to a transformation $\hat{l} : \{0,1\}^\infty \to \{\text{NO}, \text{YES}\}^\infty$ defined by $\hat{l}(Y) = X$, where $X[n] = l(Y \restriction n)$. In this sense, each sequence x is transformed into a sequence consisting of YES and NO "bits". For any set S, and function $f : \{0,1\}^\infty \to \{0,1\}^\infty$, $f^{-1}[S]$ is the set of all X with $f(X) \in S$. Define functions $\hat{l}$ on finite strings analogously.

► **Observation 10.** *For every learner l and every $Y \in \{0,1\}^\infty$, either l s-learns every $X \in \hat{l}^{-1}[Y]$ or none of them.*

This is immediate from the success criterion. s-learning a sequence is purely a matter of the asymptotic frequency with which some learner says YES on prefixes of that sequence, assuming the measure condition is satisfied by l.

Closure under finite unions also holds for s-learnability:

► **Observation 11.** *If l_1 s-learns Γ_1 and l_2 s-learns Γ_2, then there exists a learning function l_3 which s-learns $\Gamma_1 \cup \Gamma_2$.*

Proof. The idea is the same as in Observation 6. Define l_3 by:

$l_3(v) = \text{YES}$ if either $l_1(v) = \text{YES}$ or $l_2(v) = \text{YES}$, unless it's the first time that either l_1 or l_2 has said YES on any $v' \sqsubseteq v$.

Note that $\limsup_{n\to\infty} \frac{\sum^n l(X\restriction i)}{n} \geq (1-s)$ implies that $\limsup_{n\to\infty} \frac{\sum^n l(X\restriction i)}{n} - \frac{k}{n} \geq (1-s)$ for any fixed k. ◄

We will now show that s-learning characterizes Hausdorff dimension.

Let $\mathscr{G}(\Gamma) = \{s \in (0,\infty) \mid$ there exists a learning function l which s-learns every $X \in \Gamma\}$. The following theorem is a characterization of Hausdorff dimension in terms of learning functions.

► **Theorem 12.** *For all $\Gamma \subseteq \{0,1\}^\infty$,*

$$\dim_H(X) = \inf \mathscr{G}(\Gamma).$$

Proof. Assume $\dim_H(\Gamma) \leq s$. Then for all $s' > s$ there exists an s'-gale which succeeds on Γ, and by Proposition 9 there exists a martingale d which, for every $X \in \Gamma$ succeeds on X against order $h(w) = 2^{(1-s')|w|}$. That is, d doubles its money asymptotically $1 - s'$ share of the time. Let

$$\gamma_d(w) = \begin{cases} \text{YES} & \text{if there exists } w' \sqsubseteq w \text{ such that } d(w) \geq 2 \cdot d(w') \text{ and} \\ & \forall \hat{w} \text{ satisfying } w' \sqsubset \hat{w} \sqsubset w,\ \gamma_d(\hat{w}) = \text{NO} \\ \text{NO} & \text{otherwise} \end{cases} \tag{5.1}$$

γ_d says YES every time d doubles its money (attains a new 2^k value). The set of sequences on which γ_d says YES infinitely often is the same as the set of sequences on which d wins unbounded money *against* h, thus it has measure zero. Thus for any $X \in \Gamma$ there must be infinitely many $X \upharpoonright n$ such that $\gamma_d(X \upharpoonright n)$ has YES density at least $(1 - s')$ at $X \upharpoonright n$.

Let $n \in \mathbb{N}$ and let $B_n = \{w \mid d(w) \geq 2^n \text{ and } \forall v \sqsubseteq w, d(v) < 2^n\}$. B_n is the (prefix-)set of all w at which d has accumulated 2^n value for "the first time". By the Kolmogorov inequality ([31]), $\lambda(\cup_{w \in B_n} C_w) \leq 2^{-n}$. For all n,

$$\begin{aligned} A_n^{\gamma_d} &= \{Y \mid \#\{m \mid \gamma_d(Y \upharpoonright m) = \text{YES}\} \geq n\} \\ &\subseteq \{Y \mid \exists k \; d(Y \upharpoonright k) \geq 2^n\} \\ &= \cup_{w \in B_n} C_w, \end{aligned}$$

Thus, for all n, $\lambda(A_n^{\gamma_d}) \leq \lambda(\cup_{w \in B_n} C_w) \leq 2^{-n}$. Thus γ_d s'-learns every $X \in \Gamma$.

In the other direction, suppose that for all $s' > s$ there is a learning function which s'-learns every $X \in \Gamma$. Let l be such a learning function. We show $\dim_H(\Gamma) \leq s$. Let k, r be any integers. Let

$$\hat{A}_k = \{w \mid \#\,\text{YES}(w) \geq r + (1 - s')|w| \text{ and } \forall v \sqsubset w, v \notin \hat{A}_k\}.$$

$\hat{A}_k \in \mathcal{A}_k(\Gamma)$ because for all $w \in \hat{A}_k$, $|w| > k$, and $\hat{A}_k$ is a prefix set.

For every n,

$$\lambda(\{Y \mid \#\,\text{YES}(Y) \geq r + (1 - s')n\}) \leq 2^{-r-(1-s')n}.$$

Thus, the number of strings of length exactly k which can have at least $r + (1 - s')k$ YES answers is at most

$$\frac{2^{-r-(1-s')k}}{2^{-k}} = 2^{-r+s'k}.$$

$$H_k^{s'}(\Gamma) \leq \sum_{w \in \hat{A}_k} 2^{-s'|w|} \leq \sum_{n=k}^{\infty} |\hat{A}_k^{=n}| 2^{-s'n} \leq 2^{-r+s'k} 2^{-s'k} = 2^{-r},$$

where the third inequality is due to the Kraft inequality [11, 3], because $\hat{A}_k$ is a prefix set.

Thus

$$\begin{aligned} H^{s'}(\Gamma) &= \lim_{k \to \infty} \inf_{A \in \mathcal{A}_k(\Gamma)} \sum_{w \in A} 2^{-s|w|} \\ &\leq \lim_{k \to \infty} \sum_{w \in \hat{A}_k} 2^{-s|w|} \\ &= 0. \end{aligned}$$

Thus $\dim_H(\Gamma) \leq s$. ◀

6 Algorithmic dimension via learning functions

In this section, we show that algorithmic s-learning characterizes algorithmic dimension.

Recall that a learning function *algorithmically s-learns* a set $\Gamma \subseteq \{0, 1\}^\infty$ if it satisfies the definition of s-learning with $\Delta = \Sigma_1^0$. In other words, we require that the path-averages of l on all strings w are uniformly lower semicomputable real numbers.

Before we discuss the proof that algorithmic s-learnability characterizes algorithmic dimension, we note why it is at least *seemingly difficult* to directly lower semi-compute the learning function described in Theorem 12. For one, a suitable notion of lower semi-computability is hard to establish for functions mapping to $\{0,1\}$. Secondly, the learning function γ_d in the proof of Theorem 12 relies on being able to place a YES answer every time the "underlying" martingale doubles its money for the first time. That is, every time it achieves a new value 2^k for the first time along some sequence. Though checking whether this happens *eventually* is lower semicomputable, checking whether it has happened at a particular w for the *first time* is not. Instead, we can simply require that path-averages be lower semicomputable. Thus, the conditions of algorithmic s-learnability coincide with the existence of lower semi-computable martingales succeeding against exponential orders.

For a given martingale d, we define the learning function γ_d as in Theorem 12. Note that if d is lower semi-computable with computable witness $\hat{d}$, then the path averages of γ_d are lower semi-computable uniformly in w via the computable witness below (Algorithm 2).

Algorithm 2 AVG_l.

```
Input $w, k$:
Compute all the values $\hat{d}(\lambda, k), \dots, \hat{d}(w, k)$.
Compute $m_{w,k} = \max_{w' \sqsubseteq w} \left\{ \log_2 \left( \hat{d}(w', k) \right) \right\}$
return $\frac{m_{w,k}}{|w|}$
```

As a result, the set

$$T = \{w \mid \mathrm{AVG}_{\gamma_d}(w) \geq 1 - s\}$$

is computably enumerable, and so are all of the slices

$$T_m = \{w \mid \mathrm{AVG}_{\gamma_d}(w) \geq 1 - s \text{ and } |w| = m\}.$$

Let $\mathscr{G}_{\text{alg}}(\Gamma) = \{s \in [0, \infty) \mid \text{there exists a learning function } l \text{ which algorithmically } s\text{-learns every } X \in \Gamma\}$.

We now prove our main theorem.

▶ **Theorem 13.** *For all* $\Gamma \subseteq \{0,1\}^\infty$*,*

$$\mathrm{adim}(\Gamma) = \inf \mathscr{G}_{\text{alg}}(\Gamma).$$

Proof. Let $s = \inf \mathscr{G}_{\text{alg}}(\Gamma)$. We will show $\dim(\Gamma) \leq s$.

Let $s' > s$ and let l be a learning function that algorithmically s'-learns every $X \in \Gamma$.

[6] Theorem 13.3.4 ([23], Theorem 3.1) states that $\dim(X) = \liminf_n \frac{C(X \upharpoonright n)}{n}$ where C is the *plain Kolmogorov complexity*.

Let $T = \cup T_n$ where $T_n = \{w \mid |w| = n \text{ and } \mathrm{AVG}_l(w) \geq 1 - s\}$. T is computably enumerable due to the first condition of s'-learnability.

For each n, $\lambda(\cup_{w \in T_n} C_w) \leq 2^{-(1-s')n}$ as a result of the measure condition of s'-learnability, since every sequence Y with at least $n(1-s)$ YES answers at length n is in the set of sequences which have at least $n(1-s)$ YES answers total, and the measure of the latter set is at most $2^{-(1-s')n}$. Thus, since each $w \in T_n$ has measure 2^{-n} and all are disjoint, we have $|T_n| \leq 2^{s'n}$. As a result, in plain Kolmogorov complexity terms, we only need to supply at most $s'n$ bits to identify an element of T living in T_n (we get n "for free" as long as we supply *exactly* $s'n$ bits, and use them to identify the slice T_n of T).

This means that for all $w \in T$, $C(w) \leq s'|w|$. Note that for every $X \in\ \Gamma$, there are infinitely many n so that $X \upharpoonright n \in T$.

It follows, then, that for every $X \in \Gamma$,

$$\dim(X) = \liminf_n \frac{C(X \upharpoonright n)}{n} \leq \frac{s'n}{n} = s',$$

and thus - since $s' > s$ was arbitrary - $\dim(\Gamma) \leq s$.

In the other direction, we show that if $\dim(\Gamma) \leq s$, then for every $s' > s$, Γ is algorithmically s'-learnable.

Again, let $s' > s$ and let d be a martingale which succeeds against order $2^{(1-s')n}$ on every $X \in \Gamma$. Define the learning function γ_d, as in Theorem 12 (Eq. 5.1), to say YES every time d doubles its money (attains a new 2^k value) for the first time along each path.

We have established that the values $\mathrm{AVG}_{\gamma_d}(w)$ are uniformly lower-semicomputable when d is lower-semicomputable. We also know from the proof of Theorem 12 that γ_d satisfies the measure condition and that $\limsup_{n\to\infty} \mathrm{AVG}(l, X \upharpoonright n) \geq 1 - s'$, for every $X \in \Gamma$. ◀

7 Strong algorithmic dimension via learning functions

In this section, we characterize the packing dimension as well as the strong algorithmic dimension of a set of binary sequences in terms of strong s-learning and strong algorithmic s-learning, respectively.

For any $\Gamma \subseteq \{0,1\}^\infty$, let $\Gamma \upharpoonright n = \{w \mid |w| = n$ and w is a prefix of some $v \in \Gamma\}$.

▶ **Definition** ([6]). *For any $\Gamma \subseteq \{0,1\}^\infty$, let the* upper box-counting dimension *of Γ be*

$$\overline{\dim_B(\Gamma)} = \limsup_n \frac{\log |\Gamma \upharpoonright n|}{n}.$$

▶ **Theorem 14** ([1]). *For every $X \in \{0,1\}^\infty$,*

$$\mathrm{Dim}(X) = \limsup_n \frac{C(X \upharpoonright n)}{n}.$$

Let $\mathscr{G}_{str}(\Gamma) = \{s \in [0,\infty) \mid$ there exists a learning function f which strongly s-learns every $X \in \Gamma\}$.

▶ **Theorem 15.** *Let $\Gamma \subseteq \{0,1\}^\infty$. Then,*

$$\dim_P(\Gamma) = \inf \mathscr{G}_{str}(\Gamma).$$

Proof. First, we show that if, for arbitrary $s' > s$, l is a learning function that strongly s'-succeeds on Γ, then $\dim_P(\Gamma) \leq s$.

The proof is much the same as [1] and [4] Theorem 13.11.9, except we start with a learning function instead of a gale. Assume the hypothesis. Let $\hat{s} > s'$ be arbitrary.

Let

$$T_n = \{w \mid \mathrm{AVG}_l(w) \geq 1 - \hat{s} \text{ and } |w| = n\}$$

be the set of strings of length n on which l achieves the requisite density. Then, for all $X \in \Gamma$, and for all but finitely many $n \in \mathbb{N}$, $X \in \cup_{w\in T_n} C_w$, i.e.

$$\Gamma \subseteq \cup_i \cap_{j\geq i} [\![T_j]\!].$$

Let $Z_n = \cap_{m \geq n} [\![T_m]\!]$. Then, $X \in Z_n$ means that after the nth prefix, all remaining prefixes of X have density at least $1 - \hat{s}$.

It suffices - by the countable stability of $\dim_P$ and the fact that $\dim_P \leq \overline{\dim_B}$ (see [6] 13.11.3) - to show that $\overline{\dim_B}(Z_n) \leq \hat{s}$, for all n. While the original proof uses Kolmogorov's inequality, we are not dealing with a gale. We have shown in the proof of Theorem 13 that if l is an $\hat{s}$-learner, then the set T_n satisfies $|T_n| \leq 2^{\hat{s}n}$.

Then,

$$\overline{\dim_B(Z_i)} = \limsup_n \frac{\log |Z_i \restriction n|}{n} \leq \limsup_n \frac{\log |T_n|}{n} \leq \hat{s}.$$

Since $\hat{s} > s' > s$ are arbitrary, $\dim_P(\Gamma) \leq \overline{\dim_B}(\Gamma) = \overline{\dim_B}(\cup Z_i) \leq s$.

In the other direction, simply note that if $\dim_P(\Gamma) \leq s$ then for every $s' > s$ there exists a martingale $d_{s'}$ which strongly s'-succeeds against order $2^{(1-s')n}$ on every $X \in \Gamma$, and the frequency with which the martingale doubles its money is always eventually bounded below. Thus, a learner defined as in Eq. 5.1 will have the desired YES density on the same sequences. ◀

Let $\mathscr{G}^{str}_{\text{alg}}(\Gamma) = \{s \in [0, \infty) \mid$ there exists a learning function l which strongly algorithmically s-learns every $X \in \Gamma\}$.

▶ **Theorem 16.** *Let* $\Gamma \subseteq \{0,1\}^\infty$. *Then,*

$$\text{aDim}(\Gamma) = \inf \mathscr{G}^{str}_{\text{alg}}(\Gamma).$$

Proof. Much like the gale characterization of aDim resembles the gale characterization of adim with minor changes (see [1] and [6], Corollary 13.11.12), this proof closely resembles the proof of Theorem 13.

We replace the lim inf with a lim sup in order to apply Theorem 14, and we replace the observation that infinitely many n satisfy $X \restriction n \in T$ with "all but finitely many."

In the other direction, we assume there exists a lower semi-computable martingale $d_{s'}$ which succeeds strongly against order $2^{(1-s')n}$ on every $X \in \Gamma$. A learning function defined in the same way as before (Eq. 5.1) will have the requisite density of YES answers, and will also have uniformly lower-semicomputable averages at each $w \in \{0,1\}^*$, as established in Section 6. ◀

References

1 Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM journal on computing*, 37(3):671–705, 2007.

2 Patrick Billingsley. *Probability and measure.* Wiley-Interscience, 1995.

3 Thomas R. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., 2006.

4 Rod Downey and Denis R Hirschfeldt. Algorithmic randomness. *Communications of the ACM*, 62(5):70–80, 2019.

5 Rod Downey and Denis R Hirschfeldt. Computability and randomness. *Notices of the American Mathematical Society*, 66(7):1001–1012, 2019.

6 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic Randomness and Complexity.* Springer Science & Business Media, 2010.

7 E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

8 F Hausdorff. Dimension und äußeres maß. *Mathematische Annalen*, 79:157–179, 1919.

9 John M Hitchcock. Gales suffice for constructive dimension. *Information Processing Letters*, 86(1):9–12, 2003.

10 Sanjay Jain, Daniel Osherson, James S Royer, Arun Sharma, et al. *Systems that Learn: An Introduction to Learning Theory*. MIT press, 1999.

11 Leon G Kraft. A device for quantizing, grouping and coding amplitude modified pulses. *Master's thesis, Cambridge, MA*, 1949.

12 Jack H Lutz. Gales and the constructive dimension of individual sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 902–913. Springer, 2000.

13 Jack H Lutz. Dimension in complexity classes. *SIAM Journal on Computing*, 32(5):1236–1259, 2003.

14 Jack H. Lutz. The dimensions of individual strings and sequences. *Information and Computation*, 187(1):49–79, 2003. doi:10.1016/S0890-5401(03)00187-1.

15 Jack H Lutz and Neil Lutz. Algorithmic information, plane Kakeya sets, and conditional dimension. *ACM Transactions on Computation Theory (TOCT)*, 10(2):1–22, 2018.

16 Jack H Lutz and Neil Lutz. Who asked us? How the theory of computing answers questions about analysis. In *Complexity and Approximation*, pages 48–56. Springer, 2020.

17 Jack H Lutz and Elvira Mayordomo. Algorithmic fractal dimensions in geometric measure theory. In *Handbook of Computability and Complexity in Analysis*, pages 271–302. Springer, 2021.

18 Jack H Lutz, Renrui Qi, and Liang Yu. The point-to-set principle and the dimensions of Hamel bases. *Computability*, 13(2):105–112, 2024. doi:10.3233/COM-210383.

19 Neil Lutz. Fractal intersections and products via algorithmic dimension. *ACM Transactions on Computation Theory (TOCT)*, 13(3):1–15, 2021.

20 Neil Lutz and Donald M Stull. Projection theorems using effective dimension. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, 2018.

21 Neil Lutz and Donald M Stull. Bounding the dimension of points on a line. *Information and Computation*, 275:104601, 2020.

22 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

23 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Information Processing Letters*, 84(1):1–3, 2002.

24 Daniel Osherson and Scott Weinstein. Recognizing strong random reals. *The Review of Symbolic Logic*, 1(1):56–63, 2008.

25 Daniel N Osherson, Michael Stob, and Scott Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. The MIT Press, 1986.

26 Claus-Peter Schnorr. A unified approach to the definition of random sequences. *Mathematical Systems Theory*, 5(3):246–258, 1971. doi:10.1007/BF01694181.

27 Theodore Slaman. Kolmogorov complexity and capacitability of dimension. Report No. 21/2021. Mathematisches Forschungsinstitut Oberwolfach, 2021.

28 Tomasz Steifer. A note on the learning-theoretic characterizations of randomness and convergence. *The Review of Symbolic Logic*, pages 1–16, 2021.

29 Dennis Sullivan. Entropy, Hausdorff measures old and new, and limit sets of geometrically finite Kleinian groups. *Acta Mathematica*, 153(1):259–277, 1984.

30 Claude Tricot. Two definitions of fractional dimension. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 91, pages 57–74. Cambridge University Press, 1982.

31 Jean Ville. *Etude critique de la notion de collectif*. Gauthier-Villars Paris, 1939.

32 Francesca Zaffora Blando. A learning-theoretic characterisation of Martin-Löf randomness and Schnorr randomness. *The Review of Symbolic Logic*, 14(2):531–549, 2021.

On the Complexity of the Conditional Independence Implication Problem with Bounded Cardinalities

Michał Makowski
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

We show that the conditional independence (CI) implication problem with bounded cardinalities, which asks whether a given CI implication holds for all discrete random variables with given cardinalities, is co-NEXPTIME-hard. The problem remains co-NEXPTIME-hard if all variables are binary. The reduction goes from a variant of the tiling problem and is based on a prior construction used by Cheuk Ting Li to show the undecidability of a related problem where the cardinality of some variables remains unbounded. The CI implication problem with bounded cardinalities is known to be in EXPSPACE, as its negation can be stated as an existential first-order logic formula over the reals of size exponential with regard to the size of the input.

2012 ACM Subject Classification Mathematics of computing → Information theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases Conditional independence implication, exponential time, tiling problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.73

Related Version *Full Version*: https://arxiv.org/abs/2408.02550 [10]

Acknowledgements The author would like to thank Damian Niwiński for his guidance and the anonymous reviewers for their valuable comments.

1 Introduction

The implication problem for conditional independence statements is one of the major decision problems arising in multivariate statistical modeling and other applications [2]. The problem asks whether a list of conditional independence statements implies another such statement (see the exact formulation below). The problem is a special case of the conditional entropic inequality problem, as the statement *X and Y are independent, given Z* (sometimes denoted $X \perp Y \mid Z$) is equivalent to the equation $I(X;Y|Z) = 0$ in information theory. Here the random variables in consideration are of the form $(X_{i_1}, \dots, X_{i_\ell})$, abbreviated by X_Z with $Z = \{i_1, \dots, i_\ell\}$, selected from a fixed n-tuple of variables $X_1, \dots, X_n$ considered with a joint distribution. As with the general problem, this can be considered over continuous, infinite discrete or finite discrete random variables. Furthermore, the CI implication problem can be refined by imposing certain requirements on the sets A, B, C in $X_A \perp X_B \mid X_C$, e.g., they must be pairwise disjoint for *disjoint CI*, and for *saturated CI* they must additionally satisfy $A \cup B \cup C = \{1, \dots, n\}$. We will focus on discrete random variables with a finite domain and without constraints on the sets, addressing disjoint CI in the full version.

If the domain size is bounded, this problem is decidable since the conditional independence can be expressed as an arithmetic formula in terms of elementary events' probabilities. Considering all possible domain sizes yields a semi-algorithm for finding a counter-example to the implication, showing that the unbounded problem is co-recursively enumerable (as noted by Khamis *et al* [5]). The decidability of the general CI implication problem was unknown for a long time, with only special cases resolved. Finally, Cheuk Ting Li published two papers [8, 9] proving the problem to be undecidable. This was also shown independently

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 73; pp. 73:1–73:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by Kühne and Yashfe [6]. Still unknown is the complexity of the bounded problem – the method of constructing an existential formula of Tarski's arithmetic yields an upper bound of EXPSPACE (cf. [1]), while the other published algorithms appear to be mostly heuristics. Hannula *et al* [4] conjectured that the problem can be actually easier, especially in the case where all variables are binary, as the arithmetic formula in question is of a very special form.

In this paper we show that the problem is in general co-NEXPTIME-hard, and the hardness result continues to hold if all variables are binary. Our reduction is an adaptation of the construction presented by Li [8] to show that the problem is undecidable if the cardinalities of some variables are bounded. While there is still a gap between the lower and upper bound, this shows that the complexity of the CI implication problem is harder than it might have been expected.

2 Problem statement

Denote by $\mathrm{card}(X)$ the cardinality of a random variable X. Formally, the following problem will be considered:

Bounded CI Implication

Input: Integers m, n, given in unary. A list of $m+1$ triples (A_i, B_i, C_i) of subsets of $\{1, \dots, n\}$. A list of n integers K_j, given in binary.

Question: Determine whether the implication

$$\bigwedge_{i \in \{1,\dots,m\}} (I(X_{A_i}; X_{B_i} | X_{C_i}) = 0) \Rightarrow I(X_{A_{m+1}}; X_{B_{m+1}} | X_{C_{m+1}}) = 0$$

holds for all jointly distributed random variables $(X_1, \dots, X_n)$ with $\mathrm{card}(X_j) \leq K_j$ for all $j \in \{1, \dots, n\}$.

We define Constant-bounded CI Implication as a variant of the above problem in which all K_i are fixed to be equal to 2 rather than given as input. We show the following:

▶ **Theorem 1.** *Bounded CI Implication and Constant-bounded CI Implication are co-NEXPTIME-hard. This also holds in the disjoint CI case, i. e. when for each i the sets A_i, B_i, C_i are pairwise disjoint.*

We focus on the first part of the theorem – the proof of the second part differs little from that given by Li [8] and is given in the full version.

In order to state the tiling-based problems utilized in the reduction, we introduce some definitions based on those in [7]. We define a *tiling system* as a triple $\mathcal{D} = (D, H, V)$, where D is a finite set of tiles and $H, V \subseteq D^2$ are the horizontal and vertical constraints, accordingly, which give the pairs of tiles that may be neighbors. This is a generalization of Wang tiles, where a set of colors C is given and tiles (formally quadruples from the set C^4) are represented by squares with colored edges with the requirement that only edges of the same color may touch. As stated in [12], Wang tiles correspond exactly to those tiling systems for which the implication $(a\,R\,b \wedge a\,R\,c \wedge d\,R\,b) \Rightarrow d\,R\,c$ holds for all $a, b, c, d \in D$ and both $R \in \{H, V\}$.

We define a $k \times l$ *tiling* by $\mathcal{D}$ as a function $f : \{0, \dots, k-1\} \times \{0, \dots, l-1\} \to D$ such that:

- $(f(m,n), f(m+1,n)) \in H$ for all $m < k-1, n < l$,
- $(f(m,n), f(m,n+1)) \in V$ for all $m < k, n < l-1$.

A *periodic tiling* is one that also has $(f(k-1,n), f(0,n)) \in H$ and $(f(m,l-1), f(m,0)) \in V$ for all $m < k, n < l$. For a (non-periodic) $k \times l$ tiling f, the *starting tile* and *final tile* are the values of $f(0,0)$ and $f(k-1,l-1)$, respectively.

We will show a polynomial-time many-one reduction from the following problem, which is known to be NEXPTIME-complete [7, exercise 7.2.2.]:

Binary Bounded Tiling
Input: A tiling system $\mathcal{D}$, a starting tile $d_0 \in D$, an integer k given in binary.
Question: Determine whether there exists a $k \times k$ tiling f by $\mathcal{D}$ such that $f(0,0) = d_0$.

The reduction consists of two parts, the first being a purely tiling-based reduction from the above to the following intermediate problem:

Periodic Bounded Tiling
Input: A tiling system $\mathcal{D}$, a designated tile t, integers m, n given in binary.
Question: Determine whether there exists a periodic tiling by $\mathcal{D}$ of size at most $m \times n$ which uses tile t.

The second part is a reduction from Periodic Bounded Tiling to the complement of Bounded CI Implication, based on a construction by Li [8].

3 First part of the reduction

We first show a polynomial-time many-one reduction from Binary Bounded Tiling to Periodic Bounded Tiling. This means that given a tiling system $\mathcal{D}$, starting tile d_0 and integer k, we will construct in polynomial time a tiling system $\mathcal{D}''$, designated tile t and integers m, n such that Binary Bounded Tiling gives a positive answer for input $(\mathcal{D}, d_0, k)$ iff Periodic Bounded Tiling gives a positive answer for input $(\mathcal{D}'', t, m, n)$. This consists of two steps:

1. Modify $\mathcal{D}$ into system $\mathcal{D}'$ such that valid tilings by $\mathcal{D}$ of size $k \times k$ correspond to valid tilings by $\mathcal{D}'$ with certain corner constraints.
2. Modify $\mathcal{D}'$ into system $\mathcal{D}''$ and tile t such that valid tilings by $\mathcal{D}'$ with the above corner constraints correspond to periodic tilings by $\mathcal{D}''$ of size $(k+1) \times (k+1)$ that use tile t.

This is a fairly typical reduction between tilings, similar to problems considered for instance in [3].

Limiting tiling size

For the first step, we create a tiling system $\mathcal{C}$ (of polynomial size with regard to the length of k), along with starting tile c_0 and final tile c_1, implementing a binary counter that counts down from an appropriately chosen $k' \leq k$ (close to k) and whose position shifts by 1 with each decrement. The tile c_1 occurs when the counter reaches 0. Similar constructions have been shown [11], our example is given in Figure 1. Thus, any tiling by $\mathcal{C}$ with c_0 in the top-right corner and c_1 in the bottom-left must be of size exactly $k \times k$.

Consider a "layering" of $\mathcal{D}$ and $\mathcal{C}$ into system $\mathcal{D} \times \mathcal{C} = (D \times C, H_{D \times C}, V_{D \times C})$, where the relations are defined as $R_{D \times C} = \{((d,c),(d',c')) : (d,d') \in R_D \wedge (c,c') \in R_C\}$ for both $R \in \{H, V\}$. This way, any tiling by $\mathcal{D} \times \mathcal{C}$ corresponds to a pair of tilings by $\mathcal{D}$ and $\mathcal{C}$. We let $\mathcal{D}' = \mathcal{D} \times \mathcal{C}$ and define the corner constraints mentioned in point 2 by restricting the possible starting and final tiles to $S = \{(d_0, c_0)\}, F = \{(d, c_1) : d \in D\}$ respectively. This yields tilings by $\mathcal{D}'$ which consist of a tiling by $\mathcal{D}$ with starting tile d_0 and a tiling by $\mathcal{C}$ with starting tile c_0 and final tile c_1.

Periodic tilings

The second step realizes the above constraint while also converting between periodic and non-periodic tilings. It consists of adding 5 new tiles to $\mathcal{D}'$ – ⊞, ⊟, ◫, ▲, ▼ – yielding $\mathcal{D}''$. The added constraints are shown in Figure 2 and an example tiling in Figure 3. The distinguished tile t is ⊞ – the idea is that its usage forces a "border" of ⊟ and ◫ tiles. Within each of these borders (there can be multiple if ⊞ is used more than once) there is a valid tiling by the system $\mathcal{D}'$ additionally satisfying the starting and final constraints S, F.

Final conversion

Combining the above steps, a tiling by $\mathcal{D}$ of size $k \times k$ is represented by a periodic $(k+1) \times (k+1)$ tiling by $\mathcal{D}''$ and thus we let $m = n = k + 1$. The designated tile is set to ⊞, completing the reduction from Binary Bounded Tiling to Periodic Bounded Tiling.

Variant with only powers of two

Consider the following variant of the tiling problem:

Power-of-two Periodic Bounded Tiling
Input: Integers m, n given in unary, a tiling system $\mathcal{D}$, a designated tile t.
Question: Determine whether there exists a periodic tiling by $\mathcal{D}$ of size at most $2^m \times 2^n$ which uses tile t.

Note that this is the same as taking the input in binary while restricting it only to powers of two. This variant is also NEXPTIME-hard because in the above reduction, the only possible sizes of tiling by the constructed tiling system are multiples of $k + 1$, both in width and height. Letting $m = n = \lceil \log_2(k + 1) \rceil$, we have $k + 1 \leq 2^m < 2(k + 1)$ and the same for 2^n. Therefore, the possible tilings are the same for size bound $(k + 1) \times (k + 1)$ and $2^{\lceil \log_2(k+1) \rceil} \times 2^{\lceil \log_2(k+1) \rceil}$.

4 Second part of the reduction

Given a tiling system $\mathcal{D}$, a designated tile t and integers m, n given in binary, we will construct (in polynomial time) a CI implication with bounded cardinalities which does not hold iff Periodic Bounded Tiling has a solution for input $(\mathcal{D}, t, m, n)$. This is based on the construction of Li [8]. Proofs which differ from the original only by specifying cardinality bounds are deferred to the full version. The main changes to Li's construction are as follows:

- provide bounds for the random variables used in the construction – this is done as the implication is being constructed;
- reduce the size of the implication from exponential to polynomial with regard to the input – only one part (the predicate COL) needs to be replaced by a polynomial-size equivalent;
- modify the representation of tiles to better suit bounded size and non-Wang tiles;
- add the requirement of the usage of a given tile – this is done by modifying the consequent of the implication.

4.1 Preliminaries

We denote by $\mathrm{Unif}(S)$ a uniform distribution over set S and by $\mathrm{Bern}(p)$ a Bernoulli distribution with parameter p. We use the shorthand X^k to represent a tuple of random variables $(X_1, \ldots, X_k)$, which is in itself also a random variable.

						$_1$	1_0	0_1	1
					$_1$	1_0	0_0	0	
				$_0$	0_1	1^*_1	1^*		
			$_0$	0_1	1_0	0			
		$_0$	0_0	0_1	1^*				
	$_0$	0_0	0_0	0					
$\star$	1^*_1	1^*_1	1^*						

Figure 1 A tiling implementation of a binary counter which shifts its position on every decrement, with the starting and final tile in the top-right and bottom-left corners respectively. This example counts down from 5 (101 in binary). Every tile except for the final $\star$ is of the form a^c_b, where a is the bit value (possibly blank), b is the value of the bit directly to the right (or blank if there is none), and c is optionally $*$ if a borrow operation is required. The shaded tiles of the top row function in the same manner, but they are "memorized" within the tiling system such that the placement of the top-right tile forces the top row to write out the binary initial value. The tiles in the lower rows are chosen deterministically based on their right and top neighbor. Finally, the $\star$ tile only occurs when the tile above is blank and the one to the right requires a borrow, which indicates that the counter has just gone below zero. In order to be unable to further count down, we disallow any tiles being below or to the left of tile $\star$. The size of the tiling is $(k' + b + 2) \times (k' + 2)$, where b is the number of bits and k' is the initial value; however, this could be modified such that the final tiling has size $(k' + b + 2) \times (k' + b + 2)$ by padding with b dummy rows at the top. For sufficiently large k, we can always efficiently find b, k' such that $k' + b + 2 = k$.

		Right					
		⊞	⊟	◫	▲	▼	*
Left	⊞				●		
	⊟		●			●	
	◫						●
	▲		●				
	▼	●					
	*			●			

		Top							
		⊞	⊟	◫	▲	▼	s	f	*
Bottom	⊞			●					
	⊟						●	●	●
	◫	●		●					
	▲						●		
	▼						●	●	●
	s		●		●				
	f		●		●	●			
	*		●		●				

Figure 2 Modified adjacency relation for the system $\mathcal{D}''$ – only adjacencies marked by ● are permitted, as well as all adjacencies from the original tiling system. The asterisk denotes any tiles from the system $\mathcal{D}'$, while s, f represent any tile from the initial and final subset of tiles, respectively (S and F defined above).

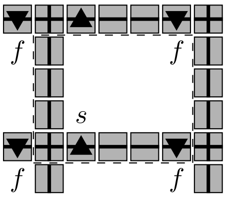


Figure 3 An example "border" created by the above tiling, with tiles from the original set not shown and s, f representing tiles from S, F respectively. The dashed rectangle represents the actual rectangle being tiled, while the tiles outside are periodic copies added to better illustrate the construction.

The *entropy* of a finite discrete random variable X with domain $\mathcal{X}$ is defined as

$$H(X) = -\sum_{x\in\mathcal{X}} \mathbf{P}(X = x)\log \mathbf{P}(X = x).$$

The *entropy of X conditioned on Y*, with X, Y finite discrete random variables with domains $\mathcal{X}, \mathcal{Y}$ respectively, is defined as

$$H(X|Y) = -\sum_{x\in\mathcal{X},y\in\mathcal{Y}} \mathbf{P}(X = x \wedge Y = y)\log \frac{\mathbf{P}(X = x \wedge Y = y)}{\mathbf{P}(X = x)}.$$

Finally, the *conditional mutual information of X and Y given Z* can be defined as

$$I(X;Y|Z) = H(X|Z) - H(X|Y,Z).$$

Similarly, the *mutual information* of X and Y is defined as

$$I(X;Y) = H(X) - H(X|Y).$$

Note that the conditional independence (CI) statement $I(X;Y|Z) = 0$ is equivalent to the fact that X and Y are independent given Z, and thus can be expressed without the usage of logarithms as

$$\mathbf{P}(X = x \wedge Y = y \wedge Z = z)\mathbf{P}(Z = z) = \mathbf{P}(X = x \wedge Z = z)\mathbf{P}(Y = y \wedge Z = z)$$

for all $x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z}$. Further, the functional dependence statement $H(X|Y) = 0$, which states that for any $y \in \mathcal{Y}$, there exists exactly one $x \in \mathcal{X}$ such that $\mathbf{P}(X = x \wedge Y = y) \neq 0$ can be expressed equivalently as a (non-disjoint) CI statement $I(X;X|Y) = 0$.

We will follow Li's construction [8], providing cardinality bounds and modifications where necessary. While the final goal is a CI implication, we will mostly construct *affine existential information predicates* (AEIP) [8], converting to a CI implication at the end. We will only consider a special form of AEIP which consists of an existentially quantified conjunction of CI statements. This family of predicates is closed under conjunction, in particular we can use a predicate within the definition of another predicate (implicitly renaming variables in the case of a naming conflict).

Since our goal is to construct a bounded CI implication, every quantified variable will be given a *cardinality bound* – the maximum allowed size of its domain. We denote the existential quantification of a variable X with $\mathrm{card}(X) \leq k$ by the shorter notation $\exists X \leq k$, similarly for tuple variables $\exists X^2 \leq k$ represents the existence of variables X_1, X_2 with $\mathrm{card}(X_1), \mathrm{card}(X_2) \leq k$. Whenever a predicate takes arguments, their cardinalities are already bounded since they have been quantified. We may need to refer to these bounds when quantifying new variables, denoting by K_X the bound already given to variable X. Finally, whenever $\leq$ is replaced by $\leq_i$, this indicates an "implicit" bound, that is one which does not change the meaning of the predicate because it is already satisfied by any such quantified variable even without the explicit bound. An example of this is that whenever $H(X|Y) = 0$, X is functionally dependent on Y and so $X \leq_i K_Y$.

The first defined predicate is TRIPLE:

$$\begin{aligned}\mathrm{TRIPLE}(Y_1, Y_2, Y_3) : {}& H(Y_1|Y_2, Y_3) = H(Y_2|Y_1, Y_3) = H(Y_3|Y_1, Y_2) = 0\\ &\wedge I(Y_1; Y_2) = I(Y_1; Y_3) = I(Y_2; Y_3) = 0.\end{aligned}$$

By definition, predicate TRIPLE of three variables Y_1, Y_2, Y_3 is satisfied iff Y_1, Y_2, Y_3 are pairwise independent and each variable is functionally dependent on the other two. Functional dependency is a special case of conditional independence, since $H(X|Y) = I(X;X|Y)$. The following is shown in [13]:

▶ **Lemma 1.** *If* $\mathrm{TRIPLE}(X,Y,Z)$ *is satisfied, then* X,Y,Z *are all uniformly distributed and have the same cardinality.*

This is used in the next predicate, UNIF:

$$\mathrm{UNIF}(X) : \exists U_1 \leq_i K_X, U_2 \leq_i K_X : \mathrm{TRIPLE}(X,U_1,U_2).$$

By definition, predicate UNIF of one variable X is satisfied iff there exist discrete random variables U_1, U_2 jointly distributed with X such that $\mathrm{TRIPLE}(X,U_1,U_2)$ holds, which in turn is equivalent to X being uniformly distributed over its support. Lemma 1 immediately shows that the implicit cardinality bounds for U_1, U_2 are correct.

For any constant k, predicate $\mathrm{UNIF}_k(X)$ is defined to imply X being uniformly distributed over a domain of size k:

$$\mathrm{UNIF}_k(X) : \mathrm{UNIF}(X) \wedge \alpha_k \leq H(X) \leq \alpha_{k+1},$$

where $\alpha_k \in \mathbb{Q}$ is some rational with $\log(k-1) < \alpha_k < \log k$ (because the entropy of a uniform variable is the logarithm of the cardinality of its support). While this is still a valid AEIP, it is not a conjunction of CI statements. However, in the bounded cardinality setting UNIF_k can be restated in this form [8, 10]. Note that this predicate imposes an exact domain size constraint, while the cardinality bounds given as input provide only an upper bound. Finally, note that the predicates UNIF, UNIF_k are satisfiable – for any $k > 0$, there exists a variable X which satisfies $\mathrm{UNIF}_k(X)$ and so $\mathrm{UNIF}(X)$.

Define the *characteristic bipartite graph* of random variables X_1, X_2 with (disjoint) supports $\mathcal{X}_1, \mathcal{X}_2$ as the undirected graph with set of vertices $V = \mathcal{X}_1 \cup \mathcal{X}_2$ and set of edges $E = \{(x_1,x_2) : x_1 \in \mathcal{X}_1, x_2 \in \mathcal{X}_2, \mathbf{P}(X_1 = x_1 \wedge X_2 = x_2) > 0\}$.

Li constructs the predicate

$$\begin{aligned}\mathrm{CYCS}(X_1,X_2) : \exists U \leq_i 2 : &\ \mathrm{UNIF}(X_1) \wedge \mathrm{UNIF}(X_2) \wedge \mathrm{UNIF}_2(U)\\ &\wedge I(X_1;U) = I(X_2;U) = 0\\ &\wedge H(X_1|X_2,U) = H(X_2|X_1,U) = 0\\ &\wedge H(U|X_1,X_2) = 0\end{aligned}$$

and shows the following (without cardinality bounds):

▶ **Lemma 2.** $\mathrm{CYCS}(X_1,X_2)$ *is satisfied iff* X_1, X_2 *are uniform and the characteristic bipartite graph of* X_1, X_2 *consists only of vertex-disjoint simple cycles.*

Furthermore, this predicate is satisfiable – for any finite collection of even-length cycles, we can clearly find X_1, X_2 such that their characteristic bipartite graph consists exactly of this collection of cycles.

4.2 Overview of the construction

We now give an overview of the following steps of the construction. Section 4.3 defines the predicate $\mathrm{TORI}'(X^2,Y^2,Z)$, which enforces that the characteristic bipartite graph of X_1 and X_2 is a collection of cycles, similarly for Y_1 and Y_2. Finally, we require that Z be distributed uniformly over two values and that the three variables X^2, Y^2, Z be independent. The distribution of (X^2,Y^2) then represents a collection of tori, with each quadruple of values of (X_1,X_2,Y_1,Y_2) representing a vertex in some torus. The addition of variable Z effectively creates a corresponding copy of this collection.

With the goal of creating meaningful labels to be applied to the vertices of the aforementioned graph, which are represented by k-tuple of binary variables W^k, Section 4.4 defines the predicates $\mathrm{SW}'(W^k, V^k, \bar{V}^k, F)$ and $\mathrm{COL}'(W^k, V^k, \bar{V}^k, F)$. The former ensures that $V_i = (1 - W_i)F, \bar{V}_i = W_i F$ (up to relabeling) with the side-effect of requiring each $W_i \sim \mathrm{Bern}(\frac{1}{2})$. The latter predicate restricts W^k such that the only values that are possible have either $W_k = 1$ and exactly one $W_i = 0$, or $W_k = 0$ and exactly one $W_i = 1$, for some $i \in \{1, \ldots, k-1\}$.

Section 4.5 combines the prior predicates in predicate $\mathrm{CTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F)$ in such a way that each vertex is assigned exactly one label, i. e. W^k depends functionally on (X^2, Y^2, Z). This is extended in predicate $\mathrm{OTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F)$, which assigns four possible *groups* to vertex labels and enforces a structure as shown in Figure 4.

Finally, the predicate $\mathrm{TTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F)$ defined in Section 4.7 restricts the possible labels of vertices connected by edges so as to enforce the given vertical and horizontal constraints of the given tile system.

4.3 Grid

A collection of tori is constructed by Li using the following predicate (where X^2 represents a pair of variables (X_1, X_2), similarly for Y^2):

$$\mathrm{TORI}(X^2, Y^2) : \mathrm{CYCS}(X^2) \wedge \mathrm{CYCS}(Y^2) \wedge I(X^2; Y^2) = 0,$$

When the above holds, fixing any three of the variables X_1, X_2, Y_1, Y_2 leaves two possible values of the fourth. Similarly, fixing one variable from X_1, X_2 and one from Y_1, Y_2 gives the remaining two variables a distribution over four values. This is visualized by a graph similar in idea to the bipartite characteristic graph: its vertices are quadruples of values (x_1, x_2, y_1, y_2) which satisfy $\mathbf{P}(X_1 = x_1 \wedge X_2 = x_2 \wedge Y_1 = y_1 \wedge Y_2 = y_2) > 0$, with edges connecting any two quadruples which differ in exactly one out of these four values. When arranged in a grid with possible pairs (X_1, X_2) on one axis and (Y_1, Y_2) on the other, as in Figure 4, the torus structure becomes apparent. Again, this predicate is satisfiable in the sense that any collection of tori which is a product of two collections of even-length cycles has a representation by random variables X^2, Y^2.

Our construction departs slightly from the construction of Li, adding another coordinate Z, corresponding to taking two copies of the collection of tori, with the edges and faces described above preserved when Z is fixed. Additionally, fixing X^2 and Y^2 but not Z "connects" two corresponding vertices in the two copies. The predicate for this is as follows:

$$\begin{aligned}\mathrm{TORI}'(X^2, Y^2, Z) : {} & \mathrm{CYCS}(X^2) \wedge \mathrm{CYCS}(Y^2) \wedge \mathrm{UNIF}_2(Z) \\ & \wedge I(X^2, Y^2; Z) = 0 \wedge I(X^2, Z; Y^2) = 0 \wedge I(Y^2, Z; X^2) = 0.\end{aligned}$$

4.4 Vertex labels

The basis for constructing labels which will later on be assigned to vertices is the following predicate defined by Li:

$$\begin{aligned}\mathrm{FLIP}(F, G_1, G_2) : \exists U \leq_i 4, Z^2 \leq_i 3 : {} & \mathrm{UNIF}_4(U) \wedge \mathrm{UNIF}_2(F) \\ & \wedge H(F, G_1, G_2 | U) = I(G_1; G_2 | F) = 0 \\ & \wedge \mathrm{UNIF}_3(Z_1) \wedge I(Z_1; G_1) = H(U | G_1, Z_1) = 0 \\ & \wedge \mathrm{UNIF}_3(Z_2) \wedge I(Z_2; G_2) = H(U | G_2, Z_2) = 0.\end{aligned}$$

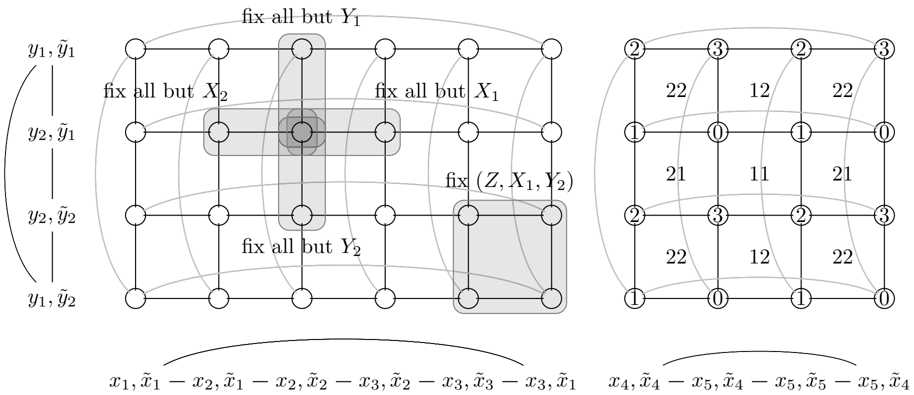


Figure 4 Visualization of the tori which are a product of the cycles created by (X_1, X_2) and (Y_1, Y_2), with each vertex corresponding to a quadruple of the values of (X_1, X_2, Y_1, Y_2). The axes show these cycles – a quadruple's (X_1, X_2) (resp. (Y_1, Y_2)) values are determined by projecting onto the horizontal (resp. vertical) axis. Additionally, the left torus shows highlighted edges which arise when all but one variable (of X_1, X_2, Y_1, Y_2, Z) are fixed as well as an example face which arises when Z and two other variables are fixed. The right torus has each face labeled with its type and each vertex labeled with its group – these are used in Section 4.6 in order to restrict allowed labelings of the vertices. The second "corresponding" torus and the Z axis are omitted for clarity.

Recalling that $\mathrm{Unif}(S)$ denotes a uniform distribution over set S, the following is shown:

▶ **Lemma 3.** $\mathrm{FLIP}(F, G_1, G_2)$ *is satisfied iff, up to relabeling,* (F, G_1, G_2) *has the distribution Unif*$(\{(0,0,0), (0,1,0), (1,0,0), (1,0,1)\})$.

For any $k \geq 4$, Li defines the predicate SW which allows us to represent strings of k bits. Here, W^k represents a tuple of variables $(W_1, \ldots, W_k)$, same for $V^k, \bar{V}^k$: Intuitively, the value of W_i determines whether F is copied into V_i or $\bar{V}_i$, hence the name SW for *switch*.

$$\begin{aligned}\mathrm{SW}(W^k, V^k, \bar{V}^k, F) : \exists G \leq_i 2 : I(W^k; F, G) = 0 \\ \wedge \bigwedge_{i \in \{1,\ldots,k\}} \big(\mathrm{UNIF}_2(W_i) \wedge H(V_i, \bar{V}_i | W_i, F) = I(V_i; \bar{V}_i | W_i) = 0 \\ \wedge \mathrm{FLIP}(F, G, V_i) \wedge \mathrm{FLIP}(F, G, \bar{V}_i)\big)\end{aligned}$$

▶ **Lemma 4.** *If* $\mathrm{SW}(W^k, V^k, \bar{V}^k, F)$ *is satisfied, then we have (without loss of generality)* $V_i = (1 - W_i)F, \bar{V}_i = W_i F$ *for all* i.

The predicate SW is satisfiable: we let (F, G) take each of the values $(0,0)$, $(0,1)$ with probability $\frac{1}{4}$, in which case we let $V_i = \bar{V}_i = 0$, and the value $(1,0)$ with probability $\frac{1}{2}$, in which case $V_i = 1 - W_i, \bar{V}_i = W_i$. As long as each $W_i \sim \mathrm{Bern}(\frac{1}{2})$ (recall that $\mathrm{Bern}(p)$ denotes a Bernoulli distribution with parameter p), we can satisfy the predicate for any distribution of W^k. This predicate additionally has the following property:

▶ **Lemma 5.** *If* $\mathrm{SW}(W^k, V^k, \bar{V}^k, F)$ *is satisfied, then for any* $S, \bar{S} \subseteq \{1, \ldots, k\}$ *we have*

$$H(F | V_S, \bar{V}_{\bar{S}}, W^k) = \mathbf{P}(sat(W^k, S, \bar{S}) = 1),$$

where

$$sat(w^k, S, \bar{S}) = \Big(\prod_{i \in S} w_i\Big)\Big(\prod_{i \in \bar{S}} (1 - w_i)\Big),$$

i.e. if $w_i = 1$ *for all* $i \in S$ *and* $w_i = 0$ *for* $i \in \bar{S}$ *then* $sat(w^k, S, \bar{S})$ *equals 1 and otherwise it equals 0.*

This immediately yields the following:

▶ **Lemma 6.** *The equality* $H(F|V_{\{i\in\{1,\dots,k\}\,:\,w_i=1\}}, \bar{V}_{\{i\in\{1,\dots,k\}\,:\,w_i=0\}}, W^k) = \mathbf{P}(W^k = w^k)$ *holds for any* $w^k \in \{0,1\}^k$. ⌟

Using these properties, we can disallow certain values of W^k from occurring using a CI statement. This is used by Li to limit the possible values of W^k to the set $T_k \subseteq \{0,1\}^k$, which consists of $2(k-1)$ *labels* (referred to as *colors* by Li), each one with a *value* and *sign*. The set consists of strings in which exactly one bit differs from the last bit, that is for any $w^k \in T_k$ we have $w_j \neq w_k$ for exactly one $j \neq k$. The sign of the label is determined by w_k – negative when $w_k = 1$, positive when $w_k = 0$ – and j is the value of the label. For example, the elements of $T_4 = \{0111, 1011, 1101, 1000, 0100, 0010\}$ correspond in order to labels $\{-1, -2, -3, +1, +2, +3\}$.

Li's predicate for enforcing this is simple:

$$\begin{aligned}\mathrm{COL}(W^k, V^k, \bar{V}^k, F):\ & \mathrm{SW}(W^k, V^k, \bar{V}^k, F)\\ \wedge & \bigwedge_{w^k\in\{0,1\}^k\setminus T_k} \big(H(F|V_{\{i:w_i=1\}}, \bar{V}_{\{i:w_i=0\}}, W^k) = 0\big),\end{aligned}$$

This predicate simply disallows the occurrence of any $w^k \notin T_k$, hence we have

▶ **Lemma 7.** *If* $\mathrm{COL}(W^k, V^k, \bar{V}^k, F)$ *is satisfied, then* $\mathbf{P}(W^k \notin T_k) = 0$. ⌟

While sufficient for showing undecidability, this predicate cannot be used in a polynomial-time reduction due to its size being exponential with regard to k. However, an equivalent polynomial-size predicate can be easily constructed: with

$$\bigwedge_{\substack{i,j\in\{1,\dots,k-1\},\\ i<j}} (H(F|V_{\{i,j\}}, \bar{V}_{\{k\}}, W^k) = 0),$$

we disallow all w^k such that $w_k = 0$ and $w_i = w_j = 1$ for some $1 \leq i < j < k$. Similarly,

$$\bigwedge_{\substack{i,j\in\{1,\dots,k-1\},\\ i<j}} (H(F|V_{\{k\}}, \bar{V}_{\{i,j\}}, W^k) = 0)$$

disallows all w^k such that $w_k = 1$ and $w_i = w_j = 0$ for some $1 \leq i < j < k$. The only remaining w^k have either $w_k = 0$ and at most one 1 in $\{1,\dots,k-1\}$ or have $w_k = 1$ and at most one 0 in $\{1,\dots,k-1\}$. The strings 0^k and 1^k are disallowed by $H(F|V_{\{1,\dots,k\}}, \bar{V}_\varnothing, W^k) = 0$ and $H(F|V_\varnothing, \bar{V}_{\{1,\dots,k\}}, W^k) = 0$. Combined, these yield the polynomial-size predicate

$$\begin{aligned}\mathrm{COL}'(W^k, V^k, \bar{V}^k, F) :\ & \mathrm{SW}(W^k, V^k, \bar{V}^k, F)\\ \wedge & \bigwedge_{\substack{i,j\in\{1,\dots,k-1\},\\ i<j}} (H(F|V_{\{i,j\}}, \bar{V}_{\{k\}}, W^k) = 0)\\ \wedge & \bigwedge_{\substack{i,j\in\{1,\dots,k-1\},\\ i<j}} (H(F|V_{\{k\}}, \bar{V}_{\{i,j\}}, W^k) = 0)\\ \wedge\ & H(F|V_{\{1,\dots,k\}}, \bar{V}_\varnothing, W^k) = 0\\ \wedge\ & H(F|V_\varnothing, \bar{V}_{\{1,\dots,k\}}, W^k) = 0\end{aligned}$$

equivalent to the exponential-size COL.

4.5 Edge constraints

The next predicate is the following, with COL used in place of COL′ in Li's original construction:

$$\begin{aligned}\mathrm{COLD}(X, W^k, V^k, \bar{V}^k, F) : &\ \mathrm{COL}'(W^k, V^k, \bar{V}^k, F)\\ &\wedge H(W^k|X) = I(V^k, \bar{V}^k, F; X|W^k) = 0.\end{aligned}$$

This predicate will represent the labeling of vertices, with X representing the coordinate and W^k (which depends functionally on X) representing its label. For any $x \in \mathcal{X}$, denote by $w^k(x)$ the unique value of w^k which satisfies $\mathbf{P}(W^k = w^k|X = x) > 0$.

Suppose that $\mathrm{COLD}(X, W^k, V^k, \bar{V}^k, F)$ is satisfied and let E be any random variable which *splits X into sets of (a constant) size l* – we say this is the case when $H(E|X) = 0$ and $X|E = e$ is uniform over l values for all $e \in \mathcal{E}$. This is not verified by a predicate; rather, we will only choose E which have this property by definition. Fixing subsets of indices $S, \bar{S} \subseteq \{1, \dots, k\}$, we define for any $e \in \mathcal{E}$ the value a_e:

$$a_e = |\{x : \mathbf{P}(X = x|E = e) > 0 \wedge \mathrm{sat}(w^k(x), S, \bar{S}) = 1\}|$$

In order to impose restrictions on the possible values of a_e, Li defines the following predicates:

$$\begin{aligned}\mathrm{SAT}_{\neq 1/2,S,\bar{S}}(E, W^k, V^k, \bar{V}^k, F) : &\ \exists U \leq_i 2 : \mathrm{UNIF}_2(U) \wedge I(U; E, V_S, \bar{V}_{\bar{S}}) = 0\\ &\wedge H(F|V_S, \bar{V}_{\bar{S}}, E, U) = 0,\\ \mathrm{SAT}_{\leq 1/2,S,\bar{S}}(E, W^k, V^k, \bar{V}^k, F) : &\exists U \leq_i 3 : \mathrm{UNIF}_3(U) \wedge I(U; E, V_S, \bar{V}_{\bar{S}}) = 0\\ &\wedge H(F|V_S, \bar{V}_{\bar{S}}, E, U) = 0,\\ \mathrm{SAT}_{\leq 3/4,S,\bar{S}}(E, W^k, V^k, \bar{V}^k, F) : &\exists U \leq_i 105 : \mathrm{UNIF}_{105}(U) \wedge I(U; E, V_S, \bar{V}_{\bar{S}}) = 0\\ &\wedge H(F|V_S, \bar{V}_{\bar{S}}, E, U) = 0.\end{aligned}$$

The predicates satisfy the following properties.

▶ **Lemma 8.** *If* $\mathrm{COLD}(X, W^k, V^k, \bar{V}^k, F)$ *is satisfied and E splits X into sets of size 2, then* $\mathrm{SAT}_{\neq 1/2,S,\bar{S}}(E, W^k, V^k, \bar{V}^k, F)$ *is satisfied iff $a_e \neq 1$ for all $e \in \mathcal{E}$.*

▶ **Lemma 9.** *If* $\mathrm{COLD}(X, W^k, V^k, \bar{V}^k, F)$ *is satisfied and E splits X into sets of size 2, then* $\mathrm{SAT}_{\leq 1/2,S,\bar{S}}(E, W^k, V^k, \bar{V}^k, F)$ *is satisfied iff $a_e \leq 1$ for all $e \in \mathcal{E}$.*

▶ **Lemma 10.** *If* $\mathrm{COLD}(X, W^k, V^k, \bar{V}^k, F)$ *is satisfied and E splits X into sets of size 4, then* $\mathrm{SAT}_{\leq 3/4,S,\bar{S}}(E, W^k, V^k, \bar{V}^k, F)$ *is satisfied iff $a_e \leq 3$ for all $e \in \mathcal{E}$.*

The next defined predicate is the following:

$$\begin{aligned}\mathrm{CTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F) &: \mathrm{TORI}'(X^2, Y^2, Z)\\ &\wedge \mathrm{COLD}((X^2, Y^2, Z), W^k, V^k, \bar{V}^k, F),\end{aligned}$$

which simply implies applying labels (without any constraints) to the vertices of the tori. In the original predicate, which uses TORI instead of TORI′, there is no coordinate Z. Clearly, this predicate is satisfiable in the sense that any collection of pairs of tori of even size which is labeled in a manner that satisfies the requirement $W_i \sim \mathrm{Bern}(\frac{1}{2})$ has a corresponding representation by $X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F$. In particular, this W_i requirement is satisfied by any labeling in which any pair of corresponding vertices (those which differ only in the Z

coordinate) has labels of the same value but opposite sign. This is because negating the sign of a label corresponds to negating all of its bits. For a set of labels $\mathcal{L} = \{0, \dots, l-1\}$, we now label the vertices with labels from the set $\{0, \dots, 4l-1\}$ and so we set $k = 4l+1$. For any $i \in \{0, \dots, l\}$, $j \in \{0, \dots, 3\}$, we identify all four labels $4i+j$ with the original label i, referring to any vertex whose label is $4i+j$ as a *group j vertex*. The group of a vertex is used to orient it relative to its neighbors – this is achieved by the following predicate:

$$\begin{aligned}
&\mathrm{OTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F):\\
&\ \mathrm{CTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F)\\
&\wedge\ \mathrm{SAT}_{\neq 1/2,\{k\},\varnothing}((X_1, X_2, Y_1, Z), W^k, V^k, \bar{V}^k, F)\\
&\wedge\ \mathrm{SAT}_{\neq 1/2,\{k\},\varnothing}((X_1, X_2, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\wedge\ \mathrm{SAT}_{\neq 1/2,\{k\},\varnothing}((X_1, Y_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\wedge\ \mathrm{SAT}_{\neq 1/2,\{k\},\varnothing}((X_2, Y_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\wedge \bigwedge_{j_1, j_2 \in J_1} \big(\mathrm{SAT}_{\leq 1/2,\varnothing,\{1,\dots,k\}\setminus\{j_1,j_2\}}((X_1, X_2, Y_1, Z), W^k, V^k, \bar{V}^k, F)\\
&\qquad\qquad \wedge \mathrm{SAT}_{\leq 1/2,\varnothing,\{1,\dots,k\}\setminus\{j_1,j_2\}}((X_1, X_2, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\qquad\qquad \wedge \mathrm{SAT}_{\leq 1/2,\{1,\dots,k\}\setminus\{j_1,j_2\},\varnothing}((X_1, X_2, Y_1, Z), W^k, V^k, \bar{V}^k, F)\\
&\qquad\qquad \wedge \mathrm{SAT}_{\leq 1/2,\{1,\dots,k\}\setminus\{j_1,j_2\},\varnothing}((X_1, X_2, Y_2, Z), W^k, V^k, \bar{V}^k, F)\big)\\
&\wedge \bigwedge_{j_1, j_2 \in J_2} \big(\mathrm{SAT}_{\leq 1/2,\varnothing,\{1,\dots,k\}\setminus\{j_1,j_2\}}((X_1, Y_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\qquad\qquad \wedge\ \mathrm{SAT}_{\leq 1/2,\varnothing,\{1,\dots,k\}\setminus\{j_1,j_2\}}((X_2, Y_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\qquad\qquad \wedge\ \mathrm{SAT}_{\leq 1/2,\{1,\dots,k\}\setminus\{j_1,j_2\},\varnothing}((X_1, Y_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\\
&\qquad\qquad \wedge\ \mathrm{SAT}_{\leq 1/2,\{1,\dots,k\}\setminus\{j_1,j_2\},\varnothing}((X_2, Y_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\big)\\
&\wedge\ \mathrm{SAT}_{\leq 1/2,\{k\},\varnothing}((X_1, X_2, Y_1, Y_2), W^k, V^k, \bar{V}^k, F)\\
&\wedge\ \mathrm{SAT}_{\leq 1/2,\varnothing,\{k\}}((X_1, X_2, Y_1, Y_2), W^k, V^k, \bar{V}^k, F),
\end{aligned}$$

where

$$\begin{aligned}
J_1 &= \{(j_1, j_2) \in \{1, \dots, k-1\} : \{j_1 \bmod 4, j_2 \bmod 4\} \notin \{\{0,1\},\{2,3\}\}\},\\
J_2 &= \{(j_1, j_2) \in \{1, \dots, k-1\} : \{j_1 \bmod 4, j_2 \bmod 4\} \notin \{\{1,2\},\{0,3\}\}\}.
\end{aligned}$$

The only difference between OTORI' and Li's original OTORI is the added variable Z and the final two $\mathrm{SAT}_{\leq 1/2}$ predicates. We have the following fact:

▶ **Lemma 11.** *If* $\mathrm{OTORI}\,'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F)$ *is satisfied, then the following statements hold:*

1. *within each torus, all vertices' labels have the same sign;*
2. *any two vertices differing only in the Z coordinate have opposite sign;*
3. *any pair of vertices connected by a vertical edge either has groups 1 and 0 or 2 and 3;*
4. *any pair of vertices connected by a horizontal edge either has groups 1 and 2 or 3 and 0.*

Proof. Recall that fixing the variable Z along with any three of the variables X_1, X_2, Y_1, Y_2 leaves two possible values for the remaining variable. These correspond to two vertices of an edge, as illustrated in Figure 4. Therefore, the first four $\mathrm{SAT}_{\neq 1/2,\{k\},\varnothing}$ predicates state that for any edge (u, v), the value of w_k for u and v cannot differ, which implies exactly Point 1 above. Point 2 follows directly from the last two $\mathrm{SAT}_{\leq 1/2}$ predicates – of the two vertices which differ only in the Z coordinate, at most one can have $w_k = 0$ and at most one can have $w_k = 1$.

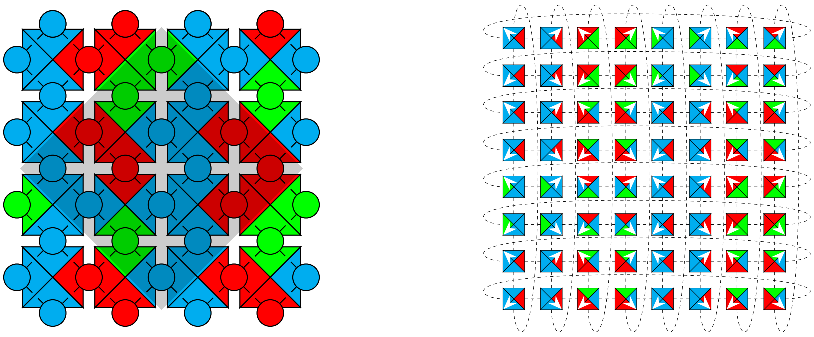


Figure 5 Left: Li's representation of a 4 × 4 torus coloring and the 4 × 4 tiling that it yields. The conversion from 16 vertices to 32 gives the tiling an additional diagonal periodicity. Combined with the fact that this does not work for non-square $n \times m$ tilings (without arranging them into a $\operatorname{lcm}(n, m) \times \operatorname{lcm}(n, m)$ square), this proves problematic for restricting the size of a periodic tiling. Right: the corresponding 8 × 8 torus labeling in our representation. Each tile has 4 labels, one for each corner of the tile (indicated by the arrow in this case). The tiles do not need to be Wang tiles.

For the next two points, note that for $j_1, j_2 \neq k$, we have $\operatorname{sat}(w^k, \{1, \dots, k\} \setminus \{j_1, j_2\}, \varnothing) = 1$ iff w^k represents one of the labels $\{-j_1, -j_2\}$ – for positive labels, we have $w_k = 0$ and for negative labels other than $-j_1, -j_2$, we have $w_i = 0$ for some $i \notin \{j_1, j_2\}$. Analogously, $\operatorname{sat}(w^k, \varnothing, \{1, \dots, k\} \setminus \{j_1, j_2\}) = 1$ iff w^k represents one of $\{+j_1, +j_2\}$. Thus, the four $\mathrm{SAT}_{\leq 1/2, \varnothing, \{1,\dots,k\} \setminus \{j_1, j_2\}}$ predicates within the conjunction over $j_1, j_2 \in J_1$ imply exactly Point 3, with the conjunction over $j_1, j_2 \in J_2$ implying Point 4 analogously. Clearly, OTORI$'$ is satisfiable – an example torus (with coordinate Z omitted) is shown in Figure 4. ◀

4.6 Tiles

Li denotes the set of four possible vertices when (Z, X_i, Y_j) is fixed as a *type ij face* for any $i, j \in \{1, 2\}$ – e. g. fixing (Z, X_1, Y_2) yields a type 12 face. The relation between face types and vertex groups is illustrated in Figure 4.

Li's construction utilizes the fact that the four corner-neighbors of any type 11 face are type 22 faces and vice versa. Because the original construction makes use of a Wang tiling system, these connecting corner vertices can represent the edge colors of the touching tiles. An example is shown in Figure 5. The diagonal nature of this tiling proves problematic when we wish to restrict its size, thus we simply use faces (of type 11) to directly represent tiles, which also allows us to use the general form of tiling systems. Figure 5 illustrates a torus labeling in our representation. For given tiling system $\mathcal{D} = (D, H, V)$, we define the following sets:

$$\begin{aligned}
\mathcal{D}_{11} &= \{(4t+1, 4t+2, 4t+3, 4t) : t \in D\},\\
\mathcal{D}_{12} &= \{(4v, 4v+3, 4u+2, 4u+1) : (u, v) \in V\},\\
\mathcal{D}_{21} &= \{(4u+2, 4u+1, 4v, 4v+3) : (u, v) \in H\},
\end{aligned}$$

and for each $i \in \{11, 12, 21\}$,

$$I_i = \{j_1, \dots, j_4 \in \{1, \dots, k-1\} : j_i \bmod 4 \text{ distinct}, \{j_1, \dots, j_4\} \notin \mathcal{D}_i\}.$$

The final predicate to enforce that the coloring represents a valid tiling (of size at most $m \times n$) is defined as follows:

$$\begin{aligned}
\mathrm{TTORI}'_{\mathcal{D}} \ : \ & \exists X^2 \le m, Y^2 \le n, Z \le_i 2, W^k \le_i 2, V^k \le_i 2, \bar{V}^k \le_i 2, F \le_i 2 : \\
& \mathrm{OTORI}'(X^2, Y^2, Z, W^k, V^k, \bar{V}^k, F) \\
& \wedge \bigwedge_{j_1,\dots,j_4 \in I_{11}} \Big(\mathrm{SAT}_{\le 3/4, \varnothing, \{1,\dots,k\} \setminus \{j_1,\dots,j_4\}}((X_1, Y_1, Z), W^k, V^k, \bar{V}^k, F) \\
& \qquad \wedge \mathrm{SAT}_{\le 3/4, \{1,\dots,k\} \setminus \{j_1,\dots,j_4\}, \varnothing}((X_1, Y_1, Z), W^k, V^k, \bar{V}^k, F)\Big) \\
& \wedge \bigwedge_{j_1,\dots,j_4 \in I_{12}} \Big(\mathrm{SAT}_{\le 3/4, \varnothing, \{1,\dots,k\} \setminus \{j_1,\dots,j_4\}}((X_1, Y_2, Z), W^k, V^k, \bar{V}^k, F) \\
& \qquad \wedge \mathrm{SAT}_{\le 3/4, \{1,\dots,k\} \setminus \{j_1,\dots,j_4\}, \varnothing}((X_1, Y_2, Z), W^k, V^k, \bar{V}^k, F)\Big) \\
& \wedge \bigwedge_{j_1,\dots,j_4 \in I_{21}} \Big(\mathrm{SAT}_{\le 3/4, \varnothing, \{1,\dots,k\} \setminus \{j_1,\dots,j_4\}}((X_2, Y_1, Z), W^k, V^k, \bar{V}^k, F) \\
& \qquad \wedge \mathrm{SAT}_{\le 3/4, \{1,\dots,k\} \setminus \{j_1,\dots,j_4\}, \varnothing}((X_2, Y_1, Z), W^k, V^k, \bar{V}^k, F)\Big).
\end{aligned}$$

▶ **Lemma 12.** $\mathrm{TTORI}'_{\mathcal{D}}$ *is satisfied iff $\mathcal{D}$ admits a periodic tiling of size at most $m \times n$.*

Proof. All implicit bounds follow from previously defined predicates used within $\mathrm{TTORI}'_{\mathcal{D}}$: TORI′ for Z, SW for W^k, and FLIP for V^k, $\bar{V}^k$, F. The bounds of m, n for X^2, Y^2 imply that the tori can take any even size up to $2m \times 2n$.

For the "only if" direction, note that the three conjunctions, similarly as in the predicate OTORI, forbid faces of type 11, 12, 21 from being not of the form in $\mathcal{D}_{11}, \mathcal{D}_{12}, \mathcal{D}_{21}$, respectively. Clearly, the set $\mathcal{D}_{11}$ consists of exactly those tiles (represented as type 11 faces) which are in D. Similarly, $\mathcal{D}_{12}, \mathcal{D}_{21}$ consist of all those "glue" type 12 and 21 faces which are allowed by H, V respectively. Therefore, in a distribution satisfying $\mathrm{TTORI}'_{\mathcal{D}}$, the type 11 faces represent tiles and neighboring tiles respect the constraints H and V. Therefore, each torus corresponds to a periodic tiling by $\mathcal{D}$. Since the tori are of size at most $2m \times 2n$, the tiling is of size at most $m \times n$.

For the "if" direction, for any given tiling by $\mathcal{D}$ of size at most $m \times n$, we create a pair of corresponding tori, labeled such that one represents the positive version of this tiling and the other the negative version. The satisfiability arguments show that this can be done for any given tiling. ◀

4.7 Final construction

Once fully expanded, the $\mathrm{TTORI}'_{\mathcal{D}}$ predicate is of the form (omitting the bounds for clarity)

$$\exists B, X^2, Y^2, W^k, V^k, \bar{V}^k, F, \dots : \Big(B \sim \mathrm{Bern}(1/2) \wedge \bigwedge\nolimits_i (I(A_i; B_i | C_i) = 0)\Big),$$

where A_i, B_i, C_i are tuples of the quantified variables. Its negation can be equivalently rewritten:

$$\begin{aligned}
& \neg\exists B, \dots : \big(B \sim \mathrm{Bern}(1/2) \wedge \bigwedge\nolimits_i (I(A_i; B_i | C_i) = 0)\big) \\
\Leftrightarrow\ & \forall B, \dots : \big(B \not\sim \mathrm{Bern}(1/2) \vee \neg \bigwedge\nolimits_i (I(A_i; B_i | C_i) = 0)\big) \\
\Leftrightarrow\ & \forall B, \dots : \big(\big(\bigwedge\nolimits_i (I(A_i; B_i | C_i) = 0)\big) \Rightarrow B \not\sim \mathrm{Bern}(1/2)\big) \\
\Leftrightarrow\ & \forall B, \dots : \big(\big(\mathrm{UNIF}(B) \wedge |B| \le 2 \wedge \bigwedge\nolimits_i (I(A_i; B_i | C_i) = 0)\big) \Rightarrow H(B) = 0\big).
\end{aligned}$$

The last equivalence holds because a variable B with $|B| \le 2$ and $\mathrm{UNIF}(B)$ can either be uniform over one value and have entropy 0 or uniform over two values and have entropy 1. This final form is a valid CI implication with (partial) cardinality bounds. In the bounded case, the established cardinality bounds are preserved.

Enforcing the usage of a designated tile

We extend Li's above construction to enforce that a given tile t be used in the tiling. Recall from Lemma 6 that for any $w^k \in \{0,1\}^k$,

$$H(F|V_{\{i:w_i=1\}}, \bar{V}_{\{i:w_i=0\}}, W^k) = \mathbf{P}(W^k = w^k).$$

Furthermore, variable F is constrained by the predicate $\mathrm{UNIF}_2(F)$. It can be equivalently restated as $\mathrm{UNIF}_=(F,B)$, where $\mathrm{UNIF}_=$ is defined by Li as follows:

$$\mathrm{UNIF}_=(Y,Z) : \exists U^3 \leq_i K_Y : \mathrm{TRIPLE}(Y,U_1,U_2) \wedge \mathrm{TRIPLE}(Z,U_1,U_3).$$

Clearly, $\mathrm{UNIF}_=(F,B)$ holds iff F and B are both uniform with the same cardinality and hence $\mathrm{UNIF}_2(F)$ can be replaced by $\mathrm{UNIF}_=(F,B)$. Therefore, if the (conditional) entropy of F is nonzero, then the entropy of B must also be nonzero and so we must have $B \sim \mathrm{Bern}(\frac{1}{2})$.

Given a designated tile t, let $w^k \in T_k$ be the value of W^k corresponding to label t (without loss of generality, of vertex group 0 and positive sign) and $S = \{i : w_i = 1\}, \bar{S} = \{i : w_i = 0\}$. The modified implication is as follows:

$$\forall B, \ldots : \big(\big(\mathrm{UNIF}(B) \wedge |B| \leq 2 \wedge \bigwedge\nolimits_i (I(A_i;B_i|C_i) = 0)\big) \Rightarrow H(F|V_S, \bar{V}_{\bar{S}}, W^k) = 0\big).$$

The counterexamples of this implication are exactly those labelings which use the tile t. Altogether, this chapter has shown the following theorem:

▶ **Theorem 2.** *For any given tiling system $\mathcal{D}$ along with tile t and natural numbers m, n, there exists a bounded CI implication which holds iff $\mathcal{D}$ does not admit a periodic tiling of size at most $m \times n$ which makes use of tile t. Moreover, the implication can be computed in time polynomial with regard to the size of the tiling system, while the bounds can be computed in time polynomial w.r.t. to the size of m, n.*

Theorem 2 gives exactly a polynomial-time many-one reduction from PERIODIC BOUNDED TILING to the complement of BOUNDED CI IMPLICATION, in particular because m, n and K are encoded in the same manner. In the case of CONSTANT-BOUNDED CI IMPLICATION, the above argument does not work since we have constant bounds larger than 2 as well as bounds whose value depends on the input. However, any variable X with cardinality bound 2^j can be replaced by the tuple $(X_1, \ldots, X_j)$, where X_i has cardinality bound 2 for each $i \in \{1, \ldots, j\}$. These are clearly equivalent, since each X_i can correspond to the i-th bit of X. More generally, a variable with cardinality bound l can be replaced by $(X_1, \ldots, X_{\lceil \log l \rceil})$, with each X_i's cardinality bounded by 2 and the additional requirement $\mathrm{UNIF}'_k((X_1, \ldots, X_{\lceil \log l \rceil}))$. Here $\mathrm{UNIF}'_k(Y)$ is a modification of the UNIF_k predicate such that it enforces Y being uniform with $|Y| \leq k$. The construction of both of these follows closely that of Li and is given in detail in the full version. Note that the resulting predicate can be large, but this is only important when the bound is not constant – in our case these are only the two bounds $X^2 \leq m, Y^2 \leq n$. To avoid this issue, we reduce from POWER-OF-TWO PERIODIC BOUNDED TILING – then X_i, Y_i (for $i \in \{1,2\}$) have bounds $2^m, 2^n$ respectively, while the remaining variables have constant bounds. Replacing X_1, X_2, Y_1, Y_2 by tuples of binary variables as shown above, we obtain a CI implication of size $N \cdot O(m+n)$, where N is the size of the original implication. The number of random variables grows similarly. The newly created bounds are all constant, and the reduction takes time polynomial with regard to the input size. The values of the remaining constant bounds are known and therefore each such variable can be converted in constant time. Together, these two results yield Theorem 1.

References

1 John F. Canny. Some algebraic and geometric computations in PSPACE. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 460–467. ACM, 1988. doi:10.1145/62212.62257.

2 Dan Geiger and Judea Pearl. Logical and algorithmic properties of conditional independence and graphical models. *The Annals of Statistics*, 21(4):2001–2021, 1993.

3 Daniel Gottesman and Sandy Irani. The quantum and classical complexity of translationally invariant tiling and hamiltonian problems. *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, May 2009. doi:10.1109/FOCS.2009.22.

4 Miika Hannula, Åsa Hirvonen, Juha Kontinen, Vadim Kulikov, and Jonni Virtema. Facets of distribution identities in probabilistic team semantics. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 304–320. Springer, 2019. doi:10.1007/978-3-030-19570-0_20.

5 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Decision problems in information theory. 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 106:1–106:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.106.

6 Lukas Kühne and Geva Yashfe. On entropic and almost multilinear representability of matroids. *CoRR*, abs/2206.03465, 2022. arXiv:2206.03465.

7 Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation, 2nd Edition*. Prentice Hall, 1998.

8 Cheuk Ting Li. The undecidability of conditional affine information inequalities and conditional independence implication with a binary constraint. *IEEE Transactions on Information Theory*, 68(12):7685–7701, 2022. doi:10.1109/TIT.2022.3190800.

9 Cheuk Ting Li. Undecidability of network coding, conditional information inequalities, and conditional independence implication. *IEEE Transactions on Information Theory*, 2023. doi:10.1109/TIT.2023.3247570.

10 Michał Makowski. On the complexity of the conditional independence implication problem with bounded cardinalities, 2024. Full version of this paper. arXiv:2408.02550.

11 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 459–468, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335358.

12 Peter van Emde Boas. The convenience of tilings. In A. Sorbi, editor, *Complexity, Logic, and recursion Theory*, volume 187 of *Lecture Notes in Pure and Applied Logic*, pages 331–363. Marcel Dekker, Inc., 1997.

13 Z. Zhang and R.W. Yeung. A non-Shannon-type conditional inequality of information quantities. *IEEE Transactions on Information Theory*, 43(6):1982–1986, 1997. doi:10.1109/18.641561.

Synthesis of Robust Optimal Real-Time Systems

Benjamin Monmege
Aix Marseille Univ, CNRS, LIS, Marseille, France

Julie Parreaux
University of Warsaw, Poland

Pierre-Alain Reynier
Aix Marseille Univ, CNRS, LIS, Marseille, France

Abstract

Weighted Timed Games (WTGs for short) are widely used to describe real-time controller synthesis problems, but they rely on an unrealistic perfect measure of time elapse. In order to produce strategies tolerant to timing imprecisions, we consider a notion of robustness, expressed as a parametric semantics, first introduced for timed automata. WTGs are two-player zero-sum games played in a weighted timed automaton in which one of the players, that we call Min, wants to reach a target location while minimising the cumulated weight. The opponent player, in addition to controlling some of the locations, can perturb delays chosen by Min. The robust value problem asks, given some threshold, whether there exists a positive perturbation and a strategy for Min ensuring to reach the target, with an accumulated weight below the threshold, whatever the opponent does.

We provide in this article the first decidability result for this robust value problem. More precisely, we show that we can compute the robust value function, in a parametric way, for the class of divergent WTGs (this class has been introduced previously to obtain decidability of the (classical) value problem in WTGs without bounding the number of clocks). To this end, we show that the robust value is the fixpoint of some operators, as is classically done for value iteration algorithms. We then combine in a very careful way two representations: piecewise affine functions introduced in [1] to analyse WTGs, and shrunk Difference Bound Matrices (shrunk DBMs for short) considered in [29] to analyse robustness in timed automata. The crux of our result consists in showing that using this representation, the operator of value iteration can be computed for infinitesimally small perturbations. Last, we also study qualitative decision problems and close an open problem on robust reachability, showing it is EXPTIME-complete for general WTGs.

2012 ACM Subject Classification Software and its engineering → Formal software verification; Theory of computation → Algorithmic game theory

Keywords and phrases Weighted timed games, Algorithmic game theory, Robustness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.74

Related Version *Full Version*: `https://arxiv.org/abs/2403.06921` [25]

Funding This work has been partly funded by the QuaSy project (ANR-23-CE48-0008) and the NCN grant 2019/35/B/ST6/02322.

1 Introduction

The design and synthesis of real-time systems have long been paramount challenges, given the critical need for dependable and efficient systems in a variety of applications. In particular, the pursuit of robustness and reliability in these systems has led researchers to explore innovative methods and formalisms to address the complexities inherent in real-time environments. In this work, we focus on game-based models, and more precisely on the game extension of timed automata [2], a.k.a. timed games, which provide an elegant framework for capturing the interplay between system components, environment dynamics, and strategic decision-making. More precisely, in this model, locations of a timed automaton are split amongst the two players, which play in turn in the infinite-state space of the automaton.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 74; pp. 74:1–74:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Regarding robustness, prior studies have primarily focused on areas such as fault tolerance, adaptive control, and formal methods. In this work, we follow a series of works based on game theory. The objective is to fill the gap between mathematical models such as timed automata, often used for model-checking purposes, and implementation constraints, in which clocks only have finite precision, and actions are not instantaneous. To that end, a parametric semantics has been considered in [27], which consists in allowing the delays to be perturbed by some limited amount. The uncertainty of the system, i.e. the perturbation of the delays, is modelled by an adversarial environment. Two kinds of problems can then be considered: first, the analysis may be done for a fixed perturbation bound (we call it a *fixed-perturbation robustness problem*); second, in order to abstract the precise settings of the implementation, and as the exact value of the perturbation bound may be unknown, one can try to determine whether there exists a perturbation bound under which the system is reliable (we call it an *existential robustness problem*). By monotonicity of the semantics w.r.t. the perturbation bound, if one manages to prove the reliability against some perturbation bound, then it still holds for smaller perturbations. Initially introduced for model-checking purposes [20, 8], this approach has been lifted to automatic synthesis, yielding the so-called *conservative semantics*, studied for instance in [30, 26, 19]. In these works, a player named Controller aims at satisfying a liveness objective while its opponent may perturb the delays.

In the present work, we aim to go beyond qualitative objectives, and tackle quantitative aspects. In real-time systems and critical applications, quantitative aspects such as resource utilization and performance metrics hold important significance. This has led to the model of *weighted timed games* (WTG for short), which has been widely studied during the last two decades. When considering a reachability objective, Controller (a.k.a. player Min) aims at reaching a set of target locations while minimizing the accumulated weight. One is then interested in the *value problem*, which consists in deciding, given some threshold, whether a strategy for Min exists to reach some target location while keeping the accumulated cost below this threshold. While this problem is undecidable in general [11, 5, 14], several subclasses have been identified that allow one to regain decidability. Amongst recent works, we can cite the class of divergent WTGs [18] which generalize to arbitrary costs the class of strictly non-Zeno costs introduced in [6], or the class of one-clock WTGs [24].

The core objective of this research is to explore the synthesis of real-time systems that not only meet timing constraints but also optimize performance with respect to specified weight objectives and are robust against timing imprecisions. To that end, we aim to study the setting of timed games extended with both robustness issues and quantitative aspects. We focus on the conservative semantics and on reachability objectives. In this setting, under a fixed perturbation, the player Min aims at reaching a set of target locations while minimizing the accumulated weight, and resisting delay perturbations. This leads to a notion of *robust value under a fixed perturbation*: this is simply the best value Min can achieve. The associated fixed perturbation robust value problem aims at comparing this value with a given threshold. When turning to the existential robustness decision problem, one considers the notion of *robust value*, defined as the limit of robust values for arbitrarily small perturbation values. We prove that this limit exists, and study the associated decision problem, which we simply call *robust value problem*, and which can be defined as follows: given a threshold, determine whether there exists a positive perturbation, and a winning strategy for Min ensuring that the accumulated weight until the target is below the threshold.

This problem is highly challenging as it combines difficulties coming from the introduction of weights, with those due to the analysis of an existential problem for the parametric semantics of robustness. Unsurprisingly, it has been shown to be undecidable [28]. To highlight the challenges we face, already in the qualitative setting (w/o weights), the setting of two-player has not been addressed yet for the conservative semantics, hence the existential robust reachability problem was left open in [28]. Indeed, in [19, 30], only the one-player case is handled (a partial extension is considered in [26]) and the existential robustness reachability problem for the two-player setting has only been solved for the excessive semantics (an alternative to the conservative semantics) in [10]. Regarding the quantitative setting, very few works have addressed robustness issues. The fixed-perturbation robust value problem is shown to be decidable for one-clock weighted timed games in [21], with non-negative weights only, and for the excessive semantics. In [9], the authors consider the one-player case and prove that the robust value problem is PSPACE-complete.

Our contributions are as follows: first, regarding the qualitative setting, we close the case of existential robust reachability in two-player timed games for the conservative semantics, and show that this problem is EXPTIME-complete. To do so, we introduce a construction which allows us to reduce the problem to the excessive semantics solved in [10]. As a corollary, we deduce an upper bound on the length of paths to the target.

Then, we turn to the quantitative setting and show that for the class of divergent WTGs (one of the largest classes of WTGs for which the decidability of the value problem is known), the robust value problem is decidable. We proceed as follows:

1. We characterize the robust value for a fixed perturbation as the fixpoint of some operator.
2. We show that for acyclic WTGs, this fixpoint can be obtained as a finite iteration of this operator, which we decompose using four simpler operators.
3. We introduce a symbolic parametric approach for the computation of this operator, for arbitrarily small values of the perturbation. This requires carefully combining the representation of value functions using piecewise affine functions introduced in [1] with the notion of shrunk DBMs, used in [29] to analyse robustness issues in timed automata. This yields the decidability of the robust value problem for the class of acyclic WTGs.
4. By combining this with the upper bound deduced from the qualitative analysis, we show the decidability of the robust value problem for the whole class of divergent WTGs.

In Section 2, we introduce WTGs, under the prism of robustness. We describe in Section 3 the robustness problems we consider, present our contributions for qualitative ones, and state that we can solve the quantitative one for acyclic WTGs. Sections 4 and 5 detail how to prove this result, following steps 1.-3. described above. Last, Section 6 extends this positive result to the class of divergent WTGs. Omitted proofs can be found in a long version of this article [25].

2 Robustness in weighted timed games

We let $\mathcal{X}$ be a finite set of variables called clocks. A valuation is a mapping $\nu\colon \mathcal{X} \to \mathbb{R}_{\geq 0}$. For a valuation ν, a delay $t \in \mathbb{R}_{\geq 0}$ and a subset $Y \subseteq \mathcal{X}$ of clocks, we define the valuation $\nu + t$ as $(\nu+t)(x) = \nu(x) + t$, for all $x \in \mathcal{X}$, and the valuation $\nu[Y := 0]$ as $(\nu[Y := 0])(x) = 0$ if $x \in Y$, and $(\nu[Y := 0])(x) = \nu(x)$ otherwise. A (non-diagonal) guard on clocks of $\mathcal{X}$ is a conjunction of atomic constraints of the form $x \bowtie c$, where $\bowtie \in \{\leq, <, =, >, \geq\}$ and $c \in \mathbb{N}$. A valuation $\nu\colon \mathcal{X} \to \mathbb{R}_{\geq 0}$ satisfies an atomic constraint $x \bowtie c$ if $\nu(x) \bowtie c$. The satisfaction relation is extended to all guards g naturally, and denoted by $\nu \models g$. We let $\mathsf{Guards}(\mathcal{X})$ denote the set of guards over $\mathcal{X}$.

▶ **Definition 1.** *A* weighted timed game *(WTG) is a tuple* $\mathcal{G} = \langle L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_T, \mathcal{X}, \Delta, \mathsf{wt}\rangle$ *where* L_{Min}, L_{Max}, L_T *are finite disjoint subsets of* Min *locations,* Max *locations, and target locations, respectively (we let* $L = L_{\mathsf{Min}} \uplus L_{\mathsf{Max}} \uplus L_T$*),* $\mathcal{X}$ *is a finite set of clocks,* $\Delta \subseteq L \times \mathsf{Guards}(\mathcal{X}) \times 2^{\mathcal{X}} \times L$ *is a finite set of transitions, and* $\mathsf{wt}\colon \Delta \uplus L \to \mathbb{Z}$ *is the weight function.*

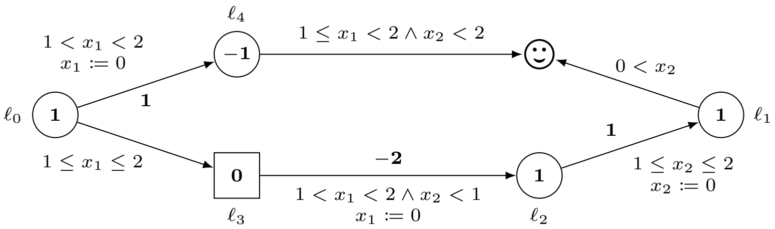


Figure 1 An acyclic WTG with two clocks.

The usual semantics, called *exact semantics*, of a WTG $\mathcal{G}$ is defined in terms of a game played on an infinite transition system whose vertices are configurations of the WTG denoted by $\mathsf{Conf} = \mathsf{Conf}_{\mathsf{Min}} \uplus \mathsf{Conf}_{\mathsf{Max}} \uplus \mathsf{Conf}_T$. A configuration is a pair (ℓ, ν) with a location and a valuation of the clocks. A configuration is final (resp. belongs to Min, or Max), and belongs to Conf_T (resp. to $\mathsf{Conf}_{\mathsf{Min}}$, or $\mathsf{Conf}_{\mathsf{Max}}$) if its location is a target location of L_T (resp. of L_{Min}, or L_{Max}). The alphabet of the transition system is given by $\Delta \times \mathbb{R}_{\geq 0}$: a pair (δ, t) encodes the delay t that a player wants to spend in the current location, before firing transition δ. An example of WTG is depicted in Figure 1.

In this article, we consider an alternative semantics to model the robustness, traditionnally called the *conservative semantics*. It is defined in a WTG $\mathcal{G}$ according to a fixed parameter $p > 0$. This semantics allows Max to slightly perturb the delays chosen by Min with an amplitude bounded by p. From the modelling perspective, the perturbations model the small errors of physical systems on the real value of clocks. Conservative means that the delays proposed by Min must remain feasible after applying all possible perturbations. In particular, the conservative semantics does not add new edges with respect to the exact one.

▶ **Definition 2.** *Let $\mathcal{G} = \langle L_{\mathsf{Min}}, L_{\mathsf{Max}}, L_T, \mathcal{X}, \Delta, \mathsf{wt}\rangle$ be a WTG. For $p \geq 0$, we let $[\![\mathcal{G}]\!]^p = \langle S, E, \mathsf{wt}\rangle$ with $S = S_{\mathsf{Min}} \uplus S_{\mathsf{Max}} \uplus S_T$ the set of* states *with $S_{\mathsf{Min}} = \mathsf{Conf}_{\mathsf{Min}}$, $S_T = \mathsf{Conf}_T$ and $S_{\mathsf{Max}} = \mathsf{Conf}_{\mathsf{Max}} \cup (\mathsf{Conf}_{\mathsf{Min}} \times \mathbb{R}_{\geq 0} \times \Delta)$; $E = E_{\mathsf{Min}} \uplus E_{\mathsf{Max}} \uplus E_{\mathit{rob}}$ the set of* edges *with*

$$E_{\mathsf{Max}} = \big\{\big((\ell,\nu) \xrightarrow{\delta,t} (\ell',\nu')\big) \mid \ell \in L_{\mathsf{Max}}, \nu + t \models g \text{ and } \nu' = (\nu + t)[Y := 0]\big\}$$
$$E_{\mathsf{Min}} = \big\{\big((\ell,\nu) \xrightarrow{\delta,t} (\ell,\nu,\delta,t)\big) \mid \ell \in L_{\mathsf{Min}}, \nu + t \models g \text{ and } \nu + t + 2p \models g\big\}$$
$$E_{\mathit{rob}} = \big\{\big((\ell,\nu,\delta,t) \xrightarrow{\delta,\varepsilon} (\ell',\nu')\big) \mid \varepsilon \in [0,2p] \text{ and } \nu' = (\nu + t + \varepsilon)[Y := 0]\big\}$$

where $\delta = (\ell, g, Y, \ell') \in \Delta$; and $\mathsf{wt}\colon S \cup E \to \mathbb{Z}$ the weight function such that for all states $s \in S$ with $s = (\ell, \nu)$ or $s = (\ell, \nu, \delta, t)$, $\mathsf{wt}(s) = \mathsf{wt}(\ell)$, and all edges $e \in E$, $\mathsf{wt}(e) = \mathsf{wt}(\delta)$ if $e = (s \xrightarrow{\delta,t} s')$ with $s \in \mathsf{Conf}$, or $\mathsf{wt}(e) = 0$ otherwise.

When $p = 0$, the infinite transition system $[\![\mathcal{G}]\!]^0$ describes the exact semantics of the game, the usual semantics where each step of the player Min is cut into the true step, followed by a useless edge $(\ell, \nu, \delta, t) \xrightarrow{\delta,0} (\ell', \nu')$ where Max has no choice. When $p > 0$, the infinite transition system $[\![\mathcal{G}]\!]^p$ describes the conservative semantics of the game: states $(\ell, \nu, \delta, t) \in \mathsf{Conf}_{\mathsf{Min}} \times \mathbb{R}_{\geq 0} \times \Delta$ where Max must choose the perturbation in the interval $[0, 2p]$ are called *perturbed states*.

Let s be a state of $[\![\mathcal{G}]\!]^p$, we denote by $E(s)$ the set of possible outgoing edges of $[\![\mathcal{G}]\!]^p$ from s. We extend this notation to locations to denote the set of outgoing transitions in $\mathcal{G}$. A state (resp. location) s is a *deadlock* when $E(s) = \emptyset$. We note that the conservative semantics may introduce deadlock in configurations of Min (even if an outgoing edge exists in the exact semantics). Thus, unlike in the literature [1, 18], we allow state *and* location deadlocks.

A *finite play* of $\mathcal{G}$ w.r.t. the conservative semantics with parameter p is a sequence of edges in the transition system $[\![\mathcal{G}]\!]^p$ starting in a configuration of $\mathcal{G}$. We denote by $|\rho|$ the number of edges of ρ, and by $\mathsf{last}(\rho)$ its last state. The concatenation of two finite plays ρ_1 and ρ_2, such that ρ_1 ends in the same state as ρ_2 starts, is denoted by $\rho_1\rho_2$. Moreover, for modelling reasons, we only consider finite plays (starting and) ending in a configuration of $\mathcal{G}$. Since a finite play is always defined regarding a parameter p for the conservative semantics, we denote by FPlays^p this set of finite plays. Moreover, we denote by $\mathsf{FPlays}^p_{\mathsf{Min}}$ (resp. $\mathsf{FPlays}^p_{\mathsf{Max}}$) the subset of these finite plays ending in a state of Min (resp. Max). A *maximal play* is then a maximal sequence of consecutive edges: it is either a finite play reaching a deadlock (not necessary in L_T), or an infinite sequence such that all its prefixes are finite plays.

The objective of Min is to reach a target configuration, while minimising the cumulated weight up to the target. Hence, we associate to every finite play $\rho = s_0 \xrightarrow{\delta_0,t_0} s_1 \xrightarrow{\delta_1,t_1} \cdots s_k$ (some edges are in E_{rob}, others are not) its cumulated weight, taking into account both discrete and continuous costs: $\mathsf{wt}_\Sigma(\rho) = \sum_{i=0}^{k-1}[t_i \times \mathsf{wt}(s_i) + \mathsf{wt}(\delta_i)]$. Then, the weight of a maximal play ρ, denoted by $\mathsf{wt}(\rho)$, is defined by $+\infty$ if ρ does not reach L_T (because it is infinite or reaches another deadlock), and $\mathsf{wt}_\Sigma(\rho)$ if it ends in (ℓ,ν) with $\ell \in L_T$.

A strategy for Min (resp. Max) is a mapping from finite plays ending in a state of Min (resp. Max) to a decision in (δ,t) labelling an edge of $[\![\mathcal{G}]\!]^p$ from the last state of the play. Since plays could reach a deadlock state of Min, we consider strategies of Min to be partial mappings. For instance, in the WTG depicted in Figure 1 and a perturbation p, a strategy for Min in all plays ending in (ℓ_2,ν) can be defined only when $\nu(x_2) \leq 2-2p$ since, otherwise, there are no outgoing edges in $[\![\mathcal{G}]\!]^p$ from this state. Symmetrically, we ask for Max to always propose a move if we are not in a deadlock state. More formally, a strategy for Min, denoted χ, is a (possibly partial) mapping $\chi\colon \mathsf{FPlays}^p_{\mathsf{Min}} \to E$ such that $\chi(\rho) \in E(\mathsf{last}(\rho))$. A strategy for Max, denoted ζ, is a (possibly partial) mapping $\zeta\colon \mathsf{FPlays}^p_{\mathsf{Max}} \to E$ such that for all ρ, if $E(\mathsf{last}(\rho)) \neq \emptyset$, then $\chi(\rho)$ is defined, and in this case belongs to $E(\mathsf{last}(\rho))$. The set of strategies of Min (resp. Max) with the perturbation p is denoted by $\mathsf{Strat}^p_{\mathsf{Min}}$ (resp. $\mathsf{Strat}^p_{\mathsf{Max}}$).

A play or finite play $\rho = s_0 \xrightarrow{\delta_0,t_0} s_1 \xrightarrow{\delta_1,t_1} \cdots$ *conforms* to a strategy χ of Min (resp. Max) if for all k such that s_k belongs to Min (resp. Max), we have that $(\delta_k,t_k) = \chi(s_0 \xrightarrow{\delta_0,t_0} \cdots s_k)$. For all strategies χ and ζ of players Min and Max, respectively, and for all configurations (ℓ_0,ν_0), we let $\mathsf{Play}((\ell_0,\nu_0),\chi,\zeta)$ be the outcome of χ and ζ, defined as the unique maximal play conforming to χ and ζ and starting in (ℓ_0,ν_0).

The semantics $[\![\mathcal{G}]\!]^p$ is monotonic with respect to the perturbation p in the sense that Min has more strategies when p decreases, while Max can obtain, against a fixed strategy of Min, a smaller weight when p decreases. Formally, we have:

▶ **Lemma 3.** *Let $\mathcal{G}$ be a WTG, and $p > p' \geq 0$ be two perturbations. Then*

1. $\mathsf{Strat}^p_{\mathsf{Min}} \subseteq \mathsf{Strat}^{p'}_{\mathsf{Min}}$;
2. *for all* $\chi \in \mathsf{Strat}^p_{\mathsf{Min}}$, $\sup_{\zeta\in\mathsf{Strat}^p_{\mathsf{Max}}} \mathsf{wt}(\mathsf{Play}((\ell,\nu),\chi,\zeta)) \geq \sup_{\zeta\in\mathsf{Strat}^{p'}_{\mathsf{Max}}} \mathsf{wt}(\mathsf{Play}((\ell,\nu),\chi,\zeta))$.

3 Deciding the robustness in weighted timed games

We aim to study what Min can guarantee, qualitatively and then quantitatively, in the conservative semantics of weighted timed games whatever Max does.

Qualitative robustness problems. Formally, given a WTG $\mathcal{G}$ and a perturbation p, we say a strategy χ of Min is *winning* in $[\![\mathcal{G}]\!]^p$ from configuration (ℓ,ν) if for all strategies ζ of Max, $\mathsf{Play}((\ell,\nu),\chi,\zeta)$ is a finite play ending in a location of L_T.

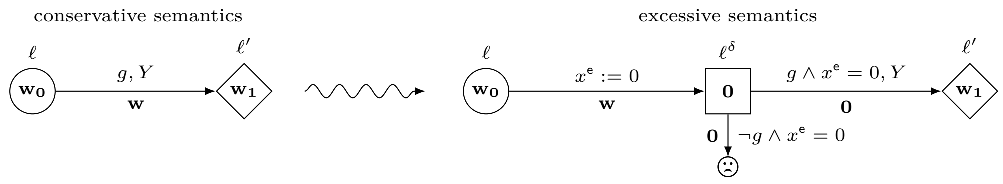


Figure 2 Gadget used to encode the conservative semantics into the excessive one. Each transition $\delta = (\ell, g, Y, \ell')$ with $\ell \in L_{\mathsf{Min}}$ is replaced by the gadget. Symbols w, w_0, w_1 denote weights from $\mathcal{G}$. The new location ℓ^δ of Max uses a fresh clock x^{e} to test the guard after the perturbation (as in the conservative semantics). The new location ☹ is a deadlock (thus winning for Max).

There are two possible questions, whether the perturbation p is fixed, or if we should consider it to be infinitesimally small:

- *fixed-perturbation robust reachability problem*: given a WTG $\mathcal{G}$, a configuration (ℓ, ν) and a perturbation $p > 0$, decide whether Min has a winning strategy χ from (ℓ, ν) in $[\![\mathcal{G}]\!]^p$;
- *existential robust reachability problem*: given a WTG $\mathcal{G}$ and a configuration (ℓ, ν), decide whether there exists $p > 0$ such that Min has a winning strategy χ from (ℓ, ν) in $[\![\mathcal{G}]\!]^p$. Notice that by Lemma 3, if Min has a winning strategy χ from (ℓ, ν) in $[\![\mathcal{G}]\!]^p$, then he has one in $[\![\mathcal{G}]\!]^{p'}$ for all $p' \leq p$.

When the perturbation p is fixed, we can encode in a WTG the conservative semantics described in $[\![\mathcal{G}]\!]^p$, by adding new locations for Max to choose a perturbation, and by modifying the guards that will now use the perturbation p. Solving the fixed-perturbation robust reachability problem then amounts to solving a reachability problem in the modified WTG[1] which can be performed in EXPTIME [3] (here weights are useless). Since the reachability problem in timed games is already EXPTIME-complete [22], we obtain:

▶ **Proposition 4.** *The fixed-perturbation robust reachability problem is* EXPTIME*-complete.*

We now turn our attention to the existential robust reachability problem. This problem was left open for the conservative semantics (see [28], Table 1.2 page 17), while it has been solved in [10] for an alternative semantics of robustness, known as the *excessive semantics*. Intuitively, while the conservative semantics requires that the delay, after perturbation, satisfies the guard, the excessive semantics only requires that the delay, *without perturbation*, satisfies the guard. We present a reduction from the conservative semantics to the excessive one, allowing us to solve the existential robust reachability problem for the conservative semantics. Intuitively, the construction (depicted on Figure 2) adds a new location (for Max) for each transitions of Min to test the delay chosen by Min after the perturbation:

▶ **Proposition 5.** *The existential robust reachability problem is* EXPTIME*-complete.*

Quantitative fixed-perturbation robustness problem. We are also interested in the minimal weight that Min can guarantee while reaching the target whatever Max does: to do that we define *robust values*. First, we define the *fixed-perturbation robust value*: for all configurations (ℓ, ν) of $\mathcal{G}$ (and not for all states of the semantics), we let $\overline{\mathsf{rVal}}^p(\ell, \nu) = \inf_{\chi \in \mathsf{Strat}^p_{\mathsf{Min}}} \sup_{\zeta \in \mathsf{Strat}^p_{\mathsf{Max}}} \mathsf{wt}(\mathsf{Play}((\ell, \nu), \chi, \zeta))$.

[1] By transforming the WTG, its guards use rational instead of natural numbers (due to p). To fit the classical definition of WTG, we can apply a scaling factor (i.e. $1/p$) to all constants appearing in this WTG. We note that this operation preserves the set of winning strategies for the reachability objective (here weights are irrelevant) by applying the scaling operations on strategies too.

Since a fixed-perturbation conservative semantics defines a *quantitative reachability game*[2], where configurations of Max also contain the robust states, we obtain that the fixed-perturbation robust value is determined, by applying [13, Theorem 2.2], i.e. that $\overline{\mathsf{rVal}}^p(\ell,\nu) = \sup_{\zeta\in\mathsf{Strat}^p_{\mathsf{Max}}} \inf_{\chi\in\mathsf{Strat}^p_{\mathsf{Min}}} \mathsf{wt}(\mathsf{Play}((\ell,\nu),\chi,\zeta))$. We therefore denote rVal^p this value.

▶ Remark 6. In [9, 10, 21], the set of possible perturbations for Max is $[-p,p]$. For technical reasons, we use a (equivalent) perturbation with a shift of the delay proposed by Min by p.

When $p=0$, rVal^0 defines the *(exact) value* that is used to study the value problem in WTGs. By Lemma 3, we can deduce that the fixed-perturbation robust value is monotonic with respect to the perturbation p and is always an upper-bound for the (exact) value.

▶ **Lemma 7.** *Let $\mathcal{G}$ be a WTG, and $p > p' \geq 0$ be two perturbations. Then, for all configurations (ℓ,ν), $\mathsf{rVal}^p(\ell,\nu) \geq \mathsf{rVal}^{p'}(\ell,\nu)$.*

As in the qualitative case, when the perturbation p is fixed, we can encode in polynomial time in a WTG the conservative semantics described in $[\![\mathcal{G}]\!]^p$. Unfortunately, the value of WTGs is not always computable since the associated decision problems (in particular the *value problem* that requires to decide if the value of a given configuration is below a given threshold) are undecidable [11, 4, 14]. However, in subclasses of WTGs where the value function can be computed, like acyclic WTGs (where every path in the graph of the WTGs is acyclic, decidable in 2-EXPTIME [15]) or divergent WTGs (that we recall in Section 6, in 3-EXPTIME [6, 16]), the fixed-perturbation robust value is also computable (if the modified game falls in the subclass). In particular, we obtain:

▶ **Proposition 8.** *We can compute the fixed-perturbation robust value of a WTG that is acyclic (in 2-*EXPTIME*) or divergent (in 3-*EXPTIME*), for all possible initial configurations.*

On top of computing the robust values, the previous works also allow one to synthesize almost-optimal (*i.e.* arbitrarily close from the value) strategies for both players.

Quantitative robustness problem. Now, we consider the existential version of this problem by considering an infinitesimal perturbation. We thus want to know what Min can guarantee as a value if Max plays infinitesimally small perturbations. To define properly the problem, we introduce a new value: given a WTG $\mathcal{G}$, the *robust value* is defined, for all configurations (ℓ,ν) of $\mathcal{G}$, by $\overline{\mathsf{rVal}}(\ell,\nu) = \lim_{p\to 0, p>0} \mathsf{rVal}^p(\ell,\nu)$. This value is defined as a limit of functions (the fixed-perturbation robust values), which can be proved to always exist as the limit of a non-increasing sequence of functions (see Lemma 7). The decision problem associated to this robust value is: given a WTG $\mathcal{G}$, an initial configuration (ℓ,ν), and a threshold $\lambda \in \mathbb{Q}\cup\{-\infty,+\infty\}$, decide if $\overline{\mathsf{rVal}}(\ell,\nu) \leq \lambda$. We call it the *robust value problem.*

Unsurprisingly, this problem is undecidable [9, Theorem 4]. We will thus consider some restrictions over WTGs. In particular, we consider classes of WTGs where the (non robust) value problem is known as decidable for the exact semantics: acyclic WTGs [1], and divergent WTGs [6, 16]. Our first main contribution concerns the acyclic case:

▶ **Theorem 9.** *The robust value problem is decidable over the subclass of acyclic WTGs.*

The next two sections sketch the proof of this theorem via an adaptation of the value iteration algorithm [1] used to compute the value function in (non-robust) acyclic WTGs: it consists in computing iteratively the best thing that both players can guarantee in a bounded

[2] A quantitative reachability game introduced in [12] is an abstract model to formally define the semantics of quantitative (infinite) games.

number of steps, that increases step by step. It is best described by a mapping $\mathcal{F}$ that explains how a value function gets modified by allowing one more step for the players to play. The adaptation we propose consists in taking the robustness into account by using shrunk DBM techniques introduced in [29]: instead of inequalities on the difference of two clock values of the form $x - y \leq c$ involving rational constants c, the constants c will now be of the form $a - bp$, with a a rational and b a positive integer, p being an infinitesimal perturbation. This will allow us to compute a description of the fixed-perturbation values for all initial configurations, for all perturbations p smaller than an upperbound that we will compute. The robust value will then be obtained by taking the limit of this parametric representation of the fixed-perturbation values when p tends to 0. Our algorithm will also compute an upperbound on the biggest allowed perturbation p. As in previous works, once the value is computed, we can also synthesize almost-optimal strategies.

Section 4 will describe the mapping $\mathcal{F}_p$, with a known perturbation p: the iteration of this operator will be shown to converge towards the fixed-perturbation value rVal^p. The robust value functions will be shown to always be piecewise affine functions with polytope pieces. Section 5 describes the parametric representation of these functions, where the perturbation is no longer fixed but is a formal parameter $\mathfrak{p}$. We then explain how the mapping $\mathcal{F}_p$ can be computed for all small enough values of p at once, allowing us to conclude.

4 Operator $\mathcal{F}_p$ to compute the fixed-perturbation value

The first step of the proof is the definition of the new operator adapted from the operator $\mathcal{F}$ of [1]. We thus fix a perturbation $p > 0$, and we define an operator $\mathcal{F}_p$ taking as input a mapping $X \colon L \times \mathbb{R}_{\geq 0}^{\mathcal{X}} \to \mathbb{R}_\infty$, computing a mapping $\mathcal{F}_p(X) \colon L \times \mathbb{R}_{\geq 0}^{\mathcal{X}} \to \mathbb{R}_\infty$ defined for all configurations (ℓ, ν) by $\mathcal{F}_p(X)(\ell, \nu)$ equal to

$$\begin{cases} 0 & \text{if } \ell \in L_T \\ +\infty & \text{if } \ell \in L_{\mathsf{Max}} \text{ and } E(\ell,\nu) = \emptyset \quad (\text{if } \mathsf{Max} \text{ reaches a deadlock, he wins}) \\ \sup_{(\ell,\nu) \xrightarrow{\delta,t} (\ell',\nu') \in [\![\mathcal{G}]\!]^p} \big[t\,\mathsf{wt}(\ell) + \mathsf{wt}(\delta) + X(\ell',\nu')\big] & \text{if } \ell \in L_{\mathsf{Max}} \text{ and } E(\ell,\nu) \neq \emptyset \\ \inf_{(\ell,\nu) \xrightarrow{\delta,t} (\ell,\nu,\delta,t) \in [\![\mathcal{G}]\!]^p} \sup_{(\ell,\nu,\delta,t) \xrightarrow{\delta,\varepsilon} (\ell,\nu') \in [\![\mathcal{G}]\!]^p} \big[(t{+}\varepsilon)\,\mathsf{wt}(\ell){+}\mathsf{wt}(\delta){+}X(\ell',\nu')\big] & \text{if } \ell \in L_{\mathsf{Min}} \end{cases}$$

In the following, we let $\mathbb{V}^0$ be the mapping $L \times \mathbb{R}_{\geq 0}^{\mathcal{X}} \to \mathbb{R}_\infty$ defined by $\mathbb{V}^0(\ell,\nu) = 0$ if $\ell \in L_T$ and $\mathbb{V}^0(\ell,\nu) = +\infty$ otherwise. By adapting the proof of the non-robust setting, we show:

▶ **Lemma 10.** *Let $\mathcal{G}$ be an acyclic WTG, $p > 0$, D is the depth of $\mathcal{G}$, i.e. the length of a longest path in $\mathcal{G}$. Then,* rVal^p *is a fixpoint of $\mathcal{F}_p$, and* $\mathsf{rVal}^p = \mathcal{F}_p^D(\mathbb{V}^0)$.

We can thus compute the fixed-perturbation robust value of an acyclic WTG by repeatedly computing $\mathcal{F}_p$. We will see in the next section that this computation can be made for all small enough p by using a parametric representation of the mappings. It will be easier to split the computation of $\mathcal{F}_p$ in several steps (as done in the non robust case [1, 18]). Each of the four operators takes as input a mapping $V \colon \mathbb{R}_{\geq 0}^{\mathcal{X}} \to \mathbb{R}_\infty$ (where the location ℓ has been fixed, with respect to mappings $L \times \mathbb{R}_{\geq 0}^{\mathcal{X}} \to \mathbb{R}_\infty$), and computes a mapping of the same type.

- The operator $\mathsf{Unreset}_Y$, with $Y \subseteq \mathcal{X}$ a subset of clocks, is such that for all $\nu \in \mathbb{R}_{\geq 0}^{\mathcal{X}}$, $\mathsf{Unreset}_Y(V)(\nu) = V(\nu[Y := 0])$.
- The operator Guard_δ, with $\delta = (\ell, g, Y, \ell')$ a transition of Δ, is such that for all $\nu \in \mathbb{R}_{\geq 0}^{\mathcal{X}}$, if $\nu \models g$, then $\mathsf{Guard}_\delta(V)(\nu) = V(\nu)$; otherwise, $\mathsf{Guard}_\delta(V)(\nu)$ is equal to $-\infty$ if $\ell \in L_{\mathsf{Max}}$, and $+\infty$ if $\ell \in L_{\mathsf{Min}}$.

- The operator Pre_ℓ, with $\ell \in L$, is such that for all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$, $\mathsf{Pre}_\ell(V)(\nu)$ is equal to $\sup_{t\geq 0}[t\,\mathsf{wt}(\ell) + V(\nu+t)]$ if $\ell \in L_{\mathsf{Max}}$, and $\inf_{t\geq 0}[t\,\mathsf{wt}(\ell) + V(\nu+t)]$ if $\ell \in L_{\mathsf{Min}}$.
- The operator $\mathsf{Perturb}^p_\ell$, with perturbation $p > 0$ and $\ell \in L_{\mathsf{Min}}$, is such that for all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$, $\mathsf{Perturb}^p_\ell(V)(\nu) = \sup_{\varepsilon\in[0,2p]}[\varepsilon\,\mathsf{wt}(\ell) + V(\nu+\varepsilon)]$.

Though the situation is not symmetrical for Min and Max in $\mathcal{F}_p$, in particular for the choice of delay t, the definition of Pre_ℓ does not differentiate the two players with respect to their choice of delay. However, the correctness of the decomposition comes from the combination of this operator with Guard_δ (that clearly penalises Min if he chooses a delay such that the translated valuation does not satisfy the guard) and $\mathsf{Perturb}^p_\ell$ that allows Max to select a legal perturbation not satisfying the guard, leading to a value $+\infty$ afterwards.

For a mapping $\mathbb{V}\colon L \times \mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$ and a location ℓ, we can extract the submapping for the location ℓ, that we denote by $\mathbb{V}_\ell\colon \mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$, defined for all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ by $\mathbb{V}_\ell(\nu) = \mathbb{V}(\ell,\nu)$. Mappings $\mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$ can be compared, by using a pointwise comparison: in particular the maximum or minimum of two such mappings is defined pointwisely. The previous operators indeed allow us to split the computation of $\mathcal{F}_p$. We also rely on the classical notion of *regions*, as introduced in the seminal work on timed automata [2]. Indeed, for a given location ℓ, the set of deadlock valuations ν where $E(\ell,\nu) = \emptyset$ is a union of regions that we denote R_ℓ in the following, and that can easily be computed.

▶ **Lemma 11.** *For all $\mathbb{V}\colon L \times \mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$, $\ell \in L$, and $p > 0$, $\mathcal{F}_p(\mathbb{V})(\ell)$ equals*

$$\begin{cases} \nu \mapsto 0 & \text{if } \ell \in L_T \\ \left(\nu \mapsto \begin{cases} +\infty & \text{if } \nu \in R_\ell \\ \Big(\max\limits_{\delta=(\ell,g,Y,\ell')\in\Delta}\big[\mathsf{wt}(\delta) + \mathsf{Pre}_\ell(\mathsf{Guard}_\delta(\mathsf{Unreset}_Y(\mathbb{V}_{\ell'})))\big]\Big)(\nu) & \text{if } \nu \notin R_\ell \end{cases}\right) & \text{if } \ell \in L_{\mathsf{Max}} \\ \min\limits_{\delta=(\ell,g,Y,\ell')\in\Delta}\big[\mathsf{wt}(\delta) + \mathsf{Pre}_\ell(\mathsf{Perturb}^p_\ell(\mathsf{Guard}_\delta(\mathsf{Unreset}_Y(\mathbb{V}_{\ell'}))))\big] & \text{if } \ell \in L_{\mathsf{Min}} \end{cases}$$

5 Encoding parametric piecewise affine functions

We now explain how to encode the mappings that the operators defined in the previous section take as input, to compute $\mathcal{F}_p$ for all perturbation bounds $p > 0$ at once. We adapt the formalism used in [1, 18] to incorporate the perturbation p. This formalism relies on the remark that $\mathbb{V}^0$ is a piecewise affine function, and that if $\mathbb{V}$ is piecewise affine, so is $\mathcal{F}_p(\mathbb{V})$: thus we only have to manipulate such piecewise affine functions.

To model the robustness, that depends on the perturbation bound p, and maintain a parametric description of all the value functions for infinitesimally small values of p, we consider piecewise affine functions that depend on a formal parameter $\mathfrak{p}$ describing the perturbation. The pieces over which the function is affine, that we call cells in the following, are polytopes described by a conjunction of affine equalities and inequalities involving $\mathfrak{p}$. Some of our computations will only hold for small enough values of the parameter $\mathfrak{p}$ and we will thus also maintain an upperbound for this parameter.

▶ **Definition 12.** *We call* parametric affine expression *an expression E of the form $\sum_{x\in\mathcal{X}} \alpha_x\, x + \beta + \gamma\mathfrak{p}$ with $\alpha_x \in \mathbb{Q}$ for all $x \in \mathcal{X}$, $\beta \in \mathbb{Q} \cup \{-\infty, +\infty\}$, and $\gamma \in \mathbb{Q}$. The semantics of such an expression is given for a particular perturbation p as a mapping $[\![E]\!]_p\colon \mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$ defined for all $\nu \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ by $[\![E]\!]_p(\nu) = \sum_{x\in\mathcal{X}} \alpha_x\, \nu(x) + \beta + \gamma p$.*

A partition of $\mathbb{R}^{\mathcal{X}}_{\geq 0}$ into cells is described by a set $\mathcal{E} = \{E_1, \dots, E_m\}$ of parametric affine expressions. Every expression can be turned into an equation or inequation by comparing it to 0 with the symbol $=$, $<$ or $>$. The partition of $\mathbb{R}^{\mathcal{X}}_{\geq 0}$ is obtained by considering all

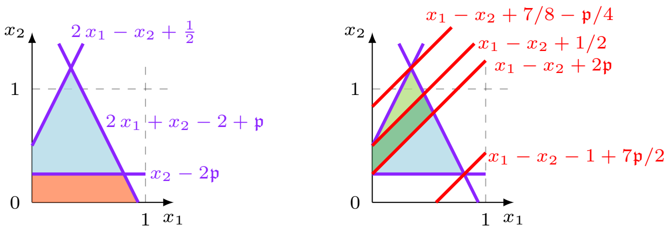

x_2
1
0
1
x_1
$2\,x_1 - x_2 + \frac{1}{2}$
$2\,x_1 + x_2 - 2 + \mathfrak{p}$
$x_2 - 2\mathfrak{p}$
x_2
1
0
1
x_1
$x_1 - x_2 + 7/8 - \mathfrak{p}/4$
$x_1 - x_2 + 1/2$
$x_1 - x_2 + 2\mathfrak{p}$
$x_1 - x_2 - 1 + 7\mathfrak{p}/2$

Figure 3 On the left, we depict the partition defined from $\mathcal{E} = \{x_2 - 2\mathfrak{p}, 2x_1 + x_2 - 2 + \mathfrak{p}, 2x_1 - x_2 + 1/2\}$, for a small enough value of $\mathfrak{p}$. On the right, we depict the atomic partition induced by $\mathcal{E}$, and draw in red the added parametric affine expressions.

the combinations of equations and inequations for each $1 \leq i \leq m$: such a combination is described by a tuple $(\bowtie_i)_{1\leq i\leq m}$ of symbols in $\{=,<,>\}$. For a given perturbation p, we let $\llbracket \mathcal{E}, (\bowtie_i)_{1\leq i\leq m}\rrbracket_p$ be the set of valuations ν such that for all $i \in \{1,\dots,m\}$, $\llbracket E_i\rrbracket_p \bowtie_i 0$.

We call *cell* every such combination such that for p that tends to 0, while being positive, the set $\llbracket \mathcal{E}, (\bowtie_i)_{1\leq i\leq m}\rrbracket_p$ is non empty. We let $\mathcal{C}(\mathcal{E})$ be the set of cells of $\mathcal{E}$. Notice that it can be decided (in at most exponential time) if a combination $(\bowtie_i)_{1\leq i\leq m}$ is a cell, by encoding the semantics in the first order theory of the reals, and deciding if there exists an upperbound $\eta > 0$ such that for all $0 < p \leq \eta$, $\llbracket \mathcal{E}, (\bowtie_i)_{1\leq i\leq m}\rrbracket_p$ is non empty. Moreover, we can compute the biggest such upperbound η if it exists. The upperbound of the partition $\mathcal{E} = \{E_1,\dots,E_m\}$ is then defined as the minimum such upperbound over all cells (there are at most 3^m cells), and denoted by $\eta(\mathcal{E})$ in the following. On the left of Figure 3, we depict the partition of $\mathbb{R}^{\mathcal{X}}_{\geq 0}$ defined from $\mathcal{E} = \{x_2 - 2\mathfrak{p}, 2x_1 + x_2 - 2 + \mathfrak{p}, 2x_1 - x_2 + 1/2\}$, with a fixed value of the perturbation. In blue, we color the cell defined by $(>,<,>)$. We note that this cell is non-empty when $p \leq 1/2$. By considering other cells, we obtain that $\eta(\mathcal{E}) = 1/2$.

In the following, we may need to record a smaller upperbound than $\eta(\mathcal{E})$, in order to keep the tightest constraint seen so far in the computation. We thus call *parametric partition* a pair $\langle \mathcal{E}, \eta\rangle$ given by a set of equations and a perturbation $\eta > 0$ that is at most $\eta(\mathcal{E})$.

For a cell $c \in \mathcal{C}(\mathcal{E})$, an expression E of $\mathcal{E}$ is said to be on the *border* of c if the removal of E from the set $\mathcal{E}$ of expressions forbids one to obtain the set of valuations $\llbracket c\rrbracket_p$ with the resulting cells for all small enough values of $p > 0$: more precisely, we require that no cell $c' \in \mathcal{C}(\mathcal{E} \setminus \{E\})$ is such that for some $p \leq \eta(\mathcal{E})$, $\llbracket c\rrbracket_p = \llbracket c'\rrbracket_p$. Because of the definition of $\eta(\mathcal{E})$, this definition does not depend on the actual value of p that we consider (and we could thus replace "for some" by "for all" above). On the left of Figure 3, all expressions are on the border for the blue cell, but only two of them are on the border of the orange cell.

The proofs that follow (in particular time delaying that requires to move along diagonal lines) requires to adapt the notion of *atomicity* of a parametric partition, originally introduced in the non-robust setting [1, 18]. A parametric affine expression $E = \sum_{x\in\mathcal{X}} \alpha_x\, x + \beta + \gamma p$ is said to be *diagonal* if $\sum_{x\in\mathcal{X}} \alpha_x = 0$: indeed, for all $p > 0$, $\llbracket E\rrbracket_p(\nu) = \llbracket E\rrbracket_p(\nu + t)$ for all $t \in \mathbb{R}$. A parametric partition is said to be *atomic* if for all cells $c \in \mathcal{C}(\mathcal{E})$, there are at most two non-diagonal parametric affine expressions on the border of c: intuitively, one border is reachable from every valuation by letting time elapse, and the other border is such that by letting time elapse from it we can reach all valuations of the cell. An atomic partition decomposes the space into tubes whose borders are only diagonal, each tube being then sliced by using only non-diagonal expressions. In particular, each cell c of an atomic partition has a finite set of cells that it can reach by time elapsing (and dually a set of cells that can reach c by time elapsing), and this set does not depend on the value of the parameter p, nor the starting valuation in $[\![c]\!]_p$. On the right of Figure 3, we depict the atomic partition associated with the same set of expressions used on the left. The diagonal expressions are depicted in red. We note that the cell colored in blue on the left is split into five cells that are non-empty when $p \leq 3/7$. We can describe the new parametric partition as $(\{x_2 - 2\mathfrak{p}, 2x_1 + x_2 - 2 + \mathfrak{p}, 2x_1 - x_2 + 1/2, x_1 - x_2 - 1 + 7\mathfrak{p}/2, x_1 - x_2 + 2\mathfrak{p}, x_1 - x_2 + 1/2, x_1 - x_2 + 7/8 - \mathfrak{p}/4\}, 3/7)$. As we will see below, a parametric partition can always be made atomic, by adding some diagonal parametric affine expressions.

A *parametric value function* (PVF for short) is a tuple $F = \langle \mathcal{E}, \eta, (f_c)_{c \in \mathcal{C}(\mathcal{E})} \rangle$ where $\langle \mathcal{E}, \eta \rangle$ is a partition and, for all cells $c \in \mathcal{C}(\mathcal{E})$, f_c is a parametric affine expression. For a perturbation $0 < p \leq \eta$, the semantics $[\![F]\!]_p$ of this tuple is a mapping $\mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$ defined for all valuations ν by $[\![F]\!]_p(\nu) = [\![f_c]\!]_p(\nu)$ where c is the unique cell such that $\nu \in [\![c]\!]_p$. A PVF is said to be *atomic* if its parametric partition is atomic. As announced above, we can always refine a PVF so that it becomes atomic.

▶ **Lemma 13.** *If $F = \langle \mathcal{E}, \eta, (f_c)_{c \in \mathcal{C}(\mathcal{E})} \rangle$ is a PVF, we can compute an atomic PVF $F' = \langle \mathcal{E}', \eta', (f'_c)_{c \in \mathcal{C}(\mathcal{E}')} \rangle$ such that $\eta' \leq \eta$, and $[\![F]\!]_p = [\![F']\!]_p$ for all $p \leq \eta'$.*

To conclude the proof of Theorem 9, we need to compute one application of $\mathcal{F}_p$ over a mapping $X \colon L \times \mathbb{R}^{\mathcal{X}}_{\geq 0} \to \mathbb{R}_\infty$ that is stored by a PVF for each location. Moreover, our computations must be done for all small enough values of p simultaneously.

▶ **Proposition 14.** *Let $F = \langle \mathcal{E}, \eta, (f_c)_{c \in \mathcal{C}(\mathcal{E})} \rangle$ be a PVF. We can compute a PVF $F' = \langle \mathcal{E}', \eta', (f'_c)_{c \in \mathcal{C}(\mathcal{E}')} \rangle$ with $\eta' \leq \eta$, and $[\![F']\!]_p = \mathcal{F}_p([\![F]\!]_p)$ for all $p \leq \eta'$.*

Sketch of the proof. By using Lemma 11, it suffices to perform the proof independently for the four operators, as well as maximum or minimum operations. Proofs from [1, 18] can be adapted for the maximum/minimum operations as well as the operators Guard_δ and $\mathsf{Unreset}_Y$ that exist also in the non-robust setting. In the case of Max, the two cases depend only on the regions and we thus only apply the various operators for starting valuations not in R_ℓ.

For the operator Pre_ℓ (and similarly $\mathsf{Perturb}^p_\ell$), the adaptation is more subtle. The key ingredient, for instance to compute it over an atomic partition for a location ℓ of Max, is to transform the computation of $([\![F]\!]_p)(\nu) = \sup_{t \geq 0}[t\,\mathsf{wt}(\ell) + [\![F]\!]_p(\nu + t)]$ involving a supremum (for a fixed valuation ν, and a fixed perturbation $p \leq \eta$), by using a maximum over a finite set of interesting delays. First, we remark that for every delay $t > 0$, the valuation $\nu + t$ belongs to the open diagonal half-line from ν, which crosses some of the semantics $[\![c']\!]_p$ for certain cells c'. Moreover, this finite (since there are anyway only a finite number of cells in the partition) subset of crossing cells neither depends on the choice of ν in a given starting cell c, nor on the perturbation p as long as it is at most η (by atomicity of the partition). Since the function $[\![F]\!]_p$ is affine in each cell, the above supremum over the possible delays is obtained for a value t that is either 0, or tending to $+\infty$, or on one of the two non-diagonal borders of the previous crossing cells, and we thus only have to consider those borders (that do not depend on the choice of ν in a given starting cell c, nor on the perturbation p). ◀

6 Divergent weighted timed games

From our algorithm to solve acyclic WTGs, we naturally want to extend the computation of the robust value to other classes of WTGs by using an unfolding of the WTG. In particular, we consider the natural extension of *divergent WTGs* (like in [16, 18]) that define a large class of decidable WTGs for the exact semantics, with no limitations on the number of clocks.

As usual in related work [1, 6, 7, 18], we now assume that all clocks are *bounded* by a constant $\mathsf{M} \in \mathbb{N}$, i.e. every transition of the WTG is equipped with a guard g such that $\nu \models g$ implies $\nu(x) \leq \mathsf{M}$ for all clocks $x \in \mathcal{X}$. We denote by W_{loc} (resp. W_{tr}, W_{e}) the maximal weight in absolute values of locations (resp. of transitions, edges) of $\mathcal{G}$, i.e. $W_{\mathsf{loc}} = \max_{\ell \in L} |\mathsf{wt}(\ell)|$ (resp. $W_{\mathsf{tr}} = \max_{\delta \in \Delta} |\mathsf{wt}(\delta)|$, $W_{\mathsf{e}} = \mathsf{M} W_{\mathsf{loc}} + W_{\mathsf{tr}}$).

We use the exact semantics to define the divergence property by relying once again on the regions [2]. We let $\mathsf{Reg}(\mathcal{X}, \mathsf{M})$ be the set of regions when clocks are bounded by M. A game $\mathcal{G}$ (w.r.t. the exact semantics) can be populated with the region information as described formally in [18]: we obtain the *region game* $\mathcal{R}(\mathcal{G})$. We call *region path* a finite or infinite sequence of transitions in this game, and we denote by π such paths. A play ρ in $\mathcal{G}$ can be projected on a region path π: we say that ρ *follows* the region path π. It is important to notice that, even if π is a *cycle* (i.e. starts and ends in the same location of the region game), there may exist plays following it in $\mathcal{G}$ that are not cycles, due to the fact that regions are sets of valuations.

Divergent WTGs are obtained by enforcing a semantical property of divergence (originally called *strictly non-Zeno cost* when only dealing with non-negative weights [6]): it asks that every play (w.r.t. the exact semantics) following a cyclic region path has weight far from 0. Formally, a cyclic region path π of $\mathcal{R}(\mathcal{G})$ is said to be a positive (resp. negative) if every finite play ρ following π satisfies $\mathsf{wt}_\Sigma(\rho) \geq 1$ (resp. $\mathsf{wt}_\Sigma(\rho) \leq -1$).

▶ **Definition 15.** *A WTG is divergent if every cyclic region path is positive or negative.*

Finally, with loss of generality, we only consider divergent WTGs containing no configurations with a value equal to $-\infty$. Intuitively, guaranteeing a value $-\infty$ resembles a Büchi condition for Min, since this means that Max cannot avoid the iteration of negative cycles with his delays. In the robust settings, testing Büchi condition for automata is already non-trivial [19], thus we remove this behaviour in our games in this article. Since the value is a lower bound of the robust value (by Lemma 7), we obtain that all locations have a robust value distinct from $-\infty$. Moreover, testing if such a location exists in a divergent WTG can be done in EXPTIME [18]. Our second contribution is to extend the symbolic algorithm used in the case of acyclic WTGs to compute the robust value in this subclass of divergent WTGs.

▶ **Theorem 16.** *The robust value problem is decidable over the subclass of divergent WTGs without configurations of value $-\infty$.*

Sketch of the proof. We compute the robust value by using an adaptation of the algorithm of [18] used to compute the (exact) value function in divergent WTGs. In particular, its termination is guaranteed by the use of an equivalent definition of divergent WTG requiring that for all strongly connected components (SCC) S of the graph of the region game, either every cyclic region path π inside S is positive (we say that the SCC is positive) or every cyclic region path π inside S is negative (we say that the SCC is negative).

We adapt this argument in the case of the computation of the robust value of a divergent WTG (without configurations with a value equal to $-\infty$). In particular, we observe that if a cyclic region path is positive (resp. negative) w.r.t. the exact semantics, then it is also positive (resp. negative) w.r.t. the conservative semantics, as the latter only filters some plays. Thus, the finite convergence of the value iteration algorithm (defined by $\mathcal{F}_p$ as for acyclic WTG, i.e. initialised by the function $\mathbb{V}^0$ defined such that $\mathbb{V}^0(\ell) = 0$ for all target locations, and $\mathbb{V}^0(\ell) = +\infty$ otherwise) is guaranteed by its finite convergence in finite time in each positive (resp. negative) SCC. Intuitively, in a positive (resp. negative) SCC, the interest of Min (resp. Max) is to quickly reach a target location of $\mathcal{G}$ to minimise the number of positive (resp. negative) cyclic region paths followed along the play that allow us to upperbound the number of iterations needed to obtain the robust value of all locations. Thus, the number of iterations needed to compute the robust value in a divergent WTG is defined by the sum of the number of iterations for each SCC along the longest path of the SCC decomposition. ◀

On top of computing the value, the modified algorithm allows one to synthesize almost-optimal strategies (we can adapt recent works [23] showing that those strategies can be taken among switching strategies for Min and memoryless strategies for Max).

7 Conclusion

This article allows one to compute (finite) robust values of weighted timed games in classes of games (acyclic and divergent) where the non-robust values are indeed computable.

As future works, we would like to carefully explore the exact complexity of our algorithms. Intuitively, each operator used to describe $\mathcal{F}_p$ can be computed in exponential time with respect to the set of cells and the size of η. By [18], the number of cells exponentially grows at each application of $\mathcal{F}_p$ (so it is doubly exponential for the whole computation) and the constants in affine expressions polynomially grow, in the non-robust setting. We hope that such upperbounds remain in the robust setting. This would imply that, for divergent WTGs, our algorithm requires a triply-exponential time, since the unfolding is exponential in the size of $\mathcal{G}$.

We also suggest to extend the setting to incorporate divergent WTGs that contain location with a value equal to $-\infty$. However, fixing it for all divergent WTGs seems to be difficult since, intuitively, the condition to guarantee $-\infty$ looks like a Büchi condition where Max can avoid the iteration of cycles with his delays. Moreover, we would like to study almost-divergent weighted timed games (studied in [17, 18]), a class of games undecidable, but with approximable value functions. We wonder if the robust values could also be approximated by similar techniques. Another direction of research would be to consider the fragment of one-clock weighted timed games, another class of games where the value function is known to be computable (for a long time in the non-negative case [5], very recently in the general case [24]). The difficulty here would be that encoding the conservative semantics in an exact semantics, even with fixed-perturbation, requires the addition of a clock, thus exiting the decidable fragment. The question thus becomes a possible adaptation of techniques used previously to solve non robust one-clock WTGs to incorporate directly the robustness.

References

1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004. doi:10.1007/978-3-540-27836-8_13.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

3 Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 19–30. Springer, 1999. doi:10.1007/3-540-48983-5_6.

4 Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2):135–175, 2007.

5 Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

6 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, LNCS, pages 148–160, Berlin, Heidelberg, 2005. Springer. doi:10.1007/978-3-540-30538-5_13.

7 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPIcs*, pages 311–324. Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.311.

8 Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata. In *Proceedings of the 7th Latin American Conference on Theoretical Informatics (LATIN'06)*, LNCS, pages 238–249. Springer, 2006. doi:10.1007/11682462_25.

9 Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust weighted timed automata and games. In *Proceedings of the 11th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'13)*, volume 8053 of *LNCS*, pages 31–46. Springer, August 2013. doi:10.1007/978-3-642-40229-6_3.

10 Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust reachability in timed automata: Game-based approach. *Journal of Theoretical Computer Science (TCS)*, 563:43–74, 2015.

11 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Formal Modeling and Analysis of Timed Systems*, LNCS, pages 49–64. Springer, 2005. doi:10.1007/11603009_5.

12 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. Simple priced timed games are not that simple. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 278–292. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.278.

13 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. One-clock priced timed games with negative weights. *Logical Methods in Computer Science*, 18(3), August 2022. doi:10.46298/lmcs-18(3:17)2022.

14 Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding negative prices to priced timed games. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *LNCS*, pages 560–575. Springer, 2014. doi:10.1007/978-3-662-44584-6_38.

15 Damien Busatto-Gaston. *Symbolic controller synthesis for timed systems: robustness and optimality.* PhD thesis, Aix-Marseille Université, 2019.

16 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal reachability in divergent weighted timed games. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2017)*, LNCS, pages 162–178. Springer, 2017. doi:10.1007/978-3-662-54458-7_10.

17 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Symbolic approximation of weighted timed games. In *Proceedings of the 38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'18)*, volume 122 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, December 2018. doi:10.4230/LIPIcs.FSTTCS.2018.28.

18 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal controller synthesis for timed systems. *Log. Methods Comput. Sci.*, 19(1), 2023. doi:10.46298/LMCS-19(1:20)2023.

19 Damien Busatto-Gaston, Benjamin Monmege, Pierre-Alain Reynier, and Ocan Sankur. Robust controller synthesis in timed Büchi automata: A symbolic approach. In *Proceedings of the 31st International Conference (CAV 2019)*, volume 11561 of *LNCS*, pages 572–590. Springer, 2019. doi:10.1007/978-3-030-25540-4_33.

20 Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robustness and implementability of timed automata. In *Proceedings of the International Conference on Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS'2004)*, LNCS, pages 118–133. Springer, 2004. doi:10.1007/978-3-540-30206-3_10.

21 Shibashis Guha, Shankara Narayanan Krishna, Lakshmi Manasa, and Ashutosh Trivedi. Revisiting robustness in priced timed games. In *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*, LIPIcs, pages 261–277. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:LIPIcs.FSTTCS.2015.261.

22 Marcin Jurdzinski and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *LNCS*, pages 838–849. Springer, 2007. doi:10.1007/978-3-540-73420-8_72.

23 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Playing Stochastically in Weighted Timed Games to Emulate Memory. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *LIPIcs*, pages 137:1–137:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.137.

24 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Decidability of one-clock weighted timed games with arbitrary weights. In *Proceedings of the 33rd International Conference on Concurrency Theory (CONCUR 2022)*, volume 243 of *LIPIcs*, pages 15:1–15:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CONCUR.2022.15.

25 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Synthesis of robust optimal strategies in weighted timed games. *CoRR*, abs/2403.06921, 2024. doi:10.48550/arXiv.2403.06921.

26 Youssouf Oualhadj, Pierre-Alain Reynier, and Ocan Sankur. Probabilistic robust timed games. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *LNCS*, pages 203–217. Springer, 2014. doi:10.1007/978-3-662-44584-6_15.

27 Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10:87–113, 2000. doi:10.1023/A:1008387132377.

28 Ocan Sankur. *Robustness in timed automata : analysis, synthesis, implementation. (Robustesse dans les automates temporisés : analyse, synthèse, implémentation)*. PhD thesis, École normale supérieure de Cachan, Paris, France, 2013.

29 Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking Timed Automata. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011)*, volume 13 of *LIPIcs*, pages 90–102. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.90.

30 Ocan Sankur, Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust controller synthesis in timed automata. In *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR 2013)*, volume 8052 of *LNCS*, pages 546–560. Springer, 2013. doi:10.1007/978-3-642-40184-8_38.

Edit and Alphabet-Ordering Sensitivity of Lex-Parse

Yuto Nakashima
Department of Informatics, Kyushu University, Fukuoka, Japan

Dominik Köppl
Department of Computer Science and Engineering, University of Yamanashi, Kofu, Japan
M&D Data Science Center, Tokyo Medical and Dental University, Japan

Mitsuru Funakoshi
NTT Communication Science Laboratories, Kyoto, Japan

Shunsuke Inenaga
Department of Informatics, Kyushu University, Fukuoka, Japan

Hideo Bannai
M&D Data Science Center, Tokyo Medical and Dental University, Japan

Abstract

We investigate the compression sensitivity [Akagi et al., 2023] of lex-parse [Navarro et al., 2021] for two operations: (1) single character edit and (2) modification of the alphabet ordering, and give tight upper and lower bounds for both operations (i.e., we show $\Theta(\log n)$ bounds for strings of length n). For both lower bounds, we use the family of Fibonacci words. For the bounds on edit operations, our analysis makes heavy use of properties of the Lyndon factorization of Fibonacci words to characterize the structure of lex-parse.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Compression sensitivity, Lex-parse, Fibonacci words

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.75

Related Version *Full Version*: `https://arxiv.org/abs/2402.19223`

Funding *Yuto Nakashima*: JSPS KAKENHI Grant Numbers JP21K17705, JP23H04386, and JST ACT-X Grant Number JPMJAX200K
Dominik Köppl: JSPS KAKENHI Grant Numbers JP23H04378
Shunsuke Inenaga: JSPS KAKENHI Grant Numbers JP20H05964, JP22H03551, JP23K24808
Hideo Bannai: JSPS KAKENHI Grant Numbers JP20H04141, JP24K02899

1 Introduction

Dictionary compression is a scheme of lossless data compression that is very effective, especially for highly repetitive text collections. Recently, various studies on dictionary compressors and repetitiveness measures have attracted much attention in the field of stringology (see [25, 26] for a detailed survey).

The *sensitivity* [1] of a string compressor/repetitiveness measure c is defined as the maximum difference in the sizes of c for a text T and for a single-character edited string T', which can represent the robustness of the compressor/measure w.r.t. small changes/errors of the input string. Akagi et al. [1] gave upper and lower bounds on the worst-case multiplicative sensitivity of various compressors and measures including the Lempel–Ziv parse family [29, 30], the run-length encoded Burrows-Wheeler transform (RLBWT) [4], and the smallest string attractors [17].

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 75; pp. 75:1–75:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the other hand, some structures built on strings, including the output of text compressors, can also depend on the order of the alphabet, meaning that a different alphabet ordering can result in a different structure for the same string. Optimization problems of these kinds of structures have recently been studied, e.g., for the RLBWT [2], the RLBWT based on general orderings [12], or the Lyndon factorization [13]. Due to their hardness, efficient exact algorithms/heuristics for minimization have been considered [5, 7, 8, 22, 23].

This paper is devoted to the analysis of the sensitivity of lex-parse. Lex-parse [27] is a greedy left-to-right partitioning of an input text T into phrases, where each phrase starting at position i is $T[i..i + \max\{1, \ell\})$ and ℓ is the longest common prefix between $T[i..n]$ and its lexicographic predecessor $T[i'..n]$ in the set of suffixes of T. Each phrase can be encoded by a pair $(0, T[i])$ if $\ell = 0$, or (ℓ, i') otherwise. By using the lex-parse of size v of a string, we can represent the string with v derivation rules.[1] Lex-parse was proposed as a new variant in a family of ordered parsings that is considered as a generalization of the Lempel–Ziv parsing and a subset of bidirectional macro schemes [28]. We stress that lex-parse can have much fewer factors than the Lempel–Ziv parsing; for instance the number of Lempel–Ziv factors of the k-th Fibonacci word is k while we have only four factors for lex-parse regardless of k (assuming that k is large enough) [27]. Besides having potential for lossless data compression, it helped to gain new insights into string repetitiveness: For instance, a direct relation $v \in O(r)$ between v and the size r of the RLBWT, one of the most important dictionary compressors, holds [27]. Hence, combinatorial studies on lex-parse can lead us to further understanding in string repetitiveness measures and compressors.

The contribution of this paper is twofold. We first consider the sensitivity of lex-parse w.r.t. edit operations, and give tight upper and lower bounds which are logarithmic in the length of the input text. Interestingly, lex-parse is the third measure with super-constant bounds out of (about) 20 measures [1]. Second, we consider a new variant of sensitivity, the alphabet ordering sensitivity (AO-sensitivity) of lex-parse, defined as the maximum difference in the number of phrases of lex-parse between any two alphabet orderings, and give tight upper and lower bounds. For both lower bounds, we use the Fibonacci word. Moreover, we also use properties of the Lyndon factorization [6] for the edit sensitivity to characterize the structure of lex-parse. These insights may be of independent interest. Properties of the Fibonacci word can contribute to the analysis of algorithm complexity. In fact, there are several results regarding lower bounds based on the Fibonacci word (i.e., [9, 14, 16, 27]).

Related work. Inspired by the results of Lagarde and Perifel [19], Akagi et al. [1] pioneered the systematic study of compression sensitivity of various measures w.r.t. edit operations. Giuliani et al. [14] showed an improved lower bound for the additive sensitivity of the run-length BWT. They also use the family of Fibonacci words to obtain their lower bound. Fujimaru et al. [11] presented tight upper and lower bounds for the additive and multiplicative sensitivity of the size of the compact directed acyclic word graph (CDAWG) [3, 10], when edit operations are restricted to the beginning of the text.

2 Preliminaries

Strings

Let Σ be an *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is the string of length 0. Let Σ^+ be the set of non-empty strings, i.e., $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For any strings x and y, let $x \cdot y$ (or sometimes xy) denote

[1] We stick to the convention to denote the size of lex-parse by v as done in literature such as [27, 26].

the concatenation of the two strings. For a string $w = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively. They are called a *proper prefix*, a *proper substring*, and a *proper suffix* of w if $x \neq w$, $y \neq w$, and $z \neq w$, respectively. A proper substring that is both a prefix and a suffix of w is also called a *border* of w. The i-th symbol of a string w is denoted by $w[i]$, where $1 \leq i \leq |w|$. For a string w and two integers $1 \leq i \leq j \leq |w|$, let $w[i..j]$ denote the substring of w that begins at position i and ends at position j. For convenience, let $w[i..j] = \varepsilon$ when $i > j$. Also, let $w[..i] = w[1..i]$ and $w[i..] = w[i..|w|]$. For a string w, let $w^1 = w$ and let $w^k = ww^{k-1}$ for any integer $k \geq 2$. A string w is said to be *primitive* if w cannot be written as x^k for any $x \in \Sigma^+$ and integer $k \geq 2$. The following property is well known.

▶ **Lemma 1** ([20]). *w is primitive if and only if w occurs exactly twice in w^2.*

For a string w, let $w' = w[1..|w|-1]$, $w'' = w[1..|w|-2]$. A sequence of k strings $w_1, \ldots, w_k$ is called a *parsing* or a *factorization* of a string w if $w = w_1 \cdots w_k$. For a binary string w, $\overline{w}$ denotes the bitwise reversed string of w (e.g., $\overline{aab} = baa$ over $\{a, b\}$). Let $\prec$ denote a (strict) total order on an alphabet Σ. A total order $\prec$ on the alphabet induces a total order on the set of strings called the *lexicographic order* w.r.t. $\prec$, also denoted as $\prec$, i.e., for any two strings $x, y \in \Sigma^*$, $x \prec y$ if and only if x is a proper prefix of y, or, there exists $1 \leq i \leq \min\{|x|, |y|\}$ s.t. $x[1..i-1] = y[1..i-1]$ and $x[i] \prec y[i]$. For a string w, let SA_w denote the *suffix array* of w, where the i-th entry $SA_w[i]$ stores the index j of the lexicographically i-th suffix $w[j..]$ of w.

Lex-parse

The lex-parse of a string w is a parsing $w = w_1, \ldots, w_v$, such that each phrase w_j starting at position $i = 1 + \sum_{k<j} |w_k|$ is $w[i..i + \max\{1, \ell\})$, where ℓ is the length of the longest common prefix between $w[i..]$ and its lexicographic predecessor $w[i'..]$ in the set of suffixes of w. Each phrase can be encoded by a pair $(0, w[i])$ if $\ell = 0$ (called an *explicit phrase*), or (ℓ, i') otherwise. We will call $w[i'..]$ the *previous suffix* of $w[i..]$. The size (the number of phrases) of the lex-parse of a string w will be denoted by $v(w)$. Note that a phrase starting at position i is explicit if and only if $w[i..]$ is the lexicographically smallest suffix starting with $w[i]$ and thus there are $|\Sigma|$ of them. Let $w = ababbaaba$. The lex-parse of w is aba, b, ba, a, b, a. Since the previous suffix of $w[1..]$ is $w[7..]$ and the longest common prefix between them is aba, the first phrase is aba. In this example, the last two phrases are explicit phrases.

Lyndon factorizations

A string w is a *Lyndon word* [21] w.r.t. a lexicographic order $\prec$, if and only if $w \prec w[i..]$ for all $1 < i \leq |w|$, i.e., w is lexicographically smaller than all its proper suffixes with respect to $\prec$. The *Lyndon factorization* [6] of a string w, denoted $LF(w)$, is a unique factorization $\lambda_1^{p_1}, \ldots, \lambda_m^{p_m}$ of w, such that each $\lambda_i \in \Sigma^+$ is a Lyndon word, $p_i \geq 1$, and $\lambda_i \succ \lambda_{i+1}$ for all $1 \leq i < m$. A suffix x of w is said to be *significant* if there exists an integer i such that $x = \lambda_i^{p_i} \cdots \lambda_m^{p_m}$ and, for every j satisfying $i \leq j < m$, $\lambda_{j+1}^{p_{j+1}} \cdots \lambda_m^{p_m}$ is a prefix of $\lambda_j^{p_j}$ (cf. [15]). Let $w = bbabbaabaabbaabaabbaabaa$. The Lyndon factorization of w is b^2, abb, $(aabaabb)^2$, aab, a^2. The suffix a^2 that is the last Lyndon factor is always significant. Since a^2 is a prefix of $(aab)a^2$, $(aab)a^2$ is also significant. Since a^2 and $(aab)a^2$ is a prefix of $(aabaabb)^2(aab)a^2$, $(aabaabb)^2(aab)a^2$ is also significant.

Table 1 Fibonacci words F_k up to $k = 8$.

k	F_k	k	F_k
1	b	2	a
3	ab	4	aba
5	$abaab$	6	$abaababa$
7	$abaababaabaab$	8	$abaababaabaababaababa$

Fibonacci words

The k-th (finite) Fibonacci word F_k over a binary alphabet $\{a, b\}$ is defined as follows: $F_1 = b$, $F_2 = a$, $F_k = F_{k-1} \cdot F_{k-2}$ for any $k \geq 3$ (see also an example in Table 1). Let f_k be the length of the k-th Fibonacci word (i.e., $f_k = |F_k|$).

We also use the infinite Fibonacci word $\mathcal{F} = \lim_{k\to\infty} F_k$ over an alphabet $\{a, b\}$. We also use $G_k = F_{k-2}F_{k-1}$ which will be useful for representing relations between even and odd Fibonacci words.

▶ **Lemma 2** (Useful properties of Fibonacci words (cf. [27])). *The following properties hold for every Fibonacci word F_k.*

1. *The length of the longest border of F_k is f_{k-2}.*
2. *F_k has exactly three occurrences of F_{k-2} at position 1, $f_{k-2} + 1$, and $f_{k-1} + 1$ (suffix) for every $k \geq 6$.*
3. $F_k = G_k[1..f_k - 2] \cdot \overline{G_k[f_k - 1..f_k]}$.
4. $G_k = F_k[1..f_k - 2] \cdot \overline{F_k[f_k - 1..f_k]}$.
5. *For every k, aaa and bb do not occur as substrings in F_k [27, Lemma 36].*
6. *F_k is primitive for every k (we can easily obtain the fact from Property 1).*

The next lemma is also useful for our proof which can be obtained from the above properties.

▶ **Lemma 3.** *For any $k \geq 8$, F_{k-4} occurs exactly eight times in F_k.*

Proof. From property 2 of Lemma 2, F_k has at least eight occurrences of F_{k-4} (since the suffix occurrence of F_{k-4} in the second F_{k-2} and the prefix occurrence of F_{k-4} in the third F_{k-2} are the same position). Suppose to the contrary that there exists an occurrence of F_{k-4} in F_k that is different from the eight occurrences. From property 2 of Lemma 2, the occurrence crosses the boundary of the first and the second F_{k-2}. Since F_{k-4} is both a prefix and a suffix of F_{k-2}, the occurrence implies that F^2_{k-4} has F_{k-4} as a substring that is neither a prefix nor a suffix. This contradicts Lemma 1. ◀

Sensitivity of lex-parse

In this paper, we consider two compression sensitivity variants of lex-parse. The first variant is the sensitivity by (single character) edit operations (cf. [1]). For any two strings w_1 and w_2, let $\mathsf{ed}(w_1, w_2)$ denote the edit distance between w_1 and w_2. The worst-case multiplicative sensitivity of lex-parse w.r.t. a substitution is defined as follows:

$$\mathsf{MS}_{\mathtt{sub}}(v, n) = \max_{w_1 \in \Sigma^n} MS_{\mathtt{sub}}(v, w_1),$$

where $MS_{\mathtt{sub}}(v, w_1) = \max\{v(w_2)/v(w_1) \mid w_2 \in \Sigma^n, \mathsf{ed}(w_1, w_2) = 1\}$. $\mathsf{MS}_{\mathtt{ins}}(v, n)$ (resp. $\mathsf{MS}_{\mathtt{del}}(v, n)$) is defined by replacing the condition $w_2 \in \Sigma^n$ with $w_2 \in \Sigma^{n+1}$ (resp. $w_2 \in \Sigma^{n-1}$). The second variant is the sensitivity by alphabet orderings. For any string w and a lexicographic order $\prec$, let $v(w, \prec)$ be the size of the lex-parse of w under $\prec$. We define the alphabet-ordering sensitivity of lex-parse as follows:

$$\mathsf{AOS}(v, n) = \max_{w \in \Sigma^n} AOS(v, w),$$

where $AOS(v, w) = \max_{\prec_1, \prec_2 \in A}\{v(w, \prec_2)/v(w, \prec_1)\}$ and A is the set of orderings over Σ.

3 Upper bounds

We first give upper bounds for both operations. We can obtain the following result by using known connections regarding the bidirectional macro scheme and lex-parse.

▶ **Theorem 4.** $\mathsf{MS}_{\mathtt{sub}}(v, n), \mathsf{MS}_{\mathtt{ins}}(v, n), \mathsf{MS}_{\mathtt{del}}(v, n), \mathsf{AOS}(v, n) \in O(\log n)$.

Proof. For any string w, let $b(w)$ be the size of the smallest bidirectional macro scheme [28]. Then, $v(w) \geq b(w)$ holds [27, Lemma 25]. For any two strings w_1 and w_2 with $\mathsf{ed}(w_1, w_2) = 1$, $v(w_2) \in O(b(w_2) \log(n/b(w_2)))$ [27, Theorem 26] and $b(w_2) \leq 2b(w_1)$ [1, Theorem 11] hold. Hence, for $|w_2| \in \Theta(n)$,

$$\frac{v(w_2)}{v(w_1)} \leq \frac{v(w_2)}{b(w_1)} \in O\left(\frac{b(w_2)\log(n/b(w_2))}{b(w_1)}\right) \subseteq O\left(\frac{b(w_1)\log(n/b(w_1))}{b(w_1)}\right) = O(\log(n/b(w_1))).$$

For any alphabet order $\prec$ on Σ, $v(w, \prec) \in O(b(w) \log(n/b(w)))$ and $v(w, \prec) \geq b(w)$ holds. Hence, for any two alphabet orders $\prec_1$ and $\prec_2$,

$$\frac{v(w, \prec_2)}{v(w, \prec_1)} \leq \frac{v(w, \prec_2)}{b(w)} \in O\left(\frac{b(w)\log(n/b(w))}{b(w)}\right) = O(\log(n/b(w))).$$

These facts imply this theorem. ◀

4 Lower bounds for edit operations

In this section, we give tight lower bounds for edit operations with the family of Fibonacci words.

▶ **Theorem 5.** $\mathsf{MS}_{\mathtt{sub}}(v, n), \mathsf{MS}_{\mathtt{ins}}(v, n), \mathsf{MS}_{\mathtt{del}}(v, n) \in \Omega(\log n)$.

We devote this section to show $\mathsf{MS}_{\mathtt{sub}}(v, n) \in \Omega(\log n)$ since a similar argument can obtain the others. We obtain the claimed lower bound by combining the result of [27] (also in Lemma 20 in Section 5) stating that $v(F_{2k}) = 4$, and the following Theorem 6.

▶ **Theorem 6.** *For every integer $k \geq 6$, there exists a string w of length f_{2k} such that* $\mathsf{ed}(F_{2k}, w) = 1$ *and $v(w) = 2k - 2$.*

Let T_{2k} denote the string obtained from F_{2k} by substituting the rightmost b of F_{2k} with a, i.e., $T_{2k} = F''_{2k} \cdot aa$. We show that the lex-parse of T_{2k} has $2k - 2$ phrases. More specifically, we show that the lengths of the phrases are

$$f_{2k-1} - 1, f_{2k-4} - 1, f_{2k-5} + 1, ..., f_4 - 1, f_3 + 1, 1, 2, 1.$$

There are three types of phrases as follows: (1) first phrase, (2) inductive phrases, (3) last three short phrases. Phrases of Type (1) or Type (3) can be obtained by simple properties of Fibonacci words. However, phrases of Type (2) need a more complicated discussion. We use the Lyndon factorizations of the strings to characterize the inductive phrases. Intuitively, we show that every suffix of T_{2k} that has an odd inductive phrase as a prefix can be written as the concatenation of the odd inductive phrase and a significant suffix.

(1) First phrase and (3) Short phrases

We start from Type (1). From the third property of Lemma 2 and the edit operation, $T_{2k}[1..]$ and $T_{2k}[f_{2k-2}+1..]$ have a (longest) common prefix $x = F'_{2k-1}$ of length $f_{2k-1}-1$ and $T_{2k}[1..] \succ T_{2k}[f_{2k-2}+1..]$ holds. $T_{2k}[f_{2k-2}+1..]$ can be written as $T_{2k}[f_{2k-2}+1..] = x \cdot a$. We show that the previous suffix of $T_{2k}[1..]$ is $T_{2k}[f_{2k-2}+1..]$. Suppose to the contrary that there exists a suffix y of F_{2k} that satisfies $T_{2k}[1..] \succ y \succ T_{2k}[f_{2k-2}+1..]$. Since both $T_{2k}[f_{2k-2}+1..]$ and $T_{2k}[1..]$ have x as a prefix, y can be written as $y = x \cdot a \cdot z_1$ or $y = x \cdot b \cdot z_2$ for some strings z_1, z_2. Since F_{2k-1} ends with aab and thus aa is a suffix of x, the existence of $y = x \cdot a \cdot z_1$ contradicts the fact that a^3 only occurs at the edit position. On the other hand, the existence of $y = x \cdot b \cdot z_2$ contradicts the fact that $x \cdot b = F_{2k-1}$ only occurs as a prefix of T_{2k}, since otherwise it would violate the second property of Lemma 2. Thus $T_{2k}[f_{2k-2}+1..]$ is the previous suffix of $T_{2k}[1..]$, and the length of the first phrase is $|x| = f_{2k-1}-1$.

Next, we consider Type (3) phrases. Since T_{2k} ends with $baaa$ and no Fibonacci word has aaa as a substring, we conclude that $SA_{T_{2k}}[1] = f_{2k}$, $SA_{T_{2k}}[2] = f_{2k}-1$, and $SA_{T_{2k}}[3] = f_{2k}-2$. In particular, ba^3 is the smallest suffix of T_{2k} that begins with b. These facts imply that $T_{2k}[f_{2k}] = a$ and $T_{2k}[f_{2k}-3] = b$ are the explicit phrases, and $T_{2k}[f_{2k}-2..f_{2k}-1] = a^2$ between the explicit phrases is a phrase. Thus the last three phrases are b, a^2, a.

(2) Inductive phrases

In the rest of this section, we explain Type (2) phrases. Firstly, we study the Lyndon factorization $LF(\mathcal{F}) = \ell_1, \ell_2, \dots$ of the (infinite) Fibonacci word. To characterize these Lyndon factors ℓ_i, we use the string morphism ϕ with $\phi(a) = aab, \phi(b) = ab$ as defined in [24, Proposition 3.2].

▶ **Lemma 7** ([24]). *$\ell_1 = ab, \ell_{k+1} = \phi(\ell_k)$, and $|\ell_k| = f_{2k+1}$ holds.*

In order to show our result, we consider the Lyndon factorization of $F''_{2k} (= T''_{2k})$ (Lemma 10). Lemmas 8 and 9 explain the Lyndon factorization of a finite prefix of the (infinite) Fibonacci word by using properties of the morphism ϕ.

▶ **Lemma 8.** *Given a string $w \in \{a,b\}^+$, let $LF(w) = \lambda_1^{p_1}, \dots, \lambda_m^{p_m}$. Then $LF(\phi(w)) = \phi(\lambda_1^{p_1}), \dots, \phi(\lambda_m^{p_m})$.*

Proof. For any two binary strings x and y, it is clear that $\phi(x) \succ \phi(y)$ if $x \succ y$. In the rest of this proof, we show that $\phi(x)$ is a Lyndon word for a Lyndon word x over $\{a,b\}$. If $|x| = 1$, then the statement clearly holds. Suppose that $|x| \geq 2$. Let $\tilde{x}$ be a non-empty proper suffix of $\phi(x)$. Then $\tilde{x}$ can be represented as $\tilde{x} = \phi(y)$ for some suffix y of x, or $\tilde{x} = \alpha \cdot \phi(y)$ for some suffix y of x and $\alpha \in \{ab, b\}$. Since $x \prec y$, $\phi(x) \prec \tilde{x}$ if $\tilde{x} = \phi(y)$ for some suffix y of x. Also in the case of $\tilde{x} = \alpha \cdot \phi(y)$, $\phi(x) \prec \tilde{x}$ holds since $x[1] = a$ (from x is a Lyndon word) and $\phi(x)[1..3] = aab$. Thus, $\phi(x) \prec \tilde{x}$ holds for all non-empty proper suffixes $\tilde{x}$ of x. This implies that $\phi(x)$ is a Lyndon word. Therefore, the statement holds. ◀

▶ **Lemma 9.** *For every integer $i \geq 1$, $LF(\ell'_i) = \phi^{i-1}(a), \ldots, \phi^0(a)$, where $\ell'_i = \ell_i[..|\ell_i| - 1]$.*

Proof. We prove the statement by induction on i. For the base case $i = 1$, $LF(\ell'_1) = a = \phi^0(a)$. Assume that the lemma holds for all $i \leq j$ for some $j \geq 1$. By the definition of the morphism ϕ, $\ell'_{j+1} = \phi(\ell_j)' = \phi(\ell'_j) \cdot a$ holds because ℓ_j ends with b. Also, by using Lemma 8 and the induction hypothesis,

$$LF(\ell'_{j+1}) = LF(\phi(\ell'_j) \cdot a) = LF(\phi(\ell'_j)), a = \phi(LF(\ell'_j)), a = \phi^j(a), \ldots, \phi^1(a), \phi^0(a).$$

Therefore, the lemma holds. ◀

▶ **Lemma 10.** *Let L_i be the i-th Lyndon factor of F''_{2k}. For every $k \geq 2$,*

$$L_i = \begin{cases} \phi^{i-1}(ab) & \text{if } 1 \leq i < k-1, \\ \phi^{2k-i-3}(a) & \text{if } k-1 \leq i \leq 2k-3. \end{cases}$$

Proof. From Lemma 7,

$$\sum_{i=1}^{k-1} |\ell_i| = \sum_{i=1}^{k-1} f_{2i+1} = f_3 + f_5 + \cdots + f_{2k-1} = f_2 + (f_3 + f_5 + \cdots + f_{2k-1}) - 1 = f_{2k} - 1.$$

Thus $LF(F_{2k}) = \ell_1, \ldots, \ell_{k-1}, a$ holds. Also, $L_i = \ell_i$ holds for all $i < k-1$ since ℓ_i is not a prefix of ℓ_{i-1}. In other words, $LF(F''_{2k}) = \ell_1, \ldots, \ell_{k-2}, LF(\ell'_{k-1})$. From Lemma 9,

$$LF(F''_{2k}) = \ell_1, \ldots, \ell_{k-2}, \phi^{k-2}(a), \ldots, \phi^0(a).$$

Then the statement also holds. ◀

Moreover, we can find the specific form of suffixes characterized by the Lyndon factorization of F''_{2k} as described in the next lemma.

▶ **Lemma 11.** *For every integer $i \geq 1$, $\phi^{i-1}(a) \cdots \phi^0(a)$ is a prefix of $\phi^i(a)$.*

Proof. We prove the lemma by induction on i. For the base case $i = 1$, $\phi^0(a) = a$ is a prefix of $\phi^1(a) = aab$. Assume that the statement holds for all $i \leq j$ for some $j \geq 1$. For $i = j+1$,

$$\phi^{j+1}(a) = \phi^j(\phi(a)) = \phi^j(aab) = \phi^j(a)\phi^j(a)\phi^j(b).$$

By induction hypothesis, $\phi^j(a) = \phi^{j-1}(a) \cdots \phi^0(a) \cdot x$ for some string x. Then $\phi^{j+1}(a) = \phi^j(a) \cdot \phi^{j-1}(a) \cdots \phi^0(a) \cdot x \cdot \phi^j(b)$. This implies that the statement also holds for $i = j+1$. Therefore, the lemma holds. ◀

With the Lemmas 10 and 11, we obtain the following lemma, which characterizes the first $k+1$ entries of the suffix array $SA_{T_{2k}}$ of T_{2k}.

Firstly, we consider the order of the significant suffixes of F''_{2k}.

▶ **Lemma 12.** *$SA_{F''_{2k}}[i] = f_{2k} - 1 - \sum_{j=0}^{i-1} |\phi^j(a)|$ for every $i \in [1..k-1]$.*

Proof. We can see that $\phi^{k-2}(a) \cdots \phi^0(a)$ is a suffix of F''_{2k} by Lemma 10. Our claim is that $\phi^{i-1}(a) \cdots \phi^0(a)$ is the i-th lexicographically smallest suffix of F''_{2k} for every $0 \leq i \leq k-1$. We prove the statement by induction on i. For the base case $i = 1$, $\phi^0(a) = a$ is the lexicographically smallest suffix of F''_{2k}. Assume that the statement holds for all $i \leq i'$ for some $i' \geq 1$. Let α be a suffix of $\phi^{i'+1}(a) \cdots \phi^0(a)$ such that there is no d with $\alpha = \phi^d(a) \cdots \phi^0(a)$. (Otherwise we already know that α is a prefix of $\phi^{i'+1}(a)$, which we already processed for a

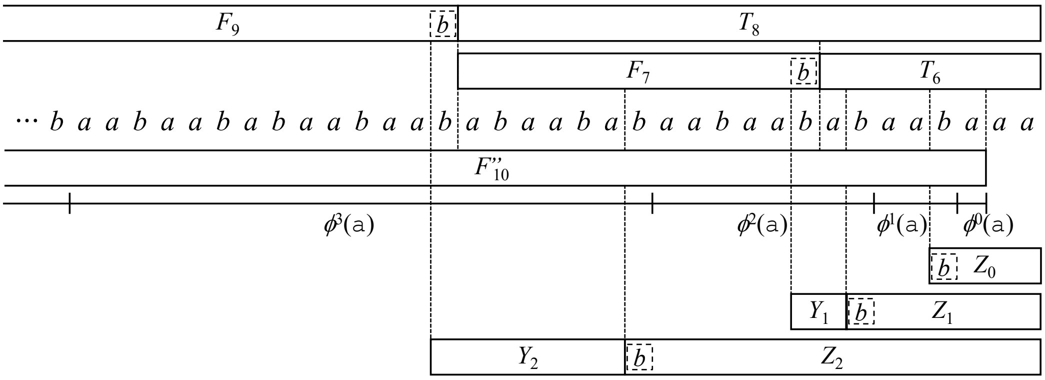


Figure 1 Illustration of the edited string T_{2k} and the Lyndon factorization of F''_{2k} (when $k = 5$).

former suffix array entry.) Then α can be written as $\alpha = \beta \cdot \phi^d(a) \cdots \phi^0(a)$ for some proper suffix β of $\phi^{d+1}(a)$ and $d \in [0..i']$. Since $\phi^{i'+1}(a)$ has $\phi^{d+1}(a)$ as a prefix and there is a mismatching character between $\phi^{d+1}(a)$ and β (from β is a suffix of Lyndon word $\phi^{d+1}(a)$), then $\phi^{i'+1}(a) \prec \beta$ holds. Therefore, the lemma holds. ◀

Because appending a's to the end does not affect suffix orders, we can easily obtain the following lemma from Lemma 12.

▶ **Lemma 13.** *For every $k \geq 2$, $SA_{T_{2k}}[1] = f_{2k}$, $SA_{T_{2k}}[2] = f_{2k} - 1$, and $SA_{T_{2k}}[i] = f_{2k} - 1 - \sum_{j=0}^{i-3} |\phi^j(a)|$ for any i that satisfying $3 \leq i \leq k + 1$.*

Proof. T_{2k} ends with the suffix aa such that $T[f_{2k}..] = a$ is the smallest, and $T[f_{2k} - 1..]$ is the second smallest suffix of T_{2k}. All other suffixes inherit their order from F''_{2k}, and thus $SA_{T_{2k}}[i] = SA_{F''_{2k}}[i - 2]$ for $i \geq 3$. ◀

We define a substring Y_i and suffixes X_i and Z_i of T_{2k} as follows:

$$X_i = T_{2k}[f_{2k} - f_{2i+4}..] \ (1 \leq i \leq k - 3),$$
$$Y_i = T_{2k}[f_{2k} - f_{2i+4}..f_{2k} - f_{2i+3} - 2] \ (1 \leq i \leq k - 3),$$
$$Z_i = T_{2k}[f_{2k} - f_{2i+3} - 1..] \ (0 \leq i \leq k - 3).$$

▶ **Observation 14.** *The following properties hold:*

$$X_i = Y_i \cdot Z_i = b \cdot T_{2i+4},$$
$$Y_i = b \cdot T''_{2i+2},$$
$$Z_i = b \cdot \phi^i(a) \cdots \phi^0(a) \cdot aa.$$

Notice that $|\phi^i(a) \cdots \phi^0(a)| = |\ell'_{i+1}| = f_{2i+3} - 1$ holds. See also Fig. 1 for an illustration of the specific substrings. Then Lemma 13 implies the following corollary.

▶ **Corollary 15.** *For every integer $k \geq 2$ and i satisfying $1 \leq i \leq k - 2$, the previous suffix of Z_i w.r.t. T_{2k} is Z_{i-1}.*

The largest suffix of T_{2k}, denoted by maxsuf, is characterized in the following lemma. We also use this suffix in the main lemma. Intuitively, we show that every suffix of T_{2k} that has an even inductive phrase as a prefix references a suffix that is the concatenation of a string and the maximum suffix (Lemma 18).

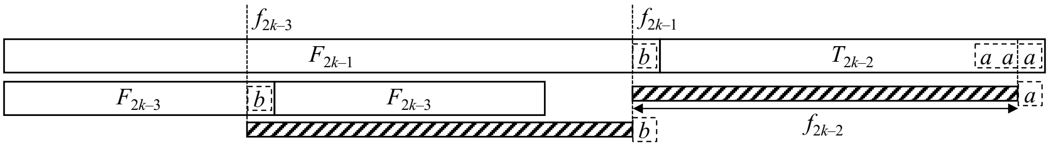


Figure 2 Illustration of T_{2k} for proof of Lemma 16.

▶ **Lemma 16.** *The lexicographically largest suffix* maxsuf *of* T_{2k} *is* $T_{2k}[f_{2k-3}..]$.

Proof. It is known that the lexicographically largest suffix of F_{2k} is $F_{2k}[f_{2k-1}..]$ and the lexicographically second largest suffix of F_{2k} is $F_{2k}[f_{2k-3}..]$ (shown in [18]). Namely, $SA_{F_{2k}}[f_{2k}] = f_{2k-1}, SA_{F_{2k}}[f_{2k} - 1] = f_{2k-3}$. Due to the edit operation, $T_{2k}[f_{2k-3}..] \succ T_{2k}[f_{2k-1}..]$ and the length of the longest prefix of these suffixes is f_{2k-2} (see Fig. 2). Assume on the contrary that there is a suffix $T_{2k}[i..]$ that is lexicographically larger than $T_{2k}[f_{2k-3}..]$. With Property 5 of Lemma 2, the suffix aaa of T_{2k} acts as a unique delimiter such that the suffix $T_{2k}[f_{2k-3}..]$ cannot be a prefix of any other suffixes of T_{2k}. Let j be the smallest positive integer such that $T_{2k}[f_{2k-3}..f_{2k-3} + j] = a$ and $T_{2k}[i..i + j] = b$. If $\max(i + j, f_{2k-3} + j) < f_{2k} - 1$, then $F_{2k}[f_{2k-3}..] \prec F_{2k}[i..]$ holds. This contradicts the fact that the lexicographically second largest suffix of F_{2k} is $F_{2k}[f_{2k-3}..]$. We can observe that there is no j with $\max(i + j, f_{2k-3} + j) \geq f_{2k} - 1$ and $i > f_{2k-3}$ since $T_{2k}[f_{2k} - 1..] = aa$ does not contain b. If $\max(i + j, f_{2k-3} + j) \geq f_{2k} - 1$ and $i < f_{2k-3}$, F_{2k-3} has a beginning position d of an occurrence satisfying $2 \leq d \leq f_{2k-3}$. This contradicts Lemma 2. Therefore, the lemma holds. ◀

To prove Lemma 18, we also introduce the following corollary.

▶ **Corollary 17** (of Lemma 2)**.** *For every i satisfying $i \geq 6$, T_i has exactly two occurrences of F_{i-2}. Moreover, T_i can be written as $T_i = F_{i-2} \cdot F_{i-2} \cdot w$ for some string w.*

▶ **Lemma 18.** *For every $k \geq 2$ and i satisfying $1 \leq i \leq k - 3$, the previous suffix of X_i w.r.t. T_{2k} is $Y_i \cdot a \cdot$* maxsuf*.*

Proof. Firstly, we show that $Y_i \cdot a \cdot \mathsf{maxsuf}$ is a suffix of T_{2k}. It is clear from definitions and properties of Lemma 2 that T_{2k} can be written as $T_{2k} = F'_{2k-3} \cdot \mathsf{maxsuf} = F''_{2k-3} \cdot a \cdot \mathsf{maxsuf} = F_{2k-4} \cdot F''_{2k-5} \cdot a \cdot \mathsf{maxsuf} = F_{2k-5} \cdot F''_{2k-4} \cdot a \cdot \mathsf{maxsuf}$. Since $Y_{k-3} = b \cdot T''_{2k-4} = b \cdot F''_{2k-4}$ and the last character of F_{2k-5} is b (from $2k - 5$ is odd), $Y_{k-3} \cdot a \cdot \mathsf{maxsuf}$ is a suffix of T_{2k}. Moreover, F_{2i+2} is a suffix of F_{2k-4} for every i that satisfying $1 \leq i \leq k - 3$. This implies that $Y_i \cdot a \cdot \mathsf{maxsuf}$ is a suffix of T_{2k} for every i that satisfying $1 \leq i \leq k - 3$.

Now we go back to our main proof of the lemma. Since $X_i = Y_i \cdot b \cdot \phi^i(a) \cdots \phi^0(a) \cdot aa$, $Y_i \cdot a \cdot \mathsf{maxsuf} \prec X_i$ holds. Moreover, $Y_i \cdot a \cdot \mathsf{maxsuf}$ is the lexicographically largest suffix of T_{2k} that has $Y_i \cdot a$ as a prefix. Hence, it is sufficient to prove that X_i is the lexicographically smallest suffix of T_{2k} that has $Y_i \cdot b$ as a prefix. From Property 5 of Lemma 2, no bb occurs in T_{2k}, so every occurrence of $Y_i \cdot b$ is also an occurrence of $Y_i \cdot ba$. We consider occurrences of $Y_i \cdot ba$ in a suffix X_i. From Observation 14 and Corollary 17, $(Y_i \cdot ba)[2..](= F_{2i+2})$ has exactly two occurrences in $X_i(= b \cdot T_{2i+4})$. At the first occurrence, $(Y_i \cdot ba)[2..]$ is preceded by b. At the second occurrence, $(Y_i \cdot ba)[2..]$ is preceded by a. Hence, the rightmost occurrence of $Y_i \cdot ba$ in T_{2k} is at the prefix of X_i. By combining with Lemma 13, we can see that there is no suffix w such that $Y_i \cdot ba \cdot w \prec X_i$ holds. Thus, X_i is the lexicographically smallest suffix of T_{2k} that has $Y_i \cdot b$ as a prefix. Therefore, the lemma holds. ◀

Lex-parse of T_{2k}

Now we can explain the lex-parse of T_{2k}. Recall that the length of the first phrase is $f_{2k-1}-1$ and the last three phrases are b, a^2, a. Hence, from Lemma 18, the second phrase is a prefix Y_{k-3} of $X_{k-3}(= b \cdot T_{2k-2})$. Since the remaining suffix is Z_{k-3}, the next phrase is a prefix $Z_{k-4}[..|Z_{k-4}|-1]$ of Z_{k-3} from Corollary 15. By applying this argument repeatedly, we can finally obtain the lex-parse of size $2k-2$ of T_{2k} as follows:

$$F_{2k}[..f_{2k-1}-1], (Y_{k-3}, Z_{k-4}[..|Z_{k-4}|-1]), \ldots, (Y_1, Z_0[..|Z_0|-1]), b, a^2, a.$$

Furthermore, we can easily obtain $v(F''_{2k} \cdot a) = 2k-2$ for a delete operation. Consider the case for the insertion operation such that \$ (which is the smallest character) is inserted to the preceding position of the last b. We can then show that the lex-parse is of size $2k+1$ by a similar argument as follows:

$$F_{2k}[1..f_{2k-1}-2], (a \cdot Y_{k-3}, Z_{k-4}[..|Z_{k-4}|-2]), \ldots, (a \cdot Y_1, Z_0[..|Z_0|-2]), a, ba, \$, b, a.$$

5 Lower bounds for Alphabet-Ordering

In this section, we give tight lower bound $\mathsf{AOS}(v,n) \in \Omega(\log n)$ with the family of Fibonacci words. Since $b(F_k) \leq 4$ for $k \geq 5$ [27, Lemma 35] also holds, our lower bound is tight for any $n \in \{f_i\}_i$. More precisely, we prove the following theorem that determines the number of lex-parse phrases of the Fibonacci words on any alphabet ordering.

▶ **Theorem 19.** *For any $k \geq 6$,*

$$v(F_k, \prec) = \begin{cases} \lceil \frac{k}{2} \rceil + 1 & \textit{(a) if } k \textit{ is odd and } a \prec b, \quad (\textit{Lemma } 23) \\ 4 & \textit{(b) if } k \textit{ is odd and } b \prec a, \quad (\textit{Lemma } 24) \\ 4 & \textit{(c) if } k \textit{ is even and } a \prec b, \quad (\textit{Lemma } 20) \\ \lceil \frac{k}{2} \rceil + 1 & \textit{(d) if } k \textit{ is even and } b \prec a. \quad (\textit{Lemma } 25) \end{cases}$$

Although the results for a smaller than b have been proven by Navarro et al. [27], we here give alternative proofs for this case that leads us to the proof for the case when b is smaller than a.

5.1 Lex-parse with constant number of phrases

We start with Cases (b) and (c). Since $F_k[f_k-1..f_k] = ba$ for even k and $F_k[f_k-1..f_k] = ab$ for odd k, we already know in the cases (b) and (c) that each of the last two characters forms an explicit phrase. It is left to analyze the non-explicit phrases, where we start with the first phrase. From Property 1 of Lemma 2, F_k has the border F_{k-2} and thus $F_{k-2} = F_k[f_{k-1}..] \prec F_k$ could be used as the reference for the first phrase, given its length is f_{k-2}. To be an eligible reference for lex-parse, we need to check that $F_k[f_{k-1}..]$ is the previous suffix of $F_k[1..]$. However, Property 2 of Lemma 2 states that there is exactly one other occurrence of F_{k-2} in F_k, starting at $f_{k-2}+1$. The proof of the following lemma shows that the suffix starting with that occurrence is lexicographically larger than F_k, and thus indeed the first phrase has length f_{k-2}, and the second phrase starting with that occurrence can make use of F_k as a reference for a phrase that just ends before the two explicit phrases at the end.

▶ **Lemma 20.** *Assume that $k \geq 6$ is even and $a \prec b$ (Case (c) of Thm. 19). Then the lex-parse of F_k is F_{k-2}, $F_k[f_{k-2}+1..f_k-2]$, b, a.*

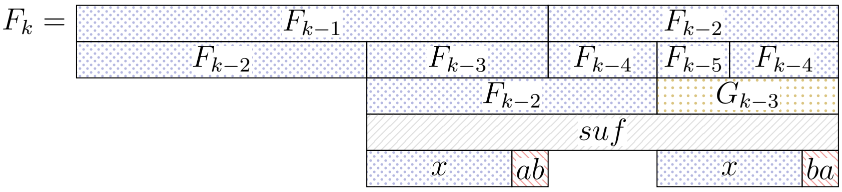


Figure 3 Illustration of the proof of Lemma 20 for k even. If k is odd, the blocks ab and ba are swapped (this gives the setting Lemma 24).

Proof. F_k can be represented as

$$F_k = F_{k-1} \cdot F_{k-2} = F_{k-2} \cdot F_{k-3} \cdot F_{k-4} \cdot F_{k-5} \cdot F_{k-4}.$$

Let $\mathit{suf} = F_{k-2} \cdot G_{k-3}$ (a suffix of F_k) and x be the longest common prefix of F_{k-3} and G_{k-3}. Then $F_k = F_{k-2} \cdot x \cdot ab \cdot F_{k-4} \cdot F_{k-5} \cdot F_{k-4}$ and $\mathit{suf} = F_{k-2} \cdot x \cdot ba$ holds since k is even and $k-3$ is odd. See Fig. 3 for a sketch. This implies that the three suffixes that have F_{k-2} as a prefix satisfy $F_{k-2} \prec F_k \prec \mathit{suf}$. By combining with Property 2 of Lemma 2, the first phrase is F_{k-2} and the second phrase is $F_{k-2} \cdot x$ (which is the longest common prefix of F_k and suf). The third phrase is b which is an explicit phrase of character b since the suffix ba is the lexicographically smallest suffix that has b as a prefix. Finally, the last phrase is an explicit phrase a. ◀

5.2 Lex-parse with logarithmic number of phrases

Next, we discuss the cases that have a logarithmic number of phrases. For any odd $k \geq 7$ and even i satisfying $4 \leq i \leq k-3$, let

$$\mathit{suf}_i = (F_i \cdot F_{i-2} \cdot F_{i-3} \cdots F_4) \cdot ab = F_{i+1},$$
$$\mathit{suf}_i^+ = F_i \cdot \mathit{suf}_i = G_{i+2}.$$

From the definitions, the following properties hold.

▶ **Lemma 21.** *For odd $k \geq 7$, suf_i and suf_i^+ are suffixes of F_k, for every even i with $4 \leq i \leq k-3$. In particular, we have*
(a) *$\mathit{suf}_i = F_{i+1}$ is a prefix of $\mathit{suf}_i^+ = G_{i+2} = F_iF_{i+1}$, and*
(b) *$\mathit{suf}_i^+ = F_iF_{i+1} = F_iF_{i-1}F_{i-2}F_{i-1} = F_{i+1}G_i = \mathit{suf}_i \cdot \mathit{suf}_{i-2}^+$.*

Proof. By definitions, $F_k = F_{k-2}F_{k-3}F_{k-2} = F_{k-2}F_{k-3}\mathit{suf}_{k-3} = F_{k-2}\mathit{suf}_{k-2}^+$. Since suf_{i-2} is a suffix of suf_i, the claim holds for suf_i for every i. Writing $\mathit{suf}_i = F_{i-1} \cdot F_{i-2} \cdot (F_{i-2} \cdot F_{i-3} \cdots F_4) \cdot ab$, we can see that suf_{i-2}^+ is a suffix of suf_i. ◀

We use these suffixes to characterize the lex-parse. The following lemma shows that suf_i is the previous suffix of suf_i^+ w.r.t. F_k for every i, where $f_k - |\mathit{suf}_i^+| + 1$ and $f_k - |\mathit{suf}_i| + 1] - 1$ are the starting positions of suf_i^+ and suf_i in F_k, respectively.

▶ **Lemma 22.** *Assume that $k \geq 7$ is odd and $a \prec b$. The previous suffix of suf_i^+ w.r.t. F_k is suf_i for every even i satisfying $4 \leq i \leq k-3$.*

Proof. Since $\mathit{suf}_i^+ = F_i \cdot F_{i-2} \cdots F_4 \cdot ab \cdot \alpha$ for some string α, suf_i is a prefix of suf_i^+. Thus, $\mathit{suf}_i \prec \mathit{suf}_i^+$. We prove that there is no suffix x of F_k with $\mathit{suf}_i \prec x \prec \mathit{suf}_i^+$ by induction on i.

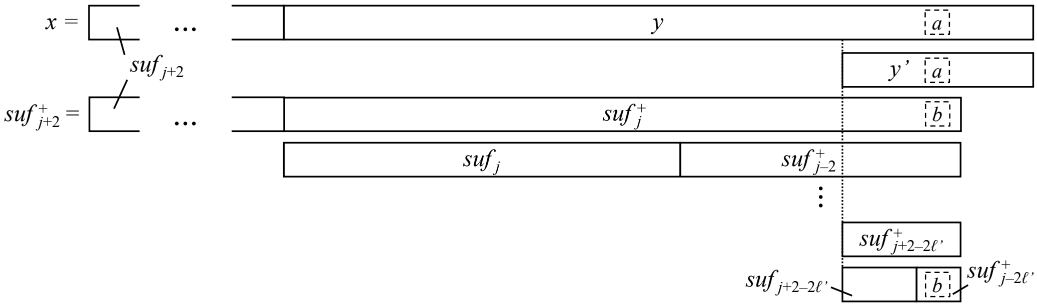


Figure 4 Illustration of the proof of Lemma 22.

Let $i = 4$ for the base case. Assume on the contrary that there exists a suffix x of F_k such that $suf_4 \prec x \prec suf_4^+$. Since $suf_4 = F_5 = F_4 \cdot ab = aba \cdot ab$ and $suf_4^+ = F_4 \cdot F_5 = F_4 \cdot abaab$, x can be written as $x = F_4 \cdot abaaa \cdot y$ for some string y. However, aaa is not a substring of F_k according to Property 5 of Lemma 2, so x cannot exist. Thus, the statement holds for the base case.

Assume that the statement holds for all even $i \leq j$ for some even $j \geq 4$. Suppose on the contrary that there exists a suffix x of F_k such that $suf_{j+2} \prec x \prec suf_{j+2}^+$. Since suf_{j+2} is a prefix of suf_{j+2}^+, we can represent x as $x = suf_{j+2} \cdot y$ for some string y. There are two cases w.r.t. the length of x. (1) Assume that $|suf_{j+2}| < |x| < |suf_{j+2}^+|$. Because of the assumption and the fact that F_{j+2}^2 is a prefix of suf_{j+2}^+ from Lemma 21(a), F_{j+2} occurs as a substring that is neither prefix nor a suffix of F_{j+2}^2. However, F_{j+2}^2 cannot have such occurrence of F_{j+2} since F_{j+2} is primitive (from Property 6 of Lemma 2), a contradiction (from Lemma 1). (2) Assume that $|suf_{j+2}^+| < |x|$. We now use that $suf_i^+ = suf_i \cdot suf_{i-2}^+$ holds from Lemma 21(b). By the assumption, y and suf_j^+ mismatch with a and b, respectively (since x and suf_{j+2}^+ have suf_{j+2} as a prefix). From Lemma 21, suf_{j+2}^+ can be represented as

$$suf_{j+2}^+ = suf_{j+2} \cdot suf_j \cdots suf_{j+2-2\ell} \cdot suf_{j-2\ell}^+ \tag{1}$$

for some integer $\ell \geq 0$. Let ℓ' be the largest integer ℓ such that the mismatch position is in the factor $suf_{i-2\ell}^+$ of the suf_{j+2}^+-factorization in Eq. 1, and y' be the suffix of y that has the factor $suf_{j+2-2\ell'}$ of the factorization in Eq. 1 as a prefix (i.e., $y' = y[|suf_j \cdots suf_{j+2-2(\ell'-1)}| + 1..]$). Since $suf_{j+2-2\ell'}$ is a prefix y', and y' and $suf_{j-2\ell'}^+$ mismatch at the same position, then $suf_{j+2-2\ell'} \prec y' \prec suf_{j+2-2\ell'}^+$ holds. This fact contradicts the induction hypothesis (see also Fig. 4). ◀

Now we can show the following main lemma from the above lemmas.

▶ **Lemma 23.** *Assume that $k \geq 7$ is odd and $a \prec b$. Then the lex-parse of F_k is*

$$F_k[1..f_{k-1} - 2], baF_{k-4}, F_{k-4}, F_{k-6}, \ldots, F_5, a, a, b.$$

Proof. Let x be the longest common prefix of F_{k-3} and G_{k-3}. Due to Property 2 of Lemma 2, there are three suffixes $F_k = F_{k-2} \cdot x \cdot ba \cdot F_{k-4} \cdot F_{k-5} \cdot F_{k-4}$, $suf = F_{k-2} \cdot x \cdot ab$, and F_{k-2} that have F_{k-2} as a prefix. This implies that $F_{k-2} \prec suf \prec F_k$. Thus, the first phrase is $F_{k-2} \cdot x$. Then the remaining suffix is $ba \cdot suf_{k-3} = ba \cdot F_{k-2}$. We show the second phrase is $ba \cdot F_{k-4}$ by proving that the previous suffix of $ba \cdot F_{k-2}$ w.r.t. F_k is $ba \cdot F_{k-4}$. It is clear that $ba \cdot F_{k-4} \prec ba \cdot F_{k-2}$ since $ba \cdot F_{k-4}$ is a prefix of $ba \cdot F_{k-2}$. Suppose on the contrary that there exists a suffix y of F_k that satisfies $ba \cdot F_{k-4} \prec y \prec ba \cdot F_{k-2}$. From the assumption, $ba \cdot F_{k-4}$ is a prefix of y. We can observe that there are three occurrences of $ba \cdot F_{k-4}$ in

F_k from Lemma 3 (i.e., the third, the fourth, and the sixth occurrence of F_{k-4} in F_k). This implies that suffix $ba \cdot F_{k-4} \cdot G_{k-1}$ (regarding the third occurrence of F_{k-4}) of F_k is the only candidate of y. However,

$$y = ba \cdot F_{k-4} \cdot G_{k-1} = ba \cdot F_{k-4} \cdot F_{k-3} \cdot F_{k-2} \succ ba \cdot F_{k-4} \cdot G_{k-3} = ba \cdot F_{k-2}$$

holds. Thus, the previous suffix of $ba \cdot F_{k-2}$ w.r.t. F_k is $ba \cdot F_{k-4}$, and the second phrase is $ba \cdot F_{k-4}$. Then, the remaining suffix is suf^{+}_{k-5}. From Lemma 22, the next phrase is $suf_{k-5} = F_{k-4}$ and the remaining suffix is suf^{+}_{k-7}. This continues until the remaining suffix is aab. It is easy to see that the last three phrases are a, a, b. ◀

We can also prove the following lemmas similarly.

▶ **Lemma 24.** *Assume that $k \geq 7$ is odd and $b \prec a$. Then the lex-parse of F_k is*

$$F_{k-2},\ F_k[f_{k-2}+1..f_k-2],\ a,\ b.$$

▶ **Lemma 25.** *Assume that $k \geq 6$ is even and $b \prec a$. Then the lex-parse of F_k is*

$$F_k[1..f_{k-1}-2], abF_{k-4}, F_{k-4}, F_{k-6}, \ldots, F_6, b, a.$$

Overall, Theorem 19 holds.

6 Conclusion

In this paper, we considered the compression sensitivity of lex-parse for two operations: single character edit and modification of the alphabet ordering, and gave $\Theta(\log n)$ bounds for both operations. A further work in this line of research on the alphabet orderings is the problem of computing optimal alphabet orderings for the lex-parse. The problems for the RLBWT and the Lyndon factorization are known to be NP-hard [2, 13].

References

1 Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga. Sensitivity of string compressors and repetitiveness measures. *Information and Computation*, 291:104999, 2023. doi:10.1016/j.ic.2022.104999.

2 Jason W. Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of BWT-runs minimization via alphabet reordering. 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 15:1–15:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.15.

3 Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.

4 Michael Burrows and David Wheeler. A block-sorting lossless data compression algorithm. Technical report, DIGITAL SRC RESEARCH REPORT, 1994.

5 Davide Cenzato, Veronica Guerrini, Zsuzsanna Lipták, and Giovanna Rosone. Computing the optimal BWT of very large string collections. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2023, Snowbird, UT, USA, March 21-24, 2023*, pages 71–80. IEEE, 2023. doi:10.1109/DCC55655.2023.00015.

6 K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus. IV. The quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958.

7 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.

8 Amanda Clare, Jacqueline W. Daykin, Thomas Mills, and Christine Zarges. Evolutionary search techniques for the Lyndon factorization of biosequences. In Manuel López-Ibáñez, Anne Auger, and Thomas Stützle, editors, *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, pages 1543–1550. ACM, 2019. doi:10.1145/3319619.3326872.

9 Maxime Crochemore and Wojciech Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995. doi:10.1007/BF01190846.

10 Maxime Crochemore and Renaud Vérin. Direct construction of compact directed acyclic word graphs. In *Proc. CPM*, volume 1264 of *LNCS*, pages 116–129, 1997. doi:10.1007/3-540-63220-4_55.

11 Hiroto Fujimaru, Yuto Nakashima, and Shunsuke Inenaga. On sensitivity of compact directed acyclic word graphs. In Anna E. Frid and Robert Mercas, editors, *Combinatorics on Words - 14th International Conference, WORDS 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, volume 13899 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2023. doi:10.1007/978-3-031-33180-0_13.

12 Raffaele Giancarlo, Giovanni Manzini, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. A new class of string transformations for compressed text indexing. *Inf. Comput.*, 294:105068, 2023. doi:10.1016/J.IC.2023.105068.

13 Daniel Gibney and Sharma V. Thankachan. Finding an optimal alphabet ordering for Lyndon factorization is hard. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 35:1–35:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.35.

14 Sara Giuliani, Shunsuke Inenaga, Zsuzsanna Lipták, Giuseppe Romana, Marinella Sciortino, and Cristian Urbina. Bit catastrophes for the Burrows–Wheeler transform. In Frank Drewes and Mikhail Volkov, editors, *Developments in Language Theory - 27th International Conference, DLT 2023, Umeå, Sweden, June 12-16, 2023, Proceedings*, volume 13911 of *Lecture Notes in Computer Science*, pages 86–99. Springer, 2023. doi:10.1007/978-3-031-33264-7_8.

15 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.

16 Hiroe Inoue, Yoshiaki Matsuoka, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Factorizing strings into repetitions. *Theory Comput. Syst.*, 66(2):484–501, 2022. doi:10.1007/S00224-022-10070-3.

17 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *STOC 2018*, pages 827–840, 2018.

18 Dominik Köppl and Tomohiro I. Arithmetics on suffix arrays of Fibonacci words. In Florin Manea and Dirk Nowotka, editors, *Combinatorics on Words - 10th International Conference, WORDS 2015, Kiel, Germany, September 14-17, 2015, Proceedings*, volume 9304 of *Lecture Notes in Computer Science*, pages 135–146. Springer, 2015. doi:10.1007/978-3-319-23660-5_12.

19 Guillaume Lagarde and Sylvain Perifel. Lempel-Ziv: A "one-bit catastrophe" but not a tragedy. 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 1478–1495. SIAM, 2018. doi:10.1137/1.9781611975031.97.

20 M Lothaire. *Applied combinatorics on words*, volume 105. Cambridge University Press, 2005.

21 R. C. Lyndon. On Burnside's problem. *Transactions of the American Mathematical Society*, 77:202–215, 1954.

22 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 Thomas Bäck, Mike Preuss, André H. Deutz, Hao Wang, Carola Doerr, Michael T. M. Emmerich, and Heike Trautmann, editors, *Parallel Problem Solving from Nature – PPSN XVI – 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*, volume 12269 of *Lecture Notes in Computer Science*, pages 390–403. Springer, 2020. doi:10.1007/978-3-030-58112-1_27.

23 Lily Major, Amanda Clare, Jacqueline W. Daykin, Benjamin Mora, and Christine Zarges. Heuristics for the run-length encoded Burrows–Wheeler transform alphabet ordering problem. *CoRR*, abs/2401.16435, 2024. doi:10.48550/arXiv.2401.16435.

24 Guy Melançon. Lyndon factorization of sturmian words. *Discrete Mathematics*, 210(1):137–149, 2000. doi:10.1016/S0012-365X(99)00123-5.

25 Gonzalo Navarro. Indexing highly repetitive string collections. *CoRR*, abs/2004.02781, 2020. arXiv:2004.02781.

26 Gonzalo Navarro. Indexing highly repetitive string collections, part I: repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.

27 Gonzalo Navarro, Carlos Ochoa, and Nicola Prezza. On the approximation ratio of ordered parsings. *IEEE Trans. Inf. Theory*, 67(2):1008–1026, 2021. doi:10.1109/TIT.2020.3042746.

28 James A. Storer and Thomas G. Szymanski. The macro model for data compression (extended abstract). In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing, May 1-3, 1978, San Diego, California, USA*, pages 30–39. ACM, 1978. doi:10.1145/800133.804329.

29 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

30 Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, 24(5):530–536, 1978.

Point-To-Set Principle and Constructive Dimension Faithfulness

Satyadev Nandakumar
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Subin Pulari
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Akhil S
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Abstract

Hausdorff Φ-dimension is a notion of Hausdorff dimension developed using a restricted class of coverings of a set. We introduce a constructive analogue of Φ-dimension using the notion of constructive Φ-s-supergales. We prove a Point-to-Set Principle for Φ-dimension, through which we get Point-to-Set Principles for Hausdorff dimension, continued-fraction dimension and dimension of Cantor coverings as special cases. We also provide a Kolmogorov complexity characterization of constructive Φ-dimension.

A class of covering sets Φ is said to be "faithful" to Hausdorff dimension if the Φ-dimension and Hausdorff dimension coincide for every set. Similarly, Φ is said to be "faithful" to constructive dimension if the constructive Φ-dimension and constructive dimension coincide for every set. Using the Point-to-Set Principle for Cantor coverings and a new technique for the construction of sequences satisfying a certain Kolmogorov complexity condition, we show that the notions of "faithfulness" of Cantor coverings at the Hausdorff and constructive levels are equivalent.

We adapt the result by Albeverio, Ivanenko, Lebid, and Torbin [1] to derive the necessary and sufficient conditions for the constructive dimension faithfulness of the coverings generated by the Cantor series expansion, based on the terms of the expansion.

2012 ACM Subject Classification Theory of computation → Computability; Mathematics of computing → Information theory

Keywords and phrases Kolmogorov complexity, Constructive dimension, Faithfulness, Point to set principle, Continued fraction dimension, Cantor series expansion

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.76

Related Version *Full Version*: https://arxiv.org/abs/2403.08278

Acknowledgements We gratefully acknowledge the anonymous reviewers for their invaluable insights and constructive feedback, which has significantly strengthened this manuscript.

1 Introduction

1.1 Faithfulness in dimension

In the study of randomness and information, an important concept is the preservation of randomness across multiple representations of the same object. Martin-Löf randomness, and computable randomness, for example, are preserved among different base-b representations of the same real (see Downey and Hirschfeldt [5], Nies [27], Staiger [31]) and when we convert from the base-b expansion to the continued fraction expansion ([23, 26, 25]).

A quantification of this notion is whether the *rate* of information is preserved across multiple representations. This rate is studied using a constructive analogue of Hausdorff dimension called Constructive dimension [11, 21]. Hitchcock and Mayordomo [8] show that constructive dimension is preserved across base-b representations. However, in a recent work,

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 76; pp. 76:1–76:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Akhil, Nandakumar and Vishnoi [24] show that the rate of information is not preserved across all representations. In particular, they show that constructive dimension is *not* preserved when we convert from base-b representation to continued fraction representation of the same real.

This raises the following question: *Under which settings is the effective rate of information – i.e., constructive dimension – preserved when we change representations of the same real?* Since constructive dimension is a constructive analogue of Hausdorff dimension, this question is a constructive analogue of the concept of "faithfulness" of Hausdorff dimension.

A family of covering sets Φ is "faithful" to Hausdorff dimension if the dimension of every set $\mathcal{F}$ defined using covers constructed using Φ, called the Hausdorff Φ-dimension, coincides with the Hausdorff dimension of $\mathcal{F}$. Faithfulness is well-studied as determining the Hausdorff dimension of a set is often a difficult problem, and faithful coverings help simplify the calculation. This notion is introduced in a work of Besicovitch [3], which shows that the class of dyadic intervals is faithful for Hausdorff dimension. Rogers and Taylor [29] further develop the idea to show that all covering families generated by comparable net measures are faithful for Hausdorff dimension. This implies that the class of covers generated by the base b expansion of reals for any $b \in \mathbb{N} \setminus \{1\}$ is faithful for Hausdorff dimension. However, not all coverings are faithful for Hausdorff dimension. A natural example is the continued fraction representation, which is not faithful for Hausdorff dimension [28]. Faithfulness of Hausdorff dimension has then been studied in various settings [1, 2, 9, 28].

1.2 Constructive Dimension Faithfulness

In this work, we introduce a constructive analogue of Hausdorff Φ-dimension which we call constructive Φ-dimension. A family of covering sets Φ is "faithful" to constructive dimension if the constructive Φ-dimension of every set $\mathcal{F}$ coincides with the constructive dimension of $\mathcal{F}$. Mayordomo and Hitchcock [8] show that all base-b representations of reals, which are faithful for Hausdorff dimension, are also faithful for constructive dimension. On the other hand, Nandakumar, Akhil, and Vishnoi's work shows that the continued fraction expansion, which is not faithful for Hausdorff dimension is also not faithful for constructive dimension [24]. This raises the natural question: *Are faithfulness with respect to Hausdorff dimension and faithfulness with respect to constructive dimension equivalent notions?* A positive answer to this question implies that Hausdorff dimension faithfulness, a geometric notion, can be studied using the tools from information theory. Conversely, the faithfulness results of Hausdorff dimension can help us understand the settings under which constructive dimension is invariant for *every* individual real.

In this work, we show that for the most inclusive generalization of base-b expansions under which faithfulness has been studied classically, namely, for classes of coverings generated by the Cantor series expansions, the notions of Hausdorff faithfulness and constructive faithfulness are indeed equivalent. The Cantor series expansion, introduced by Georg Cantor [4], uses a sequence of natural numbers $Q = \{n_k\}_{k\in\mathbb{N}}$ as the terms of representation. Whereas base-b representation use exponentials with respect to a fixed b, $\{b^n\}_{n\in\mathbb{N}}$, the Cantor series representation $Q = \{n_k\}_{k\in\mathbb{N}}$ uses factorials $\{n_1 \dots n_k\}_{k\in\mathbb{N}}$ as the basis for representation. This class is of additional interest as there are Cantor expansions that are faithful as well as non faithful for Hausdorff dimension, depending on the Cantor series representation $\{n_k\}_{k\in\mathbb{N}}$ in consideration [1]. To establish our result, we use a Φ-dimensional analogue of the Point-to-Set Principle.

1.3 Point-to-Set Principle and Faithfulness

The Point-to-Set principle introduced by J. Lutz and N. Lutz [13] relates the Hausdorff dimension of a set of n-dimensional reals with the constructive dimensions of points in the set, relative to a minimizing oracle. This theorem has been instrumental in answering open questions in classical fractal geometry using the theory of computing (See [14, 20, 19, 18, 13]). Mayordomo, Lutz, and Lutz [15] extend this work to arbitrary separable metric spaces.

In this work, we first prove the Point-to-Set principle for Φ-dimension (Theorem 27) and show that this generalizes the original Point-to-Set principle. We then develop a combinatorial construction of sequences having Kolmogorov complexities that grow at the same rate as a given sequence relative to any given oracle (Theorem 36). This new combinatorial construction may be of independent interest in the study of randomness. Using these new tools, we show that under the setting of covers generated by Cantor series expansions, the notions of constructive faithfulness and Hausdorff dimension faithfulness are equivalent (Theorem 51). We then adapt the result by Albeverio, Ivanenko, Lebid, and Torbin [1] to derive a loglimit condition for the constructive dimension faithfulness of the coverings generated by the Cantor series expansions (Theorem 53).

Our main results include the following.

1. We introduce the notion of constructive Φ-dimension using that subsumes base-b, continued fraction, and Cantor covering dimension. We also give an equivalent Kolmogorov Complexity characterization of constructive Φ-dimension. We prove a Point-to-Set principle for Φ-dimension. This generalizes the original Point-to-Set Principle and yields new Point-to-Set principles for the dimensions of continued fractions and Cantor series representations.
2. Using the point-to-set principle, we characterize constructive faithfulness for Cantor series expansions using a log limit condition of the terms appearing in the series. This generalizes the invariance result of constructive dimension under base b representations to all Cantor series expansions that obey this log limit condition. Moreover, it implies that for any Cantor series expansion that does not obey the log limit condition, there are sequences whose Cantor series dimension is different from its constructive dimension.

The recent works of J. Lutz, N. Lutz, Stull, Mayordomo and others study the "point-to-set principle" of how constructive Hausdorff dimension of points may be used to compute the classical Hausdorff dimension of arbitrary sets. In addition to the generalization of this point-to-set principle to Φ-systems, our final result may be viewed as a new *point-to-set phenomenon* for the notion of "faithfulness": here, equality of the constructive Cantor series dimensions and constructive dimensions of *every point* yield equality for the classical Cantor series and Hausdorff dimensions of *every set*, and conversely.

2 Preliminaries

2.1 Notation

We use Σ to denote the binary alphabet $\{0,1\}$, Σ^* represents the set of finite binary strings, and Σ^∞ represents the set of infinite binary sequences. We use $|x|$ to denote the length of a finite string $x \in \Sigma^*$. For an infinite sequence $X = X_0X_1X_2\ldots$, we use $X \upharpoonright n$ to denote the finite string consisting of the first n symbols of X. When $n \geq m$ we also use the notation $X[m,n]$ to denote the substring $X_mX_{m+1}\ldots X_n$ of $X \in \Sigma^\infty$. We call two sets U and V to be incomparable if $U \not\subseteq V$ and $V \not\subseteq U$. For a set $U \subseteq \mathbb{R}$, we denote $|U|$ to denote the diameter of U, that is $|U| = \sup_{x,y\in U} d(x,y)$, where d is the Euclidean metric.

We use $\emptyset$ to denote the empty set, and we assume $|\emptyset| = 0$. For any finite collection of sets $\{U\}$, we use $\#(U)$ to denote the number of elements in U. Given infinite sequences $X_1, \dots X_n$, we define the *interleaved sequence* $X_1 \oplus X_2 \oplus \dots X_n$ to be the interleaved sequence $X = X_1[0]X_2[0] \dots X_n[0]X_1[1] \dots X_n[1] \dots$. For some fixed $n \in \mathbb{N}$, we use $\mathbb{X} \subseteq \mathbb{R}^n$ to denote the metric space under consideration. We call a set of strings $\mathcal{P} \subset \Sigma^*$ to be prefix free if there are no two strings $\sigma, \tau \in \mathcal{P}$ such that σ is a proper prefix of τ. Given $n \in \mathbb{N}$ we use $[n]$ to denote $\{0, 1, \dots n-1\}$. Kolmogorov Complexity represents the amount of information contained in a finite string. For more details on Kolmogorov Complexity, see [5, 10, 27, 30].

▶ **Definition 1.** *The Kolmogorov complexity of $\sigma \in \Sigma^*$ is defined as $K(\sigma) = \min\limits_{\pi \in \Sigma^*} \{|\pi| \mid U(\pi) = \sigma\}$, where U is a fixed universal prefix free Turing machine.*

2.2 Hausdorff Dimension

The following definitions are originally given by Hausdorff [7]. We take the definitions from Falconer [6].

▶ **Definition 2** (Hausdorff [7])**.** *Given a set $\mathcal{F} \subseteq \mathbb{X}$, a collection of sets $\{U_i\}_{i \in \mathbb{N}}$ where for each $i \in \mathbb{N}$, $U_i \subseteq \mathbb{X}$ is called a δ-*cover of $\mathcal{F}$ *if for all $i \in \mathbb{N}$, $|U_i| \leq \delta$ and $\mathcal{F} \subseteq \bigcup_{i \in \mathbb{N}} U_i$.*

▶ **Definition 3** (Hausdorff [7])**.** *Given an $\mathcal{F} \subseteq \mathbb{X}$, for any $s > 0$, define*

$$\mathcal{H}^s_\delta(\mathcal{F}) = \inf \left\{ \sum_i |U_i|^s : \{U_i\}_{i \in \mathbb{N}} \textit{ is a } \delta\textit{-cover of } \mathcal{F} \right\}.$$

As δ decreases, the set of admissible δ covers decreases. Hence $\mathcal{H}^s_\delta(\mathcal{F})$ increases.

▶ **Definition 4** (Hausdorff [7])**.** *For $s \in (0, \infty)$, the* s-dimensional Hausdorff outer measure *of $\mathcal{F}$ is defined as:*

$$\mathcal{H}^s(\mathcal{F}) = \lim_{\delta \to 0} \mathcal{H}^s_\delta(\mathcal{F}).$$

Observe that for any $t > s$, if $\mathcal{H}^s(\mathcal{F}) < \infty$, then $\mathcal{H}^t(\mathcal{F}) = 0$ (see Section 2.2 in [6]). Finally, we have the following definition of Hausdorff dimension.

▶ **Definition 5** (Hausdorff [7])**.** *For any $\mathcal{F} \subset \mathbb{X}$, the* Hausdorff dimension *of $\mathcal{F}$ is defined as:*

$$\dim(\mathcal{F}) = \inf\{s \geq 0 : \mathcal{H}^s(\mathcal{F}) = 0\}.$$

2.3 Constructive dimension

Lutz [11] defines the notion of effective (equivalently, constructive) dimension of an infinite binary sequence using the notion of lower semicomputable s-gales.

▶ **Definition 6** (Lutz [11])**.** *For $s \in [0, \infty)$, a binary s-gale is a function $d : \Sigma^* \to [0, \infty)$ such that $d(\lambda) < \infty$ and for all $w \in \Sigma^*$, $d(w) = 2^s . \sum_{i \in \{0,1\}} d(wi)$.*

The *success set* of d is $S^\infty(d) = \left\{ X \in \Sigma^\infty \mid \limsup\limits_{n \to \infty} d(X \upharpoonright n) = \infty \right\}$.

For $\mathcal{F} \subseteq [0, 1]$, $\mathcal{G}(\mathcal{F})$ denotes the set of all $s \in [0, \infty)$ such that there exists a lower semicomputable (Definition 24) binary s-gale d with $\mathcal{F} \subseteq S^\infty(d)$.

▶ **Definition 7** (Lutz [11]). *The* constructive dimension *or* effective Hausdorff dimension *of* $\mathcal{F} \subseteq [0,1]$ *is*

$$\operatorname{cdim}(\mathcal{F}) = \inf \mathcal{G}(\mathcal{F}).$$

The constructive dimension of a sequence $X \in \Sigma^\infty$ *is* $\operatorname{cdim}(X) = \operatorname{cdim}(\{X\})$.

Mayordomo [21] extends the result by Lutz [12] to give the following Kolmogorov complexity characterization of constructive dimension of infinite binary sequences.

▶ **Theorem 8** (Lutz [12], Mayordomo [21]). *For any* $X \in \Sigma^\infty$*,*

$$\operatorname{cdim}(X) = \liminf_{n\to\infty} \frac{K(X \upharpoonright n)}{n}.$$

Mayordomo [22] later gave the following Kolmogorov complexity characterization of constructive dimension of points in $\mathbb{R}^n$.

▶ **Definition 9** (Mayordomo [22]). *For any* $x \in \mathbb{R}^n$*,*

$$\operatorname{cdim}(x) = \liminf_{r\to\infty} \frac{K_r(x)}{r}.$$

where $K_r(x) = \min\limits_{q\in\mathbb{Q}^n} \{K(q) : |x-q| < 2^{-r}\}$.

Constructive dimension also works in the Euclidean space. For a real $x = (x_1, ...x_n) \in \mathbb{R}^n$, let binary expansions of the fractional parts of each of the coordinates of x be $S_1 \in \Sigma^\infty, \ldots, S_n \in \Sigma^\infty$ respectively. Then $\operatorname{cdim}(x) = n \cdot \operatorname{cdim}(X)$ where X is the interleaved sequence $X = S_1 \oplus S_2 \cdots \oplus S_n$ [16].

We now state some useful properties of constructive dimension. Lutz [11] shows that the constructive dimension of a set is always greater than or equal to its Hausdorff dimension.

▶ **Lemma 10** (Lutz [11]). *For any* $\mathcal{F} \subseteq \mathbb{X}$, $\dim(\mathcal{F}) \leq \operatorname{cdim}(\mathcal{F})$.

Further, Lutz[11] also shows that the constructive dimension of a set is the supremum of the constructive dimensions of points in the set.

▶ **Lemma 11** (Lutz [11]). *For any* $\mathcal{F} \subseteq \mathbb{X}$, $\operatorname{cdim}(\mathcal{F}) = \sup\limits_{x\in\mathcal{F}} \operatorname{cdim}(x)$.

3 Hausdorff Φ-dimension and Effective Φ-dimension

Hausdorff dimension is defined using the notion of s-dimensional outer measures, where a cover is taken as the of union of a collection of covering sets $\{U_i\}_{i\in\mathbb{N}}$. Here a covering set U_i can be any arbitrary subset of the space (see Section 2.2). We define the general notion of Hausdorff Φ-dimension by restricting the class of admissible covers to Φ-covers, which are the union of sets from a family of covering sets Φ.

3.1 Family of covering sets

In this work, we consider a *family of covering sets* which satisfy the properties given below.

▶ **Definition 12** (Family of covering sets Φ). *We consider the space* $\mathbb{X} \subseteq \mathbb{R}^\eta$ *where* $\eta \in \mathbb{N}$. *A countable family of sets* $\Phi = \bigcup_{n\in\mathbb{Z}} \{U_i^n\}_{i\in\mathbb{N}}$*, where for each* $i \in \mathbb{N}, n \in \mathbb{Z}$, $U_i^n \subseteq \mathbb{X}$*, is called a family of covering sets if it satisfies the following properties:*

- *Increasing Monotonicity: For every $n \in \mathbb{Z}$, $U \in \{U_i^n\}_{i\in\mathbb{N}}$ and $m \leq n$, there is a unique $V \in \{U_i^m\}_{i\in\mathbb{N}}$ such that $U \subseteq V$.*
- *Fineness : Given any $\epsilon > 0$, and $x \in \mathbb{X}$, there exists a $U \in \Phi$ such that $|U| < \epsilon$ and $x \in U$.*

Note that the number of sets $\{U_i^n\}$ in some level $n \in \mathbb{Z}$ can also be finite and bounded by m. The definition still holds because in this case we take $U_j^n = \emptyset$ for $j > m$. Note that from Increasing monotonicity property, it follows that all elements $\{U_i^n\}$ in a particular level $n \in \mathbb{Z}$ are incomparable.

We now define the notion of a Φ-cover of a set.

▶ **Definition 13** (Φ-cover). *Let $\Phi = \bigcup_{n\in\mathbb{Z}}\{U_i^n\}_{i\in\mathbb{N}}$ be a family of covering sets. A Φ - cover of a set $\mathcal{F} \subseteq \mathbb{X}$ is collection of sets $\{V_j\}_{j\in\mathbb{N}} \subseteq \Phi$ such that $\{V_j\}_{j\in\mathbb{N}}$ covers $\mathcal{F}$, that is $\mathcal{F} \subseteq \bigcup\limits_{j\in\mathbb{N}} V_j$*

Note: Mayordomo [22] gives a definition of *Nice covers* of a metric space. They then give the definition of constructive dimension on a metric space with a nice cover. We note here that the notion of Family of covering sets is incomparable with the notion of Nice covers. Our definition does not require the c-cover property and the Decreasing monotonicity property of nice covers. Therefore, our notion includes the setting of continued fraction dimension, which is not captured by Nice covers. Also, the Fineness property required in our definition is not there in the definition of nice covers. The notion of Increasing monotonicity is present in both settings.

3.2 Hausdorff Φ-dimension

Recall from Definition 13 that a Φ-cover of $\mathcal{F}$ is a collection of sets from Φ that covers $\mathcal{F}$. We call this as a δ-cover if the diameter of elements in the cover are less than δ.

▶ **Definition 14.** *Let Φ be a family of covering sets defined over $\mathbb{X}$. Given a set $\mathcal{F} \subseteq \mathbb{X}$, a Φ-cover $\{U_i\}_{i\in\mathbb{N}}$ of $\mathcal{F}$ is called a* δ-cover of $\mathcal{F}$ using Φ *if for all $i \in \mathbb{N}$, $|U_i| \leq \delta$.*

▶ **Definition 15.** *Given an $\mathcal{F} \subseteq \mathbb{X}$, for any $s > 0$, we define*

$$\mathcal{H}_\delta^s(\mathcal{F}, \Phi) = \inf\left\{\sum_i |U_i|^s : \{U_i\}_{i\in\mathbb{N}} \textit{ is a } \delta\textit{-cover of } \mathcal{F} \textit{ using } \Phi\right\}.$$

From the fineness property given in Definition 12, it follows that for any $\mathcal{F} \subseteq \mathbb{X}$, and $\delta > 0$, δ-covers of $\mathcal{F}$ using Φ always exist.

As δ decreases, the set of admissible δ-covers using Φ decreases. Hence $\mathcal{H}_\delta^s(\mathcal{F}, \Phi)$ increases.

▶ **Definition 16.** *For $s \in (0, \infty)$, define the* s-dimensional Φ outer measure *of $\mathcal{F}$ as:*

$$\mathcal{H}^s(\mathcal{F}, \Phi) = \lim_{\delta\to 0} \mathcal{H}_\delta^s(\mathcal{F}, \Phi).$$

Observe that as with the case of classical Hausdorff dimension, for any $t > s$, if $\mathcal{H}^s(\mathcal{F}, \Phi) < \infty$, then $\mathcal{H}^t(\mathcal{F}, \Phi) = 0$ (see Section 2.2 in [6]).

Finally, we have the following definition of Hausdorff Φ-dimension.

▶ **Definition 17.** *For any $\mathcal{F} \subset \mathbb{X}$, the* Hausdorff Φ-dimension *of $\mathcal{F}$ is defined as:*

$$\dim_\Phi(\mathcal{F}) = \inf\{s \geq 0 : \mathcal{H}^s(\mathcal{F}, \Phi) = 0\}.$$

3.3 Effective Φ-dimension

We first formulate the notion of a Φ-s-supergale. A Φ-s-supergale can be seen as a gambling strategy where the bets are placed on the covering sets from Φ. The definitions in this subsection are adaptations from Mayordomo [22].

▶ **Definition 18** (Mayordomo [22]). *Let $\Phi = \bigcup_{n\in\mathbb{Z}}\{U_i^n\}_{i\in\mathbb{N}}$ be a family of covering sets from Definition 12. For $s \in [0,\infty)$, a Φ-s-supergale is a function $d : \Phi \to [0,\infty)$ such that:*

- $\sum\limits_{U\in\{U_i^0\}_{i\in\mathbb{N}}} d(U)|U|^s < \infty$ *and*
- *For all $n \in \mathbb{N}$ and all $U \in \{U_i^n\}_{i\in\mathbb{N}}$, the following condition holds:*

$$d(U).|U|^s \geq \sum_{V\in\{U_i^{n+1}\}_{i\in\mathbb{N}}, V\subseteq U} d(V)|V|^s \ .$$

The following is the generalization of Kraft inequality for s-supergales from Mayordomo [22].

▶ **Lemma 19** (Generalisation of Kraft inequality [22]). *Let d be a Φ-s-supergale. Then for every $\mathcal{E} \subseteq \Phi$ such that the sets in $\mathcal{E}$ are incomparable, we have that*

$$\sum_{V\in\mathcal{E}} d(V)|V|^s \leq \sum_{U\in\{U_i^0\}_{i\in\mathbb{N}}} d(U)|U|^s.$$

▶ **Definition 20** (Mayordomo [22]). *Given $x \in \mathbb{X}$, a Φ-representation of x is a sequence $(U_n)_{n\in\mathbb{Z}}$ such that for each $n \in \mathbb{Z}$, $U_n \in \{U_i^n\}_{i\in\mathbb{N}}$ and $x \in \cap_n U_n$.*

Note that the same x can have multiple Φ-representations. Given $x \in \mathbb{X}$, let $\mathcal{R}(x)$ be the set of Φ-representations of x.

▶ **Definition 21** (Mayordomo [22]). *A Φ-s-supergale d succeeds on $x \in \mathbb{X}$ if there is a $(U_n)_{n\in\mathbb{Z}} \in \mathcal{R}(x)$ such that $\limsup\limits_{n\to\infty} d(U_n) = \infty$.*

Equivalently, a Φ-s-supergale d succeeds on a point $x \in \mathbb{X}$ iff for every $k \in \mathbb{N}$, there exists a $U \in \Phi$ such that $x \in U$ and $d(U) > 2^k$.

▶ **Definition 22.** *The* success set *of d is $S^\infty(d) = \{x \in \mathbb{X} \mid d \text{ succeeds on } x\}$.*

To define constructive Φ-dimension, we require some additional computability restrictions over Φ. It is an adaptation of the definition from [22].

▶ **Definition 23** (Family of computable covering sets Φ). *We consider the space $\mathbb{X} \subseteq \mathbb{R}^\eta$ where $\eta \in \mathbb{N}$ and a family of covering sets $\Phi = \bigcup_{n\in\mathbb{Z}}\{U_i^n\}_{i\in\mathbb{N}}$ from Definition 12. We call Φ to be a family of computable covering sets if it satisfies the following additional properties:*

- *Computable diameter: For every $n \in \mathbb{Z}$ and $i \in \mathbb{N}$, $|U_i^n|$ is computable.*
- *Computable subsets: For every $n \in \mathbb{Z}$, and $i \in \mathbb{N}$, the set $\{j \in \mathbb{N} : U_j^{n+1} \subseteq U_i^n\}$ is uniformly computable.*

In definition 23, when we say $|U_i^n|$ is computable, we mean that there is a turing machine that on input n, i, r outputs a $q \in Q$ such that $||U_i^n| - q| < 2^r$. The set $\{j \in \mathbb{N} : U_j^{n+1} \subseteq U_i^n\}$ is uniformly computable if there is a turing machine which on input i, j, n decides if $U_j^{n+1} \subseteq U_i^n$.

We use constructive Φ-s-gales to define the notion of constructive Φ-dimension. For a Φ-s-gale d to be constructive, we require the gale function d to be lower semicomputable. Note that a lower semicomputable supergale actually takes as input (i, n) where $i \in \mathbb{N}, n \in \mathbb{Z}$ to place bets on U_i^n. We omit this technicality in this paper and keep the domain of the gale as Φ for the sake of simplicity.

▶ **Definition 24.** *A function* $d : \Phi \longrightarrow [0, \infty)$ *is called* lower semicomputable *if there exists a total computable function* $\hat{d} : \Phi \times \mathbb{N} \longrightarrow \mathbb{Q} \cap [0, \infty)$ *such that the following two conditions hold.*

- ***Monotonicity*** *: For all* $U \in \Phi$ *and for all* $n \in \mathbb{N}$*, we have* $\hat{d}(U, n) \leq \hat{d}(U, n + 1) \leq d(U)$*.*
- ***Convergence*** *: For all* $U \in \Phi$*,* $\lim\limits_{n\to\infty} \hat{d}(U, n) = d(U)$*.*

For $\mathcal{F} \subseteq \mathbb{X}$, let $\mathcal{G}_\Phi(\mathcal{F})$ denote the set of all $s \in [0, \infty)$ such that there exists a lower semicomputable Φ-s-supergale d with $\mathcal{F} \subseteq S^\infty(d)$.

▶ **Definition 25.** *The* constructive Φ-dimension *of* $\mathcal{F} \subseteq \mathbb{X}$ *is*

$$\mathrm{cdim}_\Phi(\mathcal{F}) = \inf \mathcal{G}_\Phi(\mathcal{F}).$$

The constructive Φ dimension *of a point* $x \in \mathbb{X}$ *is defined by* $\mathrm{cdim}_\Phi(\{x\})$*, the constructive* Φ*-dimension of the singleton set containing* x*.*

This definition can easily be relativized with respect to an oracle $A \subseteq \mathbb{N}$ by giving the s-supergale an additional oracle access to set $A \subseteq \mathbb{N}$. We denote this using $\mathrm{cdim}^A_\Phi(\mathcal{F})$.

We now show that the constructive Φ-dimension of a set is the supremum of constructive Φ-dimensions of points in the set. The proof is a straightforward adaptation of proof of Theorem 11 by Lutz [11].

▶ **Theorem 26.** *For any family of computable covering sets* Φ *defined over the space* $\mathbb{X}$*, for any* $\mathcal{F} \subseteq \mathbb{X}$*, we have*

$$\mathrm{cdim}_\Phi(\mathcal{F}) = \sup_{x\in\mathcal{F}} \mathrm{cdim}_\Phi(x).$$

4 Point-to-set principle for Φ-dimension

Let $\Phi = \bigcup_{n\in\mathbb{Z}}\{U^n_i\}_{i\in\mathbb{N}}$ be a family of computable covering sets from Definition 23. In this work, we introduce the Point-to-Set principle for Φ-dimension. We show that the Hausdorff Φ-dimension of any set $\mathcal{F} \subseteq \mathbb{X}$ is equal to the relative constructive Φ-dimensions of elements in the set, relative to a minimizing oracle A.

▶ **Theorem 27.** *For a family of computable covering sets* Φ *over the space* $\mathbb{X}$*, for all* $\mathcal{F} \subseteq \mathbb{X}$*,*

$$\dim_\Phi(\mathcal{F}) = \min_{A\subseteq\mathbb{N}} \sup_{x\in\mathcal{F}} \mathrm{cdim}^A_\Phi(x).$$

4.1 Point to Set Principle for constructive dimension

▶ **Definition 28** (Dyadic Family of covers)**.** *Consider the space* $\mathbb{X} = \mathbb{R}^n$*. The dyadic family of covers is the set of coverings* $\Phi_B = \bigcup_{r\in\mathbb{N}}\{[\frac{m_1}{2^r}, \frac{m_1+1}{2^r}] \times \cdots \times [\frac{m_n}{2^r}, \frac{m_n+1}{2^r}]\}_{m_1,m_2\ldots,m_n\in[2^r]}$*.*

It is straightforward to verify that Φ_B is a family of computable covering sets from Definition 23. Besicovitch [3] gave the following Φ-dimension characterization of Hausdorff dimension.

▶ **Lemma 29** (Besicovitch [3])**.** *For all* $\mathcal{F} \subseteq \mathbb{R}^n$*, we have* $\dim(\mathcal{F}) = \dim_{\Phi_B}(\mathcal{F})$*.*

Similarly, we have the following Φ-dimension characterization of Constructive dimension.

▶ **Lemma 30** (Lutz and Mayordomo [16])**.** *For all* $\mathcal{F} \subseteq \mathbb{R}^n$*, we have* $\mathrm{cdim}(\mathcal{F}) = \mathrm{cdim}_{\Phi_B}(\mathcal{F})$*.*

From Theorem 27 for Φ_B and using Lemma 29 and 30, we have the following point-to-set principle from [13] relating Hausdorff and Constructive dimensions.

▶ **Corollary 31** (J.Lutz and N.Lutz [13]). *For all $\mathcal{F} \subseteq \mathbb{X}$,*

$$\dim(\mathcal{F}) = \min_{A \subseteq \mathbb{N}} \sup_{x \in \mathcal{F}} \operatorname{cdim}^A(x).$$

4.2 Point to Set Principle for Continued Fraction dimension

The sequence $Y = [a_1, a_2, \dots]$ where each $a_i \in \mathbb{N}$ is the continued fraction expansion of the number $y = \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots}}$. Given $u = [a_1, a_2 \dots a_n] \in \mathbb{N}^*$, the cylinder set of u, C_u is defined as $C_u = [[a_1, a_2, \dots a_n], [a_1, a_2, \dots a_n + 1]]$ when n is even and $C_u = [[a_1, a_2, \dots a_n + 1], [a_1, a_2, \dots a_n]]$ when n is odd.

The notion of constructive continued fraction dimension was introduced by Nandakumar and Vishnoi [26] using continued fraction s-gales. Akhil, Nandakumar and Vishnoi [24] showed that this notion is different from that of constructive dimension.

Consider Φ_{CF} to be the set of covers generated by the continued fraction cylinders, that is $\Phi_{CF} = \bigcup_{n \in \mathbb{Z}} \{C_{[a_1, a_2, \dots a_n]}\}_{a_1 \dots a_n \in \mathbb{N}}$. It is routine to verify that this is a family of computable covering sets from Definition 23. From Theorem 27, we therefore have the following point-to-set principle for Continued fraction dimension.

▶ **Corollary 32.** *For all $\mathcal{F} \subseteq \mathbb{X}$, $\dim_{CF}(\mathcal{F}) = \min_{A \subseteq \mathbb{N}} \sup_{x \in \mathcal{F}} \operatorname{cdim}^A_{CF}(x)$.*

4.3 Effective Φ-dimension using Kolmogorov Complexity

We give an equivalent formulation of constructive Φ-dimension of a point using Kolmogorov complexity. For this, we require some additional properties for the space Φ.

▶ **Definition 33** (Family of finitely intersecting computable covering sets Φ). *We consider the space $\mathbb{X} \subseteq \mathbb{R}^\eta$ where $\eta \in \mathbb{N}$ and a family of computable covering sets $\Phi = \bigcup_{n \in \mathbb{Z}} \{U^n_i\}_{i \in \mathbb{N}}$ from Definition 23. We say that Φ is a family of finitely intersecting computable covering sets if it satisfies the following additional properties:*

- *Density of Rational points: For each $U \in \Phi$, there exists a $q \in \mathbb{Q}^n$ such that $q \in U$.*
- *Finite intersection: There exists a constant $c \in \mathbb{N}$ such that for any collection $\{U_i\} \subseteq \Phi$ satisfying*
  (1) *$U_i \not\subseteq U_j$ for all $i \neq j$, and*
  (2) *$\bigcap_i U_i \neq \emptyset$,*
  we have $\#(\{U_i\}) \leq c$.
- *Membership test: There is a computable function that given $i \in \mathbb{N}$ and $n \in \mathbb{Z}$ and a $q \in \mathbb{Q}^n$ checks if $q \in U^n_i$.*

The Finite intersection property states that the cardinality of any collection of incomparable sets from Φ having non empty intersection is bounded by a constant.

Given family of finitely intersecting computable covering sets Φ over a space $\mathbb{X}$, and an $r \in \mathbb{N}$, we define the notion of Kolmogorov Complexity of a point X at precision r with respect to Φ. We denote this using $K_r(X, \Phi)$.

▶ **Definition 34.** *Given an $r \in \mathbb{N}$, and $x \in \mathbb{X}$, define*

$$K_r(x, \Phi) = \min_{U \in \Phi}\{K(U) \mid x \in U \text{ and } |U| < 2^{-r}\}.$$

where for $U \in \Phi$, $K(U)$ is defined as $K(U) = \min\{K(q) \mid q \in U \cap \mathbb{Q}^n\}$.

For any family of finitely intersecting computable covering sets Φ, we have the following Kolmogorov Complexity characterization of constructive Φ-dimension.

▶ **Theorem 35.** *Given a family of finitely intersecting computable covering sets Φ, over a space $\mathbb{X}$. For any $x \in \mathbb{X}$,*

$$\operatorname{cdim}_\Phi(x) = \liminf_{r \to \infty} \frac{K_r(x, \Phi)}{r}$$

5 Kolmogorov Complexity Construction

In this section we give a technical construction which is crucial in proving the results in section 6. Theorem 36 says that given an infinite sequence X and an oracle A, for any oracle B, there exists a sequence Y whose relativized Kolmogorov complexity (of prefixes) with respect to B is similar to the relativized Kolmogorov complexity (of prefixes) of X with respect to A.

▶ **Theorem 36.** *For all $X \in \Sigma^\infty$ and $A \in \Sigma^\infty$, given a $B \in \Sigma^\infty$, there exists a $Y \in \Sigma^\infty$ such that for all $n \in \mathbb{N}$, $|K^A(X \upharpoonright n) - K^B(Y \upharpoonright n)| = o(n)$ and $\operatorname{cdim}^B(Y) = \operatorname{cdim}(Y)$.*

6 Equivalence of Faithfulness of Cantor Coverings at Constructive and Hausdorff Levels

In this section, we show that when the class of covers Φ is generated by computable Cantor series expansions, the faithfulness at the Hausdorff and constructive levels are equivalent notions.

6.1 Faithfulness of Family of Coverings

We will first see the definition of Hausdorff dimension faithfulness. We then introduce the corresponding notion at the effective level, which we call constructive dimension faithfulness.

A family of covering sets Φ is said to be *faithful* with respect to Hausdorff dimension if the Φ dimension of every set in the space is the same as its Hausdorff dimension.

▶ **Definition 37.** *A family of covering sets Φ over the space $\mathbb{X}$ is said to be* faithful with respect to Hausdorff dimension *if for all $\mathcal{F} \subseteq \mathbb{X}$, $\dim_\Phi(\mathcal{F}) = \dim(\mathcal{F})$.*

We extend the definition to the constructive level as well. A family of computable covering sets Φ is defined to be *faithful* with respect to constructive dimension if the constructive Φ dimension of every set is the same as its constructive dimension.

▶ **Definition 38.** *A family of computable covering sets Φ is said to be* faithful with respect to constructive dimension *if for all $\mathcal{F} \subseteq \mathbb{X}$, $\operatorname{cdim}_\Phi(\mathcal{F}) = \operatorname{cdim}(\mathcal{F})$.*

The following lemma follows from Theorem 26 and Lemma 11. It states that constructive dimension faithfulness can be equivalently stated in terms of preservation of constructive dimensions of points in the set.

▶ **Lemma 39.** *A constructive family of covers Φ is faithful with respect to Constructive dimension if and only if for all $x \in \mathbb{X}$, $\mathrm{cdim}_{\Phi}(x) = \mathrm{cdim}(x)$.*

The following lemma states that the Φ-dimension of a set is always greater than or equal to its Hausdorff dimension. Similarly, the constructive $\Phi-$dimension of a set is always greater than or equal to its constructive dimension.

▶ **Lemma 40.** *For any family of covering sets Φ over $\mathbb{X}$, for all $\mathcal{F} \subseteq \mathbb{X}$, $\dim_{\Phi}(\mathcal{F}) \geq \dim(\mathcal{F})$.*

▶ **Lemma 41.** *For any family of finitely intersecting computable covering sets Φ over $\mathbb{X}$, for all $\mathcal{F} \subseteq \mathbb{X}$, $\mathrm{cdim}_{\Phi}(\mathcal{F}) \geq \mathrm{cdim}(\mathcal{F})$.*

Therefore we have that Φ is not faithful for Hausdorff dimension if and only if there exists an $\mathcal{F} \subset \mathbb{X}$ such that $\dim_{\Phi}(\mathcal{F}) > \dim(\mathcal{F})$. Similarly, Φ is not faithful for Constructive dimension if and only if there exists an $\mathcal{F} \subset \mathbb{X}$ such that $\mathrm{cdim}_{\Phi}(\mathcal{F}) > \mathrm{cdim}(\mathcal{F})$.

6.2 Cantor coverings over unit interval

We consider the faithfulness of family of coverings generated by the computable Cantor series expansion [4]. We call such class of coverings as *Cantor coverings.*

Given a sequence $Q = \{n_k\}_{k\in\mathbb{N}}$ with $n_k \in \mathbb{N} \setminus \{1\}$, the expression

$$x = \sum_{k=1}^{\infty} \frac{\alpha_k}{n_1.n_2 \dots n_k}$$

where $\alpha_k \in [n_k]$ is called the cantor series expansion of the real number $x \in [0, 1]$.

▶ **Definition 42** (Cantor Coverings Φ_Q)**.** *The class of Cantor coverings Φ_Q over the space $\mathbb{X} = [0, 1]$ generated by the Cantor series expansion $Q = \{n_k\}_{k\in\mathbb{N}}$ is the set of intervals*

$$\bigcup_{k\in\mathbb{Z}} \{[\frac{m}{n_0.n_1.n_2 \dots n_k}, \frac{m+1}{n_0.n_1.n_2 \dots n_k}]\}_{m\in[n_0.n_1.n_2 \dots n_k]}$$

with n_0 taken as 1.

▶ **Definition 43** (Computable Cantor Coverings)**.** *The cantor series expansion $Q = \{n_k\}_{k\in\mathbb{N}}$ is said to be computable if there exists a machine that generates n_k given k. We call the class of Cantor coverings Φ_Q generated by a computable Cantor series expansion Q as a class of Computable Cantor Coverings over $\mathbb{X} = [0, 1]$.*

It is routine to verify that for any computable Cantor series expansion $Q = \{n_k\}_{k\in\mathbb{N}}$, the Cantor covering Φ_Q is a family of finitely intersecting computable covering sets from Definition 33. Therefore, from Theorem 27, we have the following Point-to-Set principle for Cantor covering dimension.

▶ **Corollary 44.** *For all $\mathcal{F} \subseteq \mathbb{X}$ and for all computable Cantor coverings Φ_Q,*

$$\dim_{\Phi_Q}(\mathcal{F}) = \min_{A\subseteq\mathbb{N}} \sup_{x\in\mathcal{F}} \mathrm{cdim}^{A}_{\Phi_Q}(x).$$

6.3 Kolmogorov Complexity Characterization of Cantor Series Dimension

We first show a Kolmogorov complexity characterization of constructive Φ-dimension for computable Cantor coverings.

▶ **Theorem 45.** *For any $x \in \mathbb{X}$, and any computable Cantor coverings Φ_Q generated by $Q = \{n_k\}_{k\in\mathbb{N}}$,*

$$\mathrm{cdim}_{\Phi_Q}(x) = \liminf_{k\to\infty} \frac{K(X \upharpoonright m_k)}{m_k}$$

where X is a binary expansion of x and $m_k = \lfloor \log_2(n_1.n_2 \ldots n_k) \rfloor$.

Theorem 8 ensures that when the Kolmogorov complexities of any two $X, Y \in \Sigma^\infty$ align over all finite prefixes, their constructive dimensions become equal. From Theorem 45, we get that when this happens, the constructive Φ-dimensions also become equal.

▶ **Lemma 46.** *For any $x, y \in \mathbb{X}$, $A, B \subseteq \mathbb{N}$ and any class of computable Cantor coverings Φ, if for all n, $|K^A(X \upharpoonright n) - K^B(Y \upharpoonright n)| = o(n)$, then $\mathrm{cdim}^A(x) = \mathrm{cdim}^B(y)$ and $\mathrm{cdim}^A_\Phi(x) = \mathrm{cdim}^B_\Phi(y)$. Here X and Y are the binary expansions of x and y respectively.*

6.4 Faithfulness of Cantor Coverings

Using the Point-to-Set Principle and properties of Kolmogorov complexity, we show that the notions of faithfulness for Cantor coverings at Hausdorff and Constructive levels are equivalent.

We first show that if a class of computable Cantor coverings Φ is faithful with respect to constructive dimension, then Φ is also faithful with respect to Hausdorff dimension.

▶ **Lemma 47.** *For any class of computable Cantor coverings Φ, if for all $\mathcal{F} \subseteq \mathbb{X}$, $\mathrm{cdim}(\mathcal{F}) = \mathrm{cdim}_\Phi(\mathcal{F})$, then for all $\mathcal{F} \subseteq \mathbb{X}$, $\dim(\mathcal{F}) = \dim_\Phi(\mathcal{F})$.*

To prove the converse, we require the construction of set $\mathcal{I}_s$ that contains all points in $\mathbb{X}$ having constructive dimension equal to s.

▶ **Definition 48.** *Given $s \in [0, \infty)$, define $\mathcal{I}_s = \{x \in \mathbb{X} \mid \mathrm{cdim}(x) = s\}$.*

Lutz and Weihrauch [17] showed that the Hausdorff dimension of $\mathcal{I}_s$ is equal to s. We provide a simple alternate proof of this using the point-to-set principle.

▶ **Lemma 49** (Lutz and Weihrauch [17])**.** $\dim(\mathcal{I}_s) = s$.

We now show that if a class of computable Cantor coverings Φ is faithful with respect to Hausdorff dimension, then Φ is also faithful with respect to constructive dimension.

▶ **Lemma 50.** *For any class of computable Cantor coverings Φ, if for all $\mathcal{F} \subseteq \mathbb{X}$, $\dim(\mathcal{F}) = \dim_\Phi(\mathcal{F})$, then for all $\mathcal{F} \subseteq \mathbb{X}$, $\mathrm{cdim}(\mathcal{F}) = \mathrm{cdim}_\Phi(\mathcal{F})$.*

Therefore, we have the following theorem which states that for the classes of Cantor coverings Φ, faithfulness with respect to Hausdorff and Constructive dimensions are equivalent notions.

▶ **Theorem 51.** *For any class of computable Cantor coverings Φ,*

$$\forall \mathcal{F} \subseteq \mathbb{X}\ ;\ \dim(\mathcal{F}) = \dim_\Phi(\mathcal{F}) \iff \forall \mathcal{F} \subseteq \mathbb{X}\ ;\ \mathrm{cdim}(\mathcal{F}) = \mathrm{cdim}_\Phi(\mathcal{F}).$$

6.5 Log limit condition for faithfulness

Albeverio, Ivanenko, Lebid and Torbin [1] showed that the Cantor coverings Φ_Q generated by the Cantor series expansions of $Q = \{n_k\}_{k\in\mathbb{N}}$ can be faithful as well as non faithful with respect to Hausdorff dimension depending on Q. Interestingly, they showed that the Hausdorff dimension faithfulness of Cantor coverings can be determined using the terms n_k in Q.

▶ **Theorem 52** (Albeverio, Ivanenko, Lebid and Torbin [1]). *A family of Cantor coverings Φ_Q generated by $Q = \{n_k\}_{k\in\mathbb{N}}$ is faithful with respect to Hausdorff dimension if and only if* $\lim\limits_{k\to\infty} \frac{\log n_k}{\log n_1 \cdot n_2 \dots n_{k-1}} = 0$.

Using the above result and the result that faithfulness at the constructive level is equivalent to faithfulness with respect to Hausdorff dimension (Theorem 51), we have that the condition stated above provides the necessary and sufficient conditions for Cantor series coverings to be faithful for constructive dimension.

▶ **Theorem 53.** *A family of Cantor coverings Φ_Q generated by $Q = \{n_k\}_{k\in\mathbb{N}}$ is faithful with respect to constructive dimension if and only if*

$$\lim_{k\to\infty} \frac{\log n_k}{\log n_1 . n_2 \dots n_{k-1}} = 0. \tag{1}$$

The Cantor series expansion is a generalization of the base-b representation, which is the special case when $n_k = b$ for all $k \in \mathbb{N}$. That is $Q_b = \{b\}_{n\in\mathbb{N}}$. Since the condition in Theorem 53 is satisfied by Q_b for any $b \in \mathbb{N}$, we have the following result by Hitchcock and Mayordomo about the base invariance of constructive dimension.

▶ **Corollary 54** (Hitchcock and Mayordomo [8]). *For any $x \in [0,1]$ and $k, l \in \mathbb{N} \setminus \{1\}$, $\mathrm{cdim}_{(k)}(x) = \mathrm{cdim}_{(l)}(x)$. where $\mathrm{cdim}_{(k)}(x)$ represents the constructive dimension of x with respect to its base-k representation.*

Note that condition (1) classifies the Cantor series expansions on the basis of constructive dimension faithfulness. As an example, when $n_k = 2^k$, condition (1) holds, and therefore $Q = \{2^k\}_{k\in\mathbb{N}}$ is faithful for constructive dimension. However, when $n_k = 2^{2^k}$, condition (1) does not hold, and therefore $Q = \{2^{2^k}\}_{k\in\mathbb{N}}$ is not faithful for constructive dimension.

7 Conclusion and Open Problems

We develop a constructive analogue of Φ-dimension and prove a Point-to-Set principle for Φ-dimension. Using this, we show that for Cantor series representations, constructive dimension faithfulness and Hausdorff dimension are equivalent notions. We also provide a loglimit condition for faithfulness of Cantor series expansions.

The following are some problems that remain open

1. Are the faithfulness at constructive and Hausdorff levels equivalent for all computable family of covering sets Φ ?
2. What is the packing dimension analogue of faithfulness, is there any relationship between faithfulness of Hausdorff dimension and packing dimension ?
3. Is there any relationship between faithfulness of constructive dimension and constructive strong dimension ?

References

1 S. Albeverio, Ganna Ivanenko, Mykola Lebid, and Grygoriy Torbin. On the Hausdorff dimension faithfulness and the Cantor series expansion. *Methods of Functional Analysis and Topology*, 26(4):298–310, 2020.

2 Sergio Albeverio and Grygoriy Torbin. Fractal properties of singular probability distributions with independent Q^*-digits. *Bull. Sci. Math.*, 129(4):356–367, 2005. doi:10.1016/j.bulsci.2004.12.001.

3 A. S. Besicovitch. On existence of subsets of finite measure of sets of infinite measure. *Indag. Math.*, 14:339–344, 1952. Nederl. Akad. Wetensch. Proc. Ser. A **55**.

4 George Cantor. Uber die einfachen zahlensysteme, zeit. für math. 14 (1869), 121-128. *Coll. Papers, Berlin*, pages 35–53, 1932.

5 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic randomness and complexity.* Theory and Applications of Computability. Springer, New York, 2010. doi:10.1007/978-0-387-68441-3.

6 Kenneth Falconer. *Fractal geometry.* John Wiley & Sons, Inc., Hoboken, NJ, second edition, 2003. Mathematical foundations and applications. doi:10.1002/0470013850.

7 F. Hausdorff. Dimension und äusseres Mass. *Mathematische Annalen*, 79:157–179, 1919.

8 John M. Hitchcock and Elvira Mayordomo. Base invariance of feasible dimension. *Inform. Process. Lett.*, 113(14-16):546–551, 2013. doi:10.1016/j.ipl.2013.04.004.

9 M. Kh. Ībragīm and G. M. Torbīn. On a probabilistic approach to the DP-transformations and faithfulness of covering systems for calculating the Hausdorff-Besicovitch dimension. *Teor. Ĭmovīr. Mat. Stat.*, 92:28–40, 2015. doi:10.1090/tpms/980.

10 Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications.* Texts in Computer Science. Springer, New York, third edition, 2008. doi:10.1007/978-0-387-49820-1.

11 J. H. Lutz. Dimensions of individual strings and sequences. *Information and Computation*, 187(1):49–79, 2003.

12 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003. doi:10.1137/S0097539701417723.

13 Jack H. Lutz and Neil Lutz. Algorithmic information, plane kakeya sets, and conditional dimension. *ACM Trans. Comput. Theory*, 10(2):7:1–7:22, 2018. doi:10.1145/3201783.

14 Jack H. Lutz and Neil Lutz. Who asked us? how the theory of computing answers questions about analysis. In Ding-Zhu Du and Jie Wang, editors, *Complexity and Approximation - In Memory of Ker-I Ko*, volume 12000 of *Lecture Notes in Computer Science*, pages 48–56. Springer, 2020. doi:10.1007/978-3-030-41672-0_4.

15 Jack H. Lutz, Neil Lutz, and Elvira Mayordomo. Extending the reach of the point-to-set principle. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 48:1–48:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.48.

16 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM Journal on Computing*, 38:1080–1112, 2008.

17 Jack H. Lutz and Klaus Weihrauch. Connectivity properties of dimension level sets. *MLQ Math. Log. Q.*, 54(5):483–491, 2008. doi:10.1002/malq.200710060.

18 Neil Lutz. Fractal intersections and products via algorithmic dimension. *ACM Trans. Comput. Theory*, 13(3):Art. 14, 15, 2021. doi:10.1145/3460948.

19 Neil Lutz and D. M. Stull. Bounding the dimension of points on a line. *Inform. and Comput.*, 275:104601, 15, 2020. doi:10.1016/j.ic.2020.104601.

20 Neil Lutz and Donald M. Stull. Projection theorems using effective dimension. In *43rd International Symposium on Mathematical Foundations of Computer Science*, volume 117 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 71, 15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

21 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inform. Process. Lett.*, 84(1):1–3, 2002. `doi:10.1016/S0020-0190(02)00343-5`.

22 Elvira Mayordomo. Effective hausdorff dimension in general metric spaces. *Theory Comput. Syst.*, 62(7):1620–1636, 2018. `doi:10.1007/s00224-018-9848-3`.

23 Satyadev Nandakumar. An effective ergodic theorem and some applications. In *STOC'08*, pages 39–44. ACM, New York, 2008. `doi:10.1145/1374376.1374383`.

24 Satyadev Nandakumar, Akhil S, and Prateek Vishnoi. Effective continued fraction dimension versus effective Hausdorff dimension of reals. In *48th International Symposium on Mathematical Foundations of Computer Science*, volume 272 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Paper No. 70, 15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. `doi:10.4230/lipics.mfcs.2023.70`.

25 Satyadev Nandakumar and Prateek Vishnoi. Randomness and effective dimension of continued fractions. In *45th International Symposium on Mathematical Foundations of Computer Science*, volume 170 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 73, 13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

26 Satyadev Nandakumar and Prateek Vishnoi. On continued fraction randomness and normality. *Information and Computation*, 285(part B):104876, 2022. `doi:10.1016/j.ic.2022.104876`.

27 André Nies. *Computability and randomness*, volume 51. OUP Oxford, 2009.

28 Yuval Peres and Gyorgiy Torbin. Continued fractions and dimensional gaps. In preparation.

29 C. A. Rogers. *Hausdorff measures.* Cambridge University Press, London-New York, 1970.

30 Alexander Shen, Vladimir A Uspensky, and Nikolay Vereshchagin. *Kolmogorov complexity and algorithmic randomness*, volume 220. American Mathematical Society, 2022.

31 Ludwig Staiger. The Kolmogorov complexity of real numbers. *Theor. Comput. Sci.*, 284(2):455–466, 2002. `doi:10.1016/S0304-3975(01)00102-5`.

Toward Grünbaum's Conjecture for 4-Connected Graphs

Christian Ortlieb
Institute of Computer Science, University of Rostock, Germany

Abstract

Given a spanning tree T of a 3-connected planar graph G, the *co-tree* of T is the spanning tree of the dual graph G^* given by the duals of the edges that are not in T. Grünbaum conjectured in 1970 that there is such a spanning tree T such that T and its co-tree both have maximum degree at most 3.

In 2014, Biedl proved that there is a spanning tree T such that T and its co-tree have maximum degree at most 5. Using structural insights into Schnyder woods, Schmidt and the author recently improved this bound on the maximum degree to 4. In this paper, we prove that in a 4-connected planar graph there exists a spanning tree T of maximum degree at most 3 such its co-tree has maximum degree at most 4. This almost solves Grünbaum's conjecture for 4-connected graphs.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases 4-connected planar graph, spanning tree, maximum degree, Schnyder wood, Grünbaum

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.77

Funding This research is supported by the grant SCHM 3186/2-1 (401348462) from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation).

1 Introduction

In 1966, Barnette showed that every 3-connected planar graph has a spanning 3-tree [3]. Here and in the following, a k-tree denotes a tree with maximum degree at most k. The dual of a 3-connected planar graph G is also 3-connected and planar. As mentioned in the abstract, for a spanning tree T the duals of the edges $E(G) - E(T)$ form a spanning tree of the dual graph G^*, the so called co-tree $\neg T^*$ of T. Thus, the question of a simultaneous bound on the maximum degree of T and its co-tree naturally arises. In 1970, Grünbaum made the following conjecture.

▶ **Conjecture 1** (Grünbaum [16, p. 1148], 1970). *Every planar 3-connected graph G contains a 3-tree T whose co-tree $\neg T^*$ is also a 3-tree.*

While Grünbaum's conjecture is to the best of our knowledge still unsolved, progress has been made by Biedl [4], who proved the existence of a 5-tree whose co-tree is a 5-tree. Her approach uses structural properties of canonical orderings. Schmidt and the author recently proved that in a 3-connected planar graph there is a spanning tree T such that T and its co-tree have maximum degree at most 4 [20]. In this paper, we prove that in a 4-connected planar graph there is a spanning 3-tree such that its co-tree is a spanning 4-tree of the dual. We use structural properties of minimal Schnyder woods. Schnyder woods are a tool which is widely applied in graph drawing [1,13,15,21,22] and beyond [6,7,10,17].

Our approach divides into two steps. Let G be the graph in question. First, we identify a suitable candidate graph H. This candidate is a spanning and connected subgraph of G of maximum degree at most 3. By the well-known cut-cycle duality [11, Prop. 4.6.1], its co-graph (that is the graph with edge set $E(G^*) - E(H)^*$) is acyclic. We show that the co-graph also has maximum degree 3. Then, we only lack the acyclicity of the primal graph.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 77; pp. 77:1–77:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Thus, in the second step, we delete edges of the candidate graph H such that it becomes acyclic and the co-graph remains acyclic. Then, we show that the degree of the resulting co-graph does not exceed 4. This then yields the desired 3-tree such that its co-tree is a 4-tree. The first step is far from being trivial. Especially, it is hard to prove the connectivity of the candidate graph. The second step uses parts of the proof of the main theorem of [20]. Hence, we focus on the first step.

We discuss Schnyder woods, their lattice structure and ordered path partitions in Section 2, the candidate graph H in Section 3 and the second and final step in Section 4. Due to space limitations some proofs are omitted or only sketched.

2 Schnyder Woods and Ordered Path Partitions

We only consider simple undirected graphs. A graph is *plane* if it is planar and embedded into the Euclidean plane without intersecting edges. The *neighborhood of a vertex set A* is the union of the neighborhoods of vertices in A. Although parts of this paper use orientation on edges, we will always let vw denote the undirected edge $\{v, w\}$.

2.1 Schnyder Woods

Let $\sigma := \{r_1, r_2, r_3\}$ be a set of three vertices of the outer face boundary of a plane graph G in clockwise order (but not necessarily consecutive). We call r_1, r_2 and r_3 *roots*. The *suspension G^σ* of G is the graph obtained from G by adding at each root of σ a half-edge pointing into the outer face. With a little abuse of notation, we define a *half-edge* as an arc that has a startvertex but no endvertex.

▶ **Definition 2** (Felsner [12]). *Let $\sigma = \{r_1, r_2, r_3\}$ and G^σ be the suspension of a 3-connected plane graph G. A* Schnyder wood *of G^σ is an orientation and coloring of the edges of G^σ (including the half-edges) with the colors 1,2,3 (red, green, blue) such that*

(a) *Every edge e is oriented in one direction (we say e is* unidirected*) or in two opposite directions (we say e is* bidirected*). Every direction of an edge is colored with one of the three colors 1,2,3 (we say an edge is i-*colored *if one of its directions has color i) such that the two colors i and j of every bidirected edge are distinct (we call such an edge i-j-*colored*). Throughout the paper, we assume modular arithmetic on the colors 1,2,3 in such a way that $i+1$ and $i-1$ for a color i are defined as $(i \mod 3)+1$ and $(i+1 \mod 3)+1$, respectively. For a vertex v, a uni- or bidirected edge is* incoming *(i-colored)* in *v if it has a direction (of color i) that is directed toward v, and* outgoing *(i-colored)* of *v if it has a direction (of color i) that is directed away from v.*

(b) *For every color i, the half-edge at r_i is unidirected, outgoing and i-colored.*

(c) *Every vertex v has exactly one outgoing edge of every color. The outgoing 1-, 2-, 3-colored edges e_1, e_2, e_3 of v occur in clockwise order around v. For every color i, every incoming i-colored edge of v is contained in the clockwise sector around v from e_{i+1} to e_{i-1} (see Figure 1).*

(d) *No inner face boundary contains a directed cycle (disregarding possible opposite edge directions) in one color.*

For a Schnyder wood and color i, let T_i be the directed graph that is induced by the directed edges of color i. The following result justifies the name of Schnyder woods.

▶ **Lemma 3** ([13, 21]). *For every color i of a Schnyder wood of G^σ, T_i is a directed spanning tree of G in which all edges are oriented to the root r_i.*

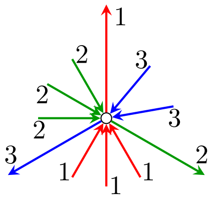


Figure 1 Properties of Schnyder woods. Condition 2c at a vertex.

For a directed graph H, we denote by H^{-1} the graph obtained from H by reversing the direction of all its edges.

▶ **Lemma 4** (Felsner [15]). *For every* $i \in \{1, 2, 3\}$, $T_i^{-1} \cup T_{i+1}^{-1} \cup T_{i+2}$ *is acyclic.*

▶ **Lemma 5** (folklore). *Let* S *be a Schnyder wood of* G^σ. *Then,* G *is internally triangulated, i.e., every face except the outer face is a triangle if and only if every internal edge of* G *is unidirected in* S.

2.2 Dual Schnyder Woods

Let G be a 3-connected plane graph. Any Schnyder wood of G^σ induces a Schnyder wood of a slightly modified planar dual of G^σ in the following way [9, 14] (see [19, p. 30] for an earlier variant of this result given without proof). As common for plane duality, we will use the plane dual operator * to switch between primal and dual objects (also on sets of objects).

Extend the three half-edges of G^σ to non-crossing infinite rays and consider the planar dual of this plane graph. Since the infinite rays partition the outer face f of G into three parts, this dual contains a triangle with vertices b_1, b_2 and b_3 instead of the outer face vertex f^* such that b_i^* is not incident to r_i for every i (see Figure 2). Let the *suspended dual* G^{σ^*} of G^σ be the graph obtained from this dual by adding at each vertex of $\{b_1, b_2, b_3\}$ a half-edge pointing into the outer face.

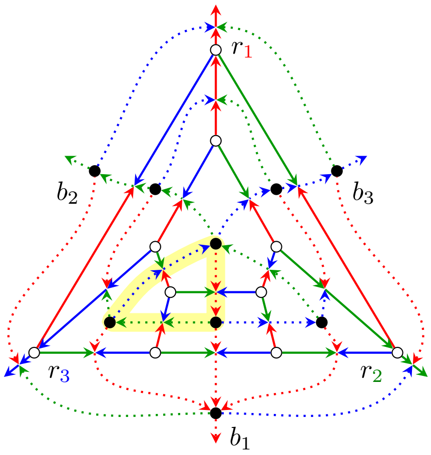


Figure 2 The completion of G obtained by superimposing G^σ and its suspended dual G^{σ^*} (the latter depicted with dotted edges). The primal Schnyder wood is not the minimal element of the lattice of Schnyder woods of G, as this completion contains a clockwise directed cycle (marked in yellow).

Consider the superposition of G^σ and its suspended dual G^{σ^*} such that exactly the primal dual pairs of edges cross (here, for every $1 \leq i \leq 3$, the half-edge at r_i crosses the dual edge $b_{i-1}b_{i+1}$).

▶ **Definition 6.** *For any Schnyder wood S of G^σ, define the orientation and coloring S^* of the suspended dual G^{σ^*} as follows (Figure 2):*

(a) *For every unidirected $(i-1)$-colored edge or half-edge e of G^σ, color e^* with the two colors i and $i+1$ such that e points to the right of the i-colored direction.*

(b) *Vice versa, for every i-$(i+1)$-colored edge e of G^σ, $(i-1)$-color e^* unidirected such that e^* points to the right of the i-colored direction.*

(c) *For every color i, make the half-edge at b_i unidirected, outgoing and i-colored.*

The following lemma states that S^* is indeed a Schnyder wood of the suspended dual. The vertices b_1, b_2 and b_3 are called the *roots* of S^*.

▶ **Lemma 7** ([18], [14, Prop. 3])**.** *For every Schnyder wood S of G^σ, S^* is a Schnyder wood of G^{σ^*}.*

Since $S^{*^*} = S$, Lemma 7 gives a bijection between the Schnyder woods of G^σ and the ones of G^{σ^*}. Let the *completion* $\widetilde{G}$ of G be the plane graph obtained from the superposition of G^σ and G^{σ^*} by subdividing each pair of crossing (half-)edges with a new vertex, which we call a *crossing vertex* (Figure 2). The completion has six half-edges pointing into its outer face.

Any Schnyder wood S of G^σ implies the following natural orientation and coloring $\widetilde{G}_S$ of its completion $\widetilde{G}$: For any edge $vw \in E(G^\sigma) \cup E(G^{\sigma^*})$, let z be the crossing vertex of $\widetilde{G}$ that subdivides vw and consider the coloring of vw in either S or S^*. If vw is outgoing of v and i-colored, we direct $vz \in E(\widetilde{G})$ toward z and i-color it; analogously, if vw is outgoing of w and j-colored, we direct $wz \in E(\widetilde{G})$ toward z and j-color it. In the case that vw is unidirected, say w.l.o.g. incoming at v and i-colored, we direct $zv \in E(\widetilde{G})$ toward v and i-color it. The three half-edges of G^{σ^*} inherit the orientation and coloring of S^* for $\widetilde{G}_S$. By Definition 6, the construction of $\widetilde{G}_S$ implies immediately the following corollary.

▶ **Corollary 8.** *Every crossing vertex of $\widetilde{G}_S$ has one outgoing edge and three incoming edges and the latter are colored 1, 2 and 3 in counterclockwise direction.*

Using results on orientations with prescribed outdegrees on the respective completions, Felsner and Mendez [8, 13] showed that the set of Schnyder woods of a planar suspension G^σ forms a distributive lattice. The order relation of this lattice relates a Schnyder wood of G^σ to a second Schnyder wood if the former can be obtained from the latter by reversing the orientation of a directed clockwise cycle in the completion. This gives the following lemma.

▶ **Lemma 9** ([8, 13])**.** *For the minimal element S of the lattice of all Schnyder woods of G^σ, $\widetilde{G}_S$ contains no clockwise directed cycle.*

We call the minimal element of the lattice of all Schnyder woods of G^σ the *minimal Schnyder wood* of G^σ.

▶ **Lemma 10** (Di Battista et al. [9])**.** *The boundary of every internal face of G can be partitioned into six paths $P_{1,3}$, $p_{2,3}$, $P_{2,1}$, $p_{3,1}$, $P_{3,2}$ and $p_{1,2}$ which appear in that clockwise order. For those paths the following holds (see Figure 3).*

(a) *$P_{i,j}$ consists of one edge which is either unidirected i-colored, unidirected j-colored or i-j-colored. Color i is directed in clockwise direction and color j in counterclockwise direction around f.*

(b) *$p_{i,j}$ consists of a possibly empty sequence of i-j-colored edges such that color i is directed clockwise around f.*

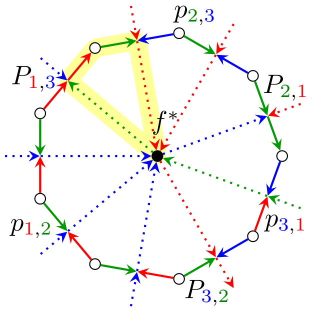


Figure 3 Illustration for Lemma 10. A face f, the paths on its boundary and the dual edges incident to f^*. If $P_{1,3}$ is unidirected 1-colored and $p_{2,3}$ is non-empty, there is a clockwise cycle in $\widetilde{G}_S$, marked in yellow.

2.3 Ordered Path Partitions

We denote paths as tuples of vertices such that consecutive vertices in the tuple are adjacent in the path. If a path P consists of only one vertex x, we might also write $P = x$. The concatenation of two paths P_1 and P_2 we denote by P_1P_2.

▶ **Definition 11.** *For any $j \in \{1,2,3\}$ and any $\{r_1, r_2, r_3\}$-internally 3-connected plane graph G, an* ordered path partition *$\mathcal{P} = (P_0, \dots, P_s)$ of G* with base-pair *(r_j, r_{j+1}) is a tuple of induced paths such that their vertex sets partition $V(G)$ and the following holds for every $i \in \{0, \dots, s-1\}$, where $V_i := \bigcup_{q=0}^{i} V(P_q)$ and the* contour *C_i is the clockwise walk from r_{j+1} to r_j on the outer face of $G[V_i]$.*

(a) *P_0 is the clockwise path from r_j to r_{j+1} on the outer face boundary of G, and $P_s = r_{j+2}$.*
(b) *Every vertex in P_i has a neighbor in $V(G) \setminus V_i$.*
(c) *C_i is a path.*
(d) *Every vertex in C_i has at most one neighbor in P_{i+1}.*

By Definition 11a and 11b, G contains for every i and every vertex $v \in P_i$ a path from v to r_{j+2} that intersects V_i only in v. Since G is plane, we conclude the following.

▶ **Lemma 12.** *Every path P_i of an ordered path partition is embedded into the outer face of $G[V_{i-1}]$ for every $1 \leq i \leq s$.*

2.3.1 Compatible Ordered Path Partitions

We describe a connection between Schnyder woods and ordered path partitions that was first given by Badent et al. [2, Theorem 5] and then revisited by Alam et al. [1, Lemma 1].

▶ **Definition 13.** *Let $j \in \{1,2,3\}$ and S be any Schnyder wood of the suspension G^σ of G. As proven in [1, arXiv version, Section 2.2], the inclusion-wise maximal j-$(j+1)$-colored paths of S then form an ordered path partition of G with base pair (r_j, r_{j+1}), whose order is a linear extension of the partial order given by reachability in the acyclic graph $T_j^{-1} \cup T_{j+1}^{-1} \cup T_{j+2}$; we call this special ordered path partition* compatible *with S and denote it by $\mathcal{P}^{j,j+1}$.*

For example, for the Schnyder wood given in Figure 2, $\mathcal{P}^{2,3}$ consists of the six maximal 2-3-colored paths, of which four are single vertices. We denote each path $P_i \in \mathcal{P}^{j,j+1}$ by $P_i := (v_1^i, \dots, v_k^i)$ such that $v_1^i v_2^i$ is outgoing j-colored at v_1^i.

Let C_i be as in Definition 11. By Definition 11c and Lemma 12, every path $P_i = (v_1^i, \ldots, v_k^i)$ of an ordered path partition satisfying $i \in \{1, \ldots, s\}$ has a neighbor $v_0^i \in C_{i-1}$ that is closest to r_{j+1} and a different neighbor $v_{k+1}^i \in C_{i-1}$ that is closest to r_j. We call v_0^i the *left neighbor* of P_i, v_{k+1}^i the *right neighbor* of P_i and $P_i^e := v_0^i P_i v_{k+1}^i$ the *extension* of P_i; we omit superscripts if these are clear from the context. For $0 < i \leq s$, let the path P_i *cover* an edge e or a vertex x if e or x is contained in C_{i-1}, but not in C_i, respectively.

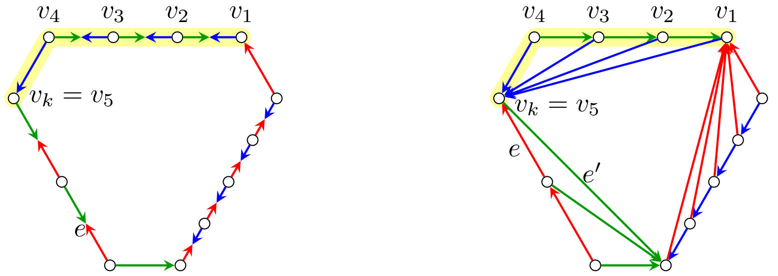
3 The Candidate Graph H

In this section, we define a special triangulation $\tau(G)$ of G. We also give some structural properties of $\tau(G)$. Afterwards, we define our candidate graph H as a subgraph of $\tau(G)$. Then, we are left to show some structural properties of H. First, we show that H is also a subgraph of G. Then, we argue that H and its co-graph have maximum degree at most 3. And finally, we show that if G is 4-connected, then H is connected. The latter will need some technical preparation.

Let P be the counterclockwise 3-colored path on the boundary of some internal face. By Lemma 10, P consists of $p_{2,3}$ (a possibly empty sequence of 2-3-colored edges) and possibly $P_{1,3}$ (an edge which is either unidirected 1-colored, unidirected 3-colored or 1-3-colored). Since S is minimal, we do not have clockwise cycles in $\widetilde{G}_S$. Hence, if $p_{2,3}$ is non-empty, then $P_{1,3}$ is either unidirected 3-colored or 3-1-colored (Figure 3), and we might define $\tau(G)$ as follows. A similar construction, but in the reverse direction, is used by Bonichon et al. [5].

▶ **Definition 14.** *Let G be a 3-connected planar graph and let S be the minimal Schnyder wood of G^σ. Define the internal triangulation $\tau(G)$ of G and the Schnyder wood of the σ-suspension of $\tau(G)$ to be the graph and Schnyder wood obtained by modifying every internal face f of G as follows (Figure 4). Let P be the counterclockwise 3-colored path on the boundary of f and let $v_1, \ldots, v_k$ be its vertices in counterclockwise order around f. If $k \geq 3$, proceed as follows. Add 3-colored edges $v_1v_k, \ldots, v_{k-2}v_k$ directed towards v_k and for $j = 2, \ldots, k-1$ change the color and orientation of v_jv_{j+1} such that v_jv_{j+1} is 2-colored and directed towards v_j. Proceed the same way for the counterclockwise 1-colored path and the counterclockwise 2-colored path on the boundary of f.*

Observe that if G is k-connected, then so is $\tau(G)$.



(a) An internal face of G.

(b) The corresponding subgraph of $\tau(G)$.

Figure 4 Illustration for the definition of $\tau(G)$. The counterclockwise 3-colored path P on the boundary of the face of G is highlighted in yellow.

▶ **Lemma 15.** *For a minimal Schnyder wood of G^σ, Definition 14 yields a minimal Schnyder wood of the σ-suspension of $\tau(G)$.*

▶ **Definition 16.** *Let G be a 3-connected plane graph. Let S be a minimal Schnyder wood of G^σ such that r_1r_3 and r_3r_2 are both edges of G. Define the subgraph H of $\tau(G)$ as follows (Figure 5). Let $V(H) = V(\tau(G))$. The edge set of H is defined in the following. The edges on the outer face of $\tau(G)$ that are either 1-2-colored or 2-3-colored are defined to be in $E(H)$. Now, we define which of the internal edges of $\tau(G)$ are in $E(H)$. For this, we treat the 1-2-colored edges and the 1-3-colored edge like unidirected 1-colored edges. The 2-3-colored edge we treat like a unidirected 2-colored edge. Then, for all $i \in \{1, 2, 3\}$, an internal i-colored edge e of $\tau(G)$ is in $E(H)$ if e and two $(i + 1)$-colored edges form a face. Observe that, by Definition 2c, this face needs to be right of e w.r.t. the orientation of e.*

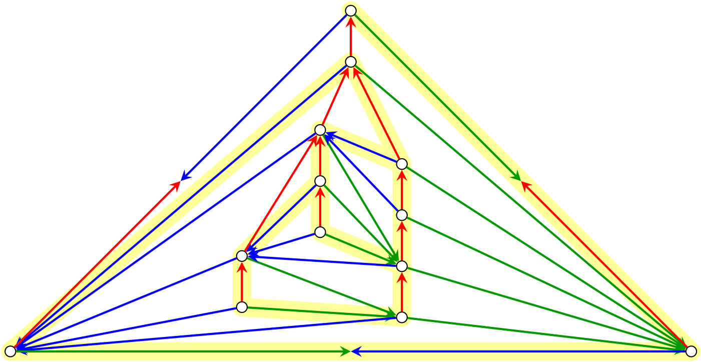


Figure 5 Illustration for the definition of H. H is depicted in yellow.

▶ **Lemma 17.** *As in Definition 16, treat the 1-2-colored edges and the 1-3-colored edge like unidirected 1-colored edges and the 2-3-colored edge like a unidirected 2-colored edge. An edge of $\tau(G)$ with head x is in $E(H)$ if and only if it is the first incoming i-colored edge in clockwise direction at x for some $i \in \{1, 2, 3\}$.*

▶ **Lemma 18.** *H has maximum degree at most 3.*

Sketch of proof. Treat the edges of the outer face of $\tau(G)$ as described in Definition 16. Let $v \in V(H)$ be a vertex that is not a root vertex. It is possible to argue for the root vertices in a similar way. We show that, for every $i \in \{1, 2, 3\}$, of the incoming i-colored and the outgoing $(i + 1)$-colored edges at v there is at most one edge in $E(H)$. Assume that at v there is an incoming edge $e = vy \in E(H)$, and w.l.o.g. e is 2-colored. By Definition 16, e and the outgoing 3-colored edge vx at v form a triangle with another 3-colored edge xy. By Definition 2c, xy is incoming 3-colored at x. Hence, xy precedes vx in clockwise order around x. Thus, by Lemma 17, $vx \notin E(H)$. Furthermore, by Lemma 17, e is the only incoming 2-colored edge at v that is in H.

If at v there is an outgoing edge $vw \in E(H)$, and w.l.o.g. vw is 3-colored. Then, by Definition 16, vw and the at v outgoing 1-colored edge vu are on the boundary of a face together with another 1-colored edge. By Definition 2c, incoming 2-colored edges at v only occur in the clockwise sector between vw and vu. As vw and vu are on the same face, this sector is empty and thus there is no incoming 2-colored edge at v.

Hence, for every $i \in \{1, 2, 3\}$, of the set of the incoming i-colored edges and the outgoing $(i + 1)$-colored edge at v there is at most one edge in $E(H)$. Those three sets cover all edges incident to v and hence $\deg_H(v) \le 3$. Similar arguments show that also the root vertices have degree at most 3. ◀

▶ **Lemma 19.** *H is a subgraph of G.*

Proof. Let $e \in E(\tau(G)) \setminus E(G)$ be w.l.o.g. 3-colored with tail v and head w. In the following, we show that e is no edge of H. This implies that H is a subgraph of G. Let f be the face of $\tau(G)$ that has v and w on the boundary in that clockwise order. Let $e' = wu$ be the edge succeeding e on f in clockwise order. As observed in Definition 14, wu is incoming 3-colored at w (Figure 4). If $w \neq r_3$, then wv is not the in clockwise order first incoming 3-colored edge at w in the sense of Lemma 17. Hence, $wv \notin E(H)$.

So let $w = r_3$. Assume, for the sake of contradiction, that $u = r_1$. The edge $uv = r_1v$ succeeding $wu = r_3r_1$ on f in clockwise order is outgoing 2-colored at u by Definition 14 (Figure 4). The only outgoing 2-colored edge at r_1 is on the clockwise path from r_1 to r_2 on the outer face of $\tau(G)$. And hence, v is on that path. As vr_3 is unidirected, $v \neq r_2$. Hence, $\{v, r_3\}$ is a 2-separator of $\tau(G)$ and thus of G, contradicting the 3-connectivity of G. This implies that $u \neq r_1$ and thus $r_1r_3 \neq wu$. As above, wv is not the in clockwise order first incoming 3-colored edge at w in the sense of Lemma 17, and thus, $wv \notin E(H)$. ◀

▶ **Lemma 20.** *The co-graph $\neg H^*$ of H in G has maximum degree at most 3. All edges in $E(\neg H^*)$ except $(r_1r_3)^*$ are bidirected.*

Sketch of proof. We show that all bidirected edges of G except r_1r_3 are in H. Let $e = xy$ be a bidirected internal edge in G.

Case 1. x and y are both internal vertices of G. Assume that e is w.l.o.g. a 2-3-colored edge in G. By Definition 14, e becomes 2-colored and is on a face with two 3-colored edges e_1 and e_2 in $\tau(G)$. As x and y are both internal vertices, e_1 and e_2 are both internal edges and, by Lemma 5, unidirected. And thus, $e \in E(H)$ by Definition 16.

Case 2. x and y are both on the outer face of G. If they do not appear consecutively, then they form a 2-separator of G, contradicting the 3-connectivity of G. Thus, xy is an edge on the outer face of G. Then, by Definition 16, $e \in E(H)$ if and only if $e \neq r_1r_3$.

Case 3. W.l.o.g. x is on the outer face of G and y is an internal vertex. This case follows with similar arguments as Case 1.

Hence, only $(r_1r_3)^*$ and the dual edges of unidirected edges might be edges of $\neg H^*$. By Corollary 8, the dual edges of unidirected edges in G are bidirected. Also, observe that $(r_1r_3)^*$ is unidirected and points into the outer face of G. Thus, for an internal face f of G only the outgoing edges of f^* might be in $\neg H^*$. Hence, $\deg_{\neg H^*}(f^*) \leq 3$. As, by Definition 16, only one edge on the boundary of the outer face of G is not in H, the dual of the outer face has degree 1 in $\neg H^*$. ◀

The following definition and lemmas are in preparation of the final statement (Proposition 26) of this section. They study the structure of $\tau(G)$ under the assumption that H is not connected. In the end, they allow us to show that if H is not connected, then there is a 3-separator in G (Figure 6). This yields that if G is 4-connected, then H is connected.

▶ **Definition 21.** *For $x \in V(\tau(G))$, define $DFS(x)$ to be the DFS-index of x for a depth first search on T_1 that starts at r_1 and explores the children of each vertex in counterclockwise order.*

For a vertex $x \in V(\tau(G))$ let $p_3(x)$ and $p_1(x)$ be the parent of x in T_3 and T_1, respectively (Figure 6). Let $c_2(x)$ be the vertex such that $c_2(x)x$ is the clockwise first incoming 2-colored edge at x if existent, i.e., $c_2(x)$ is the clockwise first child of x in T_2. Define $ld(x)$ ("leftmost descendant") to be the descendant v of x in T_1 such that v is a leaf of T_1 and $DFS(v)$ is minimal.

For a set of vertices $S \subseteq V(\tau(G))$ denote the set of the descendants of S in T_1 by $desc(S)$. Define $desc^+(S) := desc(S) \cup S$.

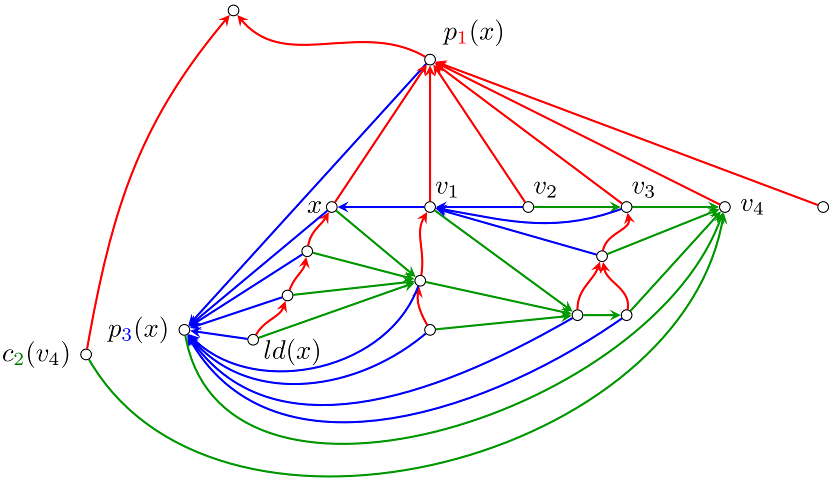


Figure 6 Illustration for Proposition 26. If x is not connected to a vertex with smaller DFS-index in H, then $p_3(x)$, $p_1(x)$ and $v_j = v_4$ form a separating triangle.

▶ Remark 22. Remember that, by Lemma 4, ${T_1}^{-1} \cup T_2 \cup {T_3}^{-1}$ is acyclic. Observe that the DFS-index is a total order of ${T_1}^{-1} \cup T_2 \cup {T_3}^{-1}$. E.g. for an edge xy that is 1-colored incoming at y we have that $DFS(x) > DFS(y)$.

Also, observe that we obtain the path from x to $ld(x)$ in T_1 by descending T_1 starting at x and choosing the counterclockwise first child until we hit a leaf of T_1.

▶ **Lemma 23.** *Assume x is not connected to $p_1(x)$ in H, i.e., there is no path in H that has x and $p_1(x)$ as endpoints. Then, the situation in $\tau(G)$ is as follows. Let $v_0, \dots, v_k$ be the DFS-ordered children of $p_1(x)$ in T_1, i.e., $DFS(v_0) < \dots < DFS(v_k)$, and $x = v_t$ for some $t \in \{0, \dots, k\}$. Then, there exists $j \in \{t+1, \dots, k\}$ such that $c_2(v_j)$ exists and is not in $desc^+(v_t, \dots, v_k)$ (Figure 6). We say that x has property B.*

Proof. Let the *height* of a vertex v in T_1 be the length of a longest oriented path in T_1 from a leaf of T_1 to v. The proof is by induction on the height of $p_1(x)$ in T_1. Assume that this height is one. Then, the vertices $x = v_t, \dots, v_k$ are all leaves in T_1. We now show that if x does not have property B, then x is connected to $p_1(x)$ in H. So assume that for every $i \in \{t+1, \dots, k\}$ either $c_2(v_i)$ does not exist or $c_2(v_i)$ is a vertex of $v_t, \dots, v_k$ in T_1.

If $c_2(v_i)$ does not exist, for some $i \in \{t+1, \dots, k\}$, i.e., v_i does not have an incoming 2-colored edge, then, as $\tau(G)$ is internally 3-connected, v_iv_{i-1} exists and is 3-colored by Definition 2c. As $v_ip_1(x)$ and $v_{i-1}p_1(x)$ are 1-colored, v_iv_{i-1} is in H by Definition 16. Otherwise, $c_2(v_i) = v_j$ for some $j \in \{t, \dots, k\}$. As observed in Definition 21, $DFS(c_2(v_i)) < DFS(v_i)$ and hence $j \in \{t, \dots, i-1\}$. Since $c_2(v_i)v_i$ is in $E(H)$ by Lemma 17, v_j and v_i are connected in H. Hence, in any case v_i is connected in H to a vertex v_j with $j \in \{t, \dots, i-1\}$. This finally yields that $x = v_t$ and v_k are connected in H. As $v_kp_1(x)$ is the clockwise first incoming 1-colored edge at $p_1(x)$, $v_kp_1(x)$ is in $E(H)$ by Lemma 17. And thus, $x = v_t$ and $p_1(x)$ are connected in H.

Assume that the height of $p_1(x)$ is at least two. Again, we show that if x does not have property B, then x and $p_1(x)$ are connected in H. As before, we observe that if $c_2(v_i)$ does not exist for some $i = t+1, \dots, k$, then v_i and v_{i-1} are connected in H. Also, if $c_2(v_i) = v_j$ for some $j \in \{t, \dots, k\}$, then $j < i$ and v_j and v_i are connected in H.

So assume that $c_2(v_i)$ is a descendant of v_j, $j \in \{t, \ldots, k\}$. For the same reason as above, $j \in \{t, \ldots, i-1\}$. Let P be the 1-colored path from $c_2(v_i)$ to v_j in $\tau(G)$. P, $c_2(v_i)v_i$, $v_i p_1(x)$ and $v_j p_1(x)$ form a cycle C. Observe that all vertices of C except for $p_1(x)$ are descendants of $p_1(x)$. Thus, by planarity, for every vertex in the interior of C the outgoing 1-colored path meets $p_1(x)$ or a descendant of $p_1(x)$. And thus, all vertices in the interior of C are descendants of $p_1(v)$ in T_1. Let $y \neq v_j$ be a vertex of P. As $p_1(y)$ is a descendant of $p_1(x)$, it has smaller height in T_1 than $p_1(x)$. We want to apply induction on y. Hence, we need to show that y does not have property B.

Assume, for the sake of contradiction, that y has property B. Then, there is a child z of $p_1(y)$ such that $DFS(y) < DFS(z)$, $c_2(z)$ exists and for every child b of $p_1(y)$ with $DFS(y) \leq DFS(b)$, $c_2(z)$ is neither a descendant nor $c_2(z) = b$. Especially, $DFS(c_2(z)) < DFS(y)$. All vertices of C and all vertices in its interior, except for vertices on the 1-colored path from $p_1(y)$ to $p_1(x)$, have a higher DFS-index than y. By Lemma 4, $c_2(z)$ cannot be on this path from $p_1(y)$ to $p_1(x)$. Thus, $c_2(z)$ cannot occur on C or in its interior. And hence, $c_2(z)$ needs to be in the exterior of C. As all neighbors of y that have a higher DFS-index than y are in the interior of C, we have that z is in the interior of C. Hence, $c_2(z)$ is in the exterior and z is in the interior of C, violating planarity.

This yields that y does not have property B and by induction y and $p_1(y)$ are connected in H. This holds for every vertex on $P \setminus \{v_j\}$. Thus, v_j and $c_2(v_i)$ are connected in H. By Lemma 17, $c_2(v_i)v_i \in E(H)$, and thus v_j and v_i are connected in H. As before, we obtain that x and $p_1(x)$ are connected in H. ◀

▶ **Lemma 24.** *If $x \in V(\tau(G)) \setminus \{r_1\}$ is not connected to a vertex v with $DFS(v) < DFS(x)$ in H, then x is not connected to $p_1(x)$ in H and x is the child of $p_1(x)$ in T_1 with smallest DFS-index. Furthermore, all vertices w on the x-$ld(x)$-path in T_1 are connected to x in H. And we have for all vertices w on the $p_1(x)$-$ld(x)$-path in T_1 that $p_3(w) = p_3(x)$ (Figure 6).*

Proof. Let $x \in V(\tau(G))$ such that x is not connected to a vertex v with $DFS(v) < DFS(x)$. Let $v_0, \ldots, v_k$ be the DFS-ordered children of $p_1(x)$ in T_1. First, we show that $x = v_0$. Assume, for the sake of contradiction, that $x = v_j$ with $j \in \{1, \ldots, k\}$. Then, xv_{j-1} has either color 2 or 3. If it has color 3, then, by Definition 16, xv_{j-1} is in H and $DFS(v_{j-1}) < DFS(x)$, a contradiction. If xv_{j-1} has color 2, then this edge is incoming 2-colored at x, by Definition 2c. This implies that $c_2(x)$ exists. $DFS(c_2(x)) < DFS(x)$ and, by Lemma 17, $c_2(x)x \in E(H)$, a contradiction. And hence, $x = v_0$.

Assume, for the sake of contradiction, that there exists a vertex w on the x-$ld(x)$-path in T_1 such that w is not connected to x in H. Choose w such that $DFS(w)$ is minimal. Then, $p_1(w)$ is connected to x in H, and hence, w is not connected to $p_1(w)$ in H. Thus, by Lemma 23, w has property B. Let $w = w_0, \ldots, w_k$ be the DFS-ordered children of $p_1(w)$ in T_1. Let j be the maximal index such that $c_2(w_j)$ exists and is not a descendant of $p_1(w)$. Since w has property B, j exists. Since j is maximal, w_j does not have property B. Hence, by Lemma 23, it is connected to $p_1(w)$ in H and thus to x. Since $c_2(w_j)w_j \in E(H)$, x is connected to $c_2(w_j)$ in H.

As observed in Definition 21, $DFS(c_2(w_j)) < DFS(w_j)$. The DFS-indices of the vertices $w_0, \ldots, w_{j-1}$ and their descendants are exactly the indices in between $DFS(w_j)$ and $DFS(w_0)$. And the DFS-indices of the vertices on the $p_1(w)$-x-path are exactly the indices in between $DFS(p_1(w))$ and $DFS(x)$. By Lemma 4, $c_2(w_j)$ cannot be on the $p_1(w)$-x-path. Hence, $DFS(c_2(w_j)) < DFS(x)$, a contradiction. Hence, w is connected to x in H.

As all vertices on the x-$ld(x)$-path in T_1 are connected to x in H, they all do not have incoming 2-colored edges. Indeed, incoming 2-colored edges at those vertices would connect x to a vertex with smaller DFS-index. Since every vertex on the x-$ld(x)$-path needs to form a triangle with its parent, the vertex and the parent send their 3-colored outgoing edge to the same vertex $p_3(x)$. ◀

Similar arguments are used to show the following lemma.

▶ **Lemma 25.** *Let $x \in V(\tau(G)) \setminus \{r_1\}$ be not connected to a vertex v with $DFS(v) < DFS(x)$ in H. Let $x = v_0, \dots, v_k$ be the DFS-ordered children of $p_1(x)$ in T_1. Let v_j be the vertex of smallest index such that $c_2(v_j)$ exists and is not in $desc^+(v_0, \dots, v_{j-1})$. Then, the edge $v_j p_3(x)$ is in $E(\tau(G))$.*

▶ **Proposition 26.** *If G is 4-connected, then H is connected.*

Proof. Observe that if G is 4-connected, then so is $\tau(G)$. We show that every vertex $x \neq r_1$ is connected in H to a vertex with lower DFS-index. Assume, for the sake of contradiction, that x is not connected to a vertex with lower DFS-index. With the help of the previous lemmas we show that $p_3(x)$, $p_1(x)$ and v_j (as defined in Lemma 23) form a 3-separator in H. Let $v_0, \dots, v_k$ be the DFS-ordered children of $p_1(x)$ in T_1.

By Lemma 23, there exists a vertex v_j, $j \in \{1, \dots, k\}$ of smallest DFS-index such that $c_2(v_j)$ exists and is not in $desc^+(v_0, \dots, v_{j-1})$. Observe that $p_1(x)v_j$ is an edge of $\tau(G)$.

By Lemma 24, $x = v_0$ and all vertices on the $p_1(x)$-$ld(x)$-path in T_1 send their 3-colored outgoing edge to the same vertex $p_3(x)$. Thus, $p_1(x)p_3(x) \in E(\tau(G))$.

By Lemma 25, the edge $v_j p_3(x)$ exists in $\tau(G)$. Hence, the edges $v_j p_3(x)$, $p_1(x)p_3(x)$ and $p_1(x)v_j$ form a triangle in $\tau(G)$. The vertex x is in the interior of this triangle. Assume, for the sake of contradiction, that r_1 is not in the exterior of this triangle. Then, $r_1 = p_1(x)$ and, by Lemma 24, $x = r_3$. By Definition 16, r_3 is connected to r_1 by the clockwise path on the outer face from r_1 to r_3. As $DFS(r_1) < DFS(r_3)$, we arrive at a contradiction. Hence, r_1 is in the exterior of the triangle of $v_j p_3(x)$, $p_1(x)p_3(x)$, i.e., this triangle is a separating triangle in $\tau(G)$, contradicting the 4-connectivity of $\tau(G)$. This yields that every vertex, except for r_1, is connected to a vertex with lower DFS-index in H, and thus H is connected. ◀

4 A Tree of Maximum Degree 3 and a Co-Tree of Maximum Degree 4

In this section, we give one lemma on the structure of ordered path partitions. Then, we finally prove the main theorem.

We want to remind the reader of the definition of ordered path partitions and the fact that the maximal 2-3-colored paths of a Schnyder wood yield the compatible ordered path partition $\mathcal{P}^{2,3}$.

▶ **Lemma 27** ([20]). *Let G be a 4-connected plane graph, S be the minimal Schnyder wood of G^σ and $\mathcal{P}^{2,3} = (P_0, \dots, P_s)$ be the ordered path partition that is compatible with S. Let $P_i := (v_1, \dots, v_k) \neq P_0$ be a path of $\mathcal{P}^{2,3}$ and v_0 and v_{k+1} be its left and right neighbor. Then, every edge $v_l w \notin \{v_0v_1, v_kv_{k+1}\}$ with $v_l \in P_i$ and $w \in V_{i-1}$ is unidirected, 1-colored and incoming at v_k and $w \notin \{v_0, v_{k+1}\}$.*

▶ **Theorem 28.** *Every 4-connected planar graph G contains a 3-tree such that its co-tree is a 4-tree.*

Sketch of proof. Let S be a minimal Schnyder wood of G such that r_1r_3 and r_3r_2 are both edges of G. By Lemma 9, the completion $\widetilde{G}_S$ of G contains no clockwise directed cycle. By

Proposition 26 and Lemma 19, G has a connected subgraph H as defined in Definition 16. It has maximum degree at most 3 by Lemma 18, and its co-graph has maximum degree at most 3 by Lemma 20. Now, we define a subset D of the edges of H such that $H - D$ becomes acyclic and the degree of its co-graph does not exceed 4.

Let C be a cycle in H. Let $\mathcal{P}^{2,3} = (P_0, \dots, P_s)$ be the compatible ordered path partition of S. Let P be the path of maximal length in C such that $P \subseteq P_M = (v_1, \dots, v_k)$ with $M := \max\{i \mid P_i \cap V(C) \neq \emptyset\}$. P is the *index maximal* subpath of C.

Since $P \subseteq P_M$ is the index maximal subpath of the cycle C, there need to be two edges in H that join a vertex of P with a vertex of V_{M-1}. Say that those edges are the *associated* edges of P. Remember that $V_i := \bigcup_{q=0}^{i} V(P_q)$. The Schnyder wood S is minimal. Hence, by Lemma 27, every edge except for v_0v_1 and v_kv_{k+1} that joins a vertex of P_M with a vertex of V_{M-1} is unidirected, 1-colored and incoming in v_k. Let e_P be the clockwise first incoming 1-colored edge at v_k if existent. Otherwise, let e_P be v_kv_{k+1}. It is possible to show that v_0v_1 and e_P are the only edges in H that join a vertex of P with a vertex of V_{M-1}. This also directly yields that $P = P_M$.

Let $\mathcal{P}_{max}$ be the set of all index maximal subpaths of cycles in H. Now, we need to identify for each $P = (v_1, \dots, v_k) \in \mathcal{P}_{max}$ an edge of the set $E(P) \cup \{v_0v_1, e_P\}$ such that removing those edges leaves H acyclic and connected and does not raise the maximum degree of $\neg H^*$ above 4. Define $D = D_{uni} \cup D_{bi}$ to be this set of edges. Start with $D_{uni} = D_{bi} = \emptyset$.

If e_P is unidirected, add it to D_{uni}. Otherwise, if e_P is bidirected and v_0v_1 is unidirected add v_0v_1 to D_{uni}. By Lemma 20, all edges of $\neg H^*$ except for $(r_1r_3)^*$ are bidirected. As all edges in D_{uni} are unidirected, their duals are bidirected, by Corollary 8. Hence, all edges in $\neg H^* + D^*_{uni}$ except for $(r_1r_3)^*$ are bidirected. By the same reasoning as in the proof of Lemma 20, the maximum degree of $\neg H^* + D^*_{uni}$ is at most 3.

Let $\mathcal{P}^{bi}_{max} \subseteq \mathcal{P}_{max}$ be the paths $P = (v_1, \dots, v_k)$ such that both v_0v_1 and e_P are bidirected. Observe that in this case $e_P = v_kv_{k+1}$. Hence, we now are in the same situation as in the proof of the main theorem of [20]. We are able to apply the exact same arguments in order to obtain the desired set D_{bi} such that $H - D$ becomes acyclic and $\neg H^* + D^*$ remains acyclic and has maximum degree at most 4. Since H has maximum degree at most 3, so does $H - D$. And thus, $H - D$ and $\neg H^* + D^*$ are the desired trees. ◀

References

1 Md. J. Alam, W. Evans, S. G. Kobourov, S. Pupyrev, J. Toeniskoetter, and T. Ueckerdt. Contact representations of graphs in 3D. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS '15)*, volume 9214 of *Lecture Notes in Computer Science*, pages 14–27, 2015. Technical Report accessible on arXiv: arxiv.org/abs/1501.00304. doi:10.1007/978-3-319-21840-3_2.

2 M. Badent, U. Brandes, and S. Cornelsen. More canonical ordering. *Journal of Graph Algorithms and Applications*, 15(1):97–126, 2011. doi:10.7155/JGAA.00219.

3 D. Barnette. Trees in polyhedral graphs. *Canadian Journal of Mathematics*, 18:731–736, 1966. doi:10.4153/CJM-1966-073-4.

4 T. Biedl. Trees and co-trees with bounded degrees in planar 3-connected graphs. In *14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'14)*, pages 62–73, 2014. doi:10.1007/978-3-319-08404-6_6.

5 Nicolas Bonichon, Stefan Felsner, and Mohamed Mosbah. Convex drawings of 3-connected plane graphs. *Algorithmica*, 47(4):399–420, 2007.

6 Luca Castelli Aleardi and Olivier Devillers. Array-based compact data structures for triangulations: practical solutions with theoretical guarantees. *J. Comput. Geom.*, 9(1):247–289, 2018. doi:10.20382/jocg.v9i1a8.

7 Julien Courtiel, Eric Fusy, Mathias Lepoutre, and Marni Mishna. Bijections for Weyl chamber walks ending on an axis, using arc diagrams and Schnyder woods. *European J. Combin.*, 69:126–142, 2018. doi:10.1016/j.ejc.2017.10.003.

8 P. O. de Mendez. *Orientations bipolaires*. PhD thesis, École des Hautes Études en Sciences Sociales, Paris, 1994.

9 G. Di Battista, R. Tamassia, and L. Vismara. Output-sensitive reporting of disjoint paths. *Algorithmica*, 23(4):302–340, 1999. doi:10.1007/PL00009264.

10 Emilio Di Giacomo, Giuseppe Liotta, and Tamara Mchedlidze. Lower and upper bounds for long induced paths in 3-connected planar graphs. *Theoret. Comput. Sci.*, 636:47–55, 2016. doi:10.1016/j.tcs.2016.04.034.

11 R. Diestel. *Graph theory*. Graduate texts in mathematics 173. Springer, Berlin, 4th edition edition, 2012. URL: http://swbplus.bsz-bw.de/bsz377230375cov.htm.

12 S. Felsner. Geodesic embeddings and planar graphs. *Order*, 20:135–150, 2003.

13 S. Felsner. *Geometric Graphs and Arrangements*. Advanced Lectures in Mathematics. Vieweg+Teubner, Wiesbaden, 2004. doi:10.1007/978-3-322-80303-0.

14 S. Felsner. Lattice structures from planar graphs. *Electronic Journal of Combinatorics*, 11(1):R15, 1–24, 2004. doi:10.37236/1768.

15 Stefan Felsner. Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order*, 18(1):19–37, 2001. doi:10.1023/A:1010604726900.

16 B. Grünbaum. Polytopes, graphs, and complexes. *Bulletin of the American Mathematical Society*, 76(6):1131–1201, 1970. doi:10.1090/S0002-9904-1970-12601-5.

17 Xin He and Huaming Zhang. A simple routing algorithm based on Schnyder coordinates. *Theoret. Comput. Sci.*, 494:112–121, 2013. doi:10.1016/j.tcs.2013.01.017.

18 G. Kant. Drawing planar graphs using the lmc-ordering. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 101–110, 1992. doi:10.1109/SFCS.1992.267814.

19 G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996. doi:10.1007/BF02086606.

20 Christian Ortlieb and Jens M. Schmidt. Toward Grünbaum's Conjecture. In *19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024)*, volume 294 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2024.37.

21 Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 138–148, USA, 1990. Society for Industrial and Applied Mathematics.

22 Hendrik Schrezenmaier. Homothetic triangle contact representations. In *Graph-theoretic concepts in computer science*, volume 10520 of *Lecture Notes in Comput. Sci.*, pages 425–437. Springer, Cham, 2017. doi:10.1007/978-3-319-68705-6_3.

C_{2k+1}-Coloring of Bounded-Diameter Graphs

Marta Piecyk
Warsaw University of Technology, Poland

Abstract

For a fixed graph H, in the graph homomorphism problem, denoted by Hom(H), we are given a graph G and we have to determine whether there exists an edge-preserving mapping $\varphi : V(G) \to V(H)$. Note that Hom(C_3), where C_3 is the cycle of length 3, is equivalent to 3-Coloring. The question of whether 3-Coloring is polynomial-time solvable on diameter-2 graphs is a well-known open problem. In this paper we study the Hom(C_{2k+1}) problem on bounded-diameter graphs for $k \geq 2$, so we consider all other odd cycles than C_3. We prove that for $k \geq 2$, the Hom(C_{2k+1}) problem is polynomial-time solvable on diameter-$(k+1)$ graphs – note that such a result for $k = 1$ would be precisely a polynomial-time algorithm for 3-Coloring of diameter-2 graphs. Furthermore, we give subexponential-time algorithms for diameter-$(k+2)$ and -$(k+3)$ graphs.

We complement these results with a lower bound for diameter-$(2k+2)$ graphs – in this class of graphs the Hom(C_{2k+1}) problem is NP-hard and cannot be solved in subexponential-time, unless the ETH fails.

Finally, we consider another direction of generalizing 3-Coloring on diameter-2 graphs. We consider other target graphs H than odd cycles but we restrict ourselves to diameter 2. We show that if H is triangle-free, then Hom(H) is polynomial-time solvable on diameter-2 graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Graph coloring

Keywords and phrases graph homomorphism, odd cycles, diameter

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.78

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2403.06694 [29]

Funding *Marta Piecyk*: This research was funded by Polish National Science Centre, grant no. 2022/45/N/ST6/00237.

Acknowledgements We are grateful to Paweł Rzążewski for inspiring and fruitful discussions.

1 Introduction

A natural approach to computationally hard problems is to restrict the class of input graphs, for example by bounding some parameters. One of such parameters is the diameter, i.e., for a graph G, its diameter is the least integer d such that for every pair of vertices u, v of G, there is a u-v path on at most d edges. We say that G is a diameter-d graph, if its diameter is at most d. Recently, bounded-diameter graphs received a lot of attention [23, 20, 25, 3, 2, 24, 9, 6]. It is known that graphs from real life applications often have bounded diameter, for instance social networks tend to have bounded diameter [30]. Furthermore, almost all graphs have diameter 2, i.e., the probability that a random graph on n vertices has diameter 2 tends to 1 when n tends to infinity [21]. Therefore, solving a problem on bounded-diameter graphs captures a wide class of graphs. On the other hand, not all of the standard approaches can be used – note that the class of diameter-d graphs is not closed under vertex deletion.

Even if we consider the class of diameter-2 graphs, its members can contain any graph as an induced subgraph. Indeed, consider any graph G, and let G^+ be the graph obtained from G by adding a universal vertex u, i.e., we add vertex u and make it adjacent to all vertices of G. It is straightforward to observe that the diameter of G^+ is at most 2. This construction

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 78; pp. 78:1–78:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can be used for many graph problems on diameter-2 graphs as a hardness reduction, which proves that they cannot be solved in subexponential time under the Exponential Time Hypothesis (ETH, see [19]), for instance Max Independent Set.

The construction of G^+ also gives us a reduction from $(k-1)$-Coloring to k-Coloring on diameter-2 graphs, and thus for any $k \geq 4$, the k-Coloring problem on diameter-2 graphs is NP-hard and cannot be solved in subexponential time, unless the ETH fails. Note that this argument does not work for $k = 3$ since then we reduce from 2-Coloring, which is polynomial-time solvable. A textbook reduction from NAE-SAT implies that for $d \geq 4$, the 3-Coloring problem is NP-hard and cannot be solved in subexponential-time, unless the ETH fails [28]. Therefore, it is only interesting to study 3-Coloring on diameter-2 and -3 graphs. Mertzios and Spirakis proved that 3-Coloring is NP-hard on diameter-3 graphs [25]. However, the question of whether 3-Coloring can be solved in polynomial time on diameter-2 graphs remains open.

For 3-Coloring on diameter-2 graphs, subexponential-time algorithms were given, first by Mertzios and Spirakis with running time $2^{\mathcal{O}(\sqrt{n \log n})}$ [25]. This was later improved by Dębski, Piecyk, and Rzążewski, who gave an algorithm with running time $2^{\mathcal{O}(n^{1/3} \cdot \log^2 n)}$ [9]. They also provided a subexponential-time algorithm for 3-Coloring for diameter-3 graphs.

The 3-Coloring problem on bounded-diameter graphs was also intensively studied on instances with some additional restrictions, i.e., on graphs with some forbidden induced subgraphs – and on such graph classes polynomial-time algorithms were given [23, 20, 24].

One of the generalizations of graph coloring are homomorphisms. For a fixed graph H, in the graph homomorphism problem, denoted by Hom(H), we are given a graph G, and we have to determine whether there exists an edge-preserving mapping $\varphi : V(G) \to V(H)$, i.e., for every $uv \in E(G)$, it holds that $\varphi(u)\varphi(v) \in E(H)$. Observe that for K_k being a complete graph on k vertices, the Hom(K_k) problem is equivalent to k-Coloring. Observe also that the problem is trivial when H contains a vertex x with a loop since we can map all vertices of G to x. In case when H is bipartite, in fact we have to verify whether G is bipartite and this can be done in polynomial time. Hell and Nešetřil proved that for all other graphs H, i.e., loopless and non-bipartite, the Hom(H) problem is NP-hard [18]. Such a complete dichotomy was provided by Feder, Hell, and Huang also for the list version of the problem [13]. The graph homomorphism problem and its variants in various graph classes and under various parametrizations received recently a lot of attention [26, 15, 14, 4, 5, 7, 8, 14, 27, 17]. We also point out that among all target graphs H, odd cycles received a lot of attention [16, 1, 10, 31, 22]. Note that the cycle on 5 vertices is the smallest graph H such that the Hom(H) problem is not equivalent to graph coloring.

Our contribution. In this paper we consider the Hom(C_{2k+1}) problem on bounded-diameter graphs, where C_{2k+1} denotes the cycle on $2k+1$ vertices. Note that for $k = 1$, we have C_3, so this problem is equivalent to 3-Coloring. In this work we consider all other values of k. Our first result is the following.

▶ **Theorem 1.** *Let $k \geq 2$. Then Hom(C_{2k+1}) can be solved in polynomial time on diameter-$(k+1)$ graphs.*

Note that such a result for $k = 1$ would yield a polynomial-time algorithm for 3-Coloring on diameter-2 graphs. Let us discuss the crucial points where this algorithm cannot be applied directly for $k = 1$. The first property, which holds for every cycle except C_3 and C_6, is that if for some set S of vertices, any two of them have a common neighbor, then there is a vertex that is a common neighbor of all vertices of S. Furthermore, for every cycle of length at least 5 except C_6, for a set S of 3 vertices, every vertex of S has a private neighbor with respect to S, i.e., a neighbor that is non-adjacent to any other vertex of S.

Table 1 Complexity of $\textsc{Hom}(C_{2k+1})$ on bounded-diameter graphs. The symbol in the cell (k, d) denotes that $\textsc{Hom}(C_{2k+1})$ on diameter-d, resp., P – is polynomial-time solvable, NP-h – is NP-hard, S – can be solved in subexponential time, and NS – cannot be solved in subexponential time under the ETH. The rows for $k \geq 2$ are filled due to Theorems 1–3. The first row is based on [25, 9].

k / diam	1	2	3	4	5	6	7	8	9	≥ 10
1	P	S	S, NP-h	NS	NS	NS	NS	NS	NS	NS
2	P	P	P	S	S	NS	NS	NS	NS	NS
3	P	P	P	P	S	S	?	?	NS	NS
4	P	P	P	P	P	S	S	?	?	NS
≥ 5	P	P	P	P	P	P	S	S	?	?

We first show that for an instance of $\textsc{Hom}(C_{2k+1})$, for each vertex v we can deduce the set of vertices it can be mapped to and define a list of v – all lists are of size at most 3. The properties discussed above allow us to encode coloring of a vertex with list of size 3 using its neighbors with lists of size two, and such a reduced instance of a slightly more general problem (we have more constraints than just the edges, but all of them are binary) can be solved in polynomial time by reduction to 2-SAT, similar to the one of Edwards [11].

Furthermore, we give subexponential-time algorithms.

▶ **Theorem 2.** *Let $k \geq 2$. Then $\textsc{Hom}(C_{2k+1})$ can be solved in time:*

(1.) $2^{\mathcal{O}((n \log n)^{\frac{k+1}{k+2}})}$ *on diameter-$(k+2)$ n-vertex graphs,*

(2.) $2^{\mathcal{O}((n \log n)^{\frac{k+2}{k+3}})}$ *on diameter-$(k+3)$ n-vertex graphs.*

Here the branching part of the algorithm is rather standard. The more involved part is to show that after applying braching and reduction rules we are left with an instance that can be solved in polynomial time. Similar to Theorem 1, we first analyze the lists of all vertices, and then reduce to an instance of more general problem where all lists are of size at most 2.

We complement Theorem 1 and Theorem 2 with the following NP-hardness result – since our reduction from 3-SAT is linear, we also prove that the problem cannot be solved in subexponential time under the ETH. The summary of the results is presented in Table 1.

▶ **Theorem 3 (♠).** *Let $k \geq 2$. Then $\textsc{Hom}(C_{2k+1})$ is NP-hard on diameter-$(2k+2)$ graphs (of radius $k+1$) and cannot be solved in subexponential time, i.e., there is no algorithm solving every n-vertex instance G of $\textsc{Hom}(C_{2k+1})$ in time $2^{o(n)} \cdot n^{\mathcal{O}(1)}$, unless the ETH fails.*

The next direction we study in the paper is the following. Instead of considering larger odd cycles and apropriate diameter, we focus on diameter-2 input graphs, but we change the target graph to arbitrary H. Note that it only makes sense to consider graphs H of diameter-2, since the homomorphic image of a diameter-2 graph has to induce a diameter-2 subgraph. We consider triangle-free target graphs. We point out that the class of triangle-free diameter-2 graphs is still very rich, for instance, contains all Mycielski graphs.

▶ **Theorem 4.** *Let H be a triangle-free graph. Then the $\textsc{Hom}(H)$ problem can be solved in polynomial time on diameter-2 graphs.*

Finally, let us point out that we actually prove stronger statements of Theorem 1, Theorem 2, Theorem 4 as we consider more general list version of the problem.

The proofs of statements marked with ♠ can be found in the full version of the paper [29].

2 Preliminaries

For a vertex v, by $N_G(v)$ we denote the neighborhood of v in G, and for a set $U \subseteq V(G)$, we denote $N_G(U) := \bigcup_{u \in U} N_G(u) \setminus U$. By $\text{dist}_G(u,v)$ we denote the length (number of edges) of a shortest u-v path in G. For a positive integer d, by $N_G^{\leq d}(v)$ we denote the set of all vertices $u \in V(G)$ such that $\text{dist}_G(u,v) \leq d$. If G is clear from the contex, we omit the subscript G and simply write $N(v)$, $N(U)$, $N^{\leq d}(v)$, and $\text{dist}(u,v)$. A *diameter* of G, denoted by $\text{diam}(G)$, is the maximum $\text{dist}(u,v)$ over all pairs of vertices $u, v \in V(G)$. We say that G is *diameter-d* graph if $\text{diam}(G) \leq d$. A *radius* of G is the minimum integer r such that there is a vertex $z \in V(G)$ such that for every $v \in V(G)$, it holds that $\text{dist}(v,z) \leq r$. By $[n]$ we denote the set $\{1,2,\ldots,n\}$ and by $[n]_0$ we denote $\{0,1,\ldots,n\}$. Throughout this paper all graphs we consider are simple, i.e., no loops, no multiple edges.

Homomorphisms. For graphs G, H, a homomorphism from G to H is an edge-preserving mapping $\varphi : V(G) \to V(H)$, i.e., for every $uv \in E(G)$, it holds $\varphi(u)\varphi(v) \in E(H)$. For fixed H, called *target*, in the homomorphism problem, denoted by $\text{Hom}(H)$, we are given a graph G, and we have to determine whether there exists a homomorphism from G to H. In the list homomorphism problem, denoted by $\text{LHom}(H)$, G is given along with lists $L : V(G) \to 2^{V(H)}$, and we have to determine if there is a homomorphism φ from G to H which additionally respects lists, i.e., for every $v \in V(G)$ it holds $\varphi(v) \in L(v)$. We will write $\varphi : G \to H$ (resp. $\varphi : (G,L) \to H$) if φ is a (list) homomorphism from G to H, and $G \to H$ (resp. $(G,L) \to H$) to indicate that such a (list) homomorphism exists. Since the graph homomorphism problem generalizes graph coloring we will often refer to homomorphism as coloring and to vertices of H as colors. For an instance (G,L) and an induced subgraph G' of G while refering to a subinstance $(G', L|_{G'})$, we will often simply write (G', L).

Cycles. Whenever C_{2k+1} is the target graph, we will denote its vertex set by $[2k]_0$, unless stated explicitly otherwise. Moreover, whenever we refer to the vertices of the $(2k+1)$-cycle, i.e., cycle on $2k+1$ vertices, by $+$ and $-$ we denote respectively the addition and the subtraction modulo $2k+1$.

Lists. For an instance (G,L) of $\text{LHom}(C_{2k+1})$, by V_i we denote the set of vertices v of G such that $|L(v)| = i$. Sometimes we will refer to vertices of V_1 as *precolored vertices*. We also define $V_{\geq i} = \bigcup_{j \geq i} V_j$. We say that a list $L(v)$ is *of type* $(\ell_1, \ldots, \ell_r)$ if $|L(v)| = r+1$ and its vertices can be ordered $c_0, \ldots, c_r$ so that for every $i \in [r-1]_0$, we have that $c_{i+1} = c_i + \ell_i$. For example, for $k \geq 4$, one of the types of the list $\{1,4,6,7\}$ is $(3,2,1)$.

Binary CSP and 2-SAT. For a given set (domain) D, in the Binary Constraint Satisfaction problem (BCSP) we are given a set V of variables, list function $L : V \to 2^D$, and constraint function $C : V \times V \to 2^{D \times D}$. The task is to determine whether there exists an assignment $f : V \to D$ such that for every $v \in V$, we have $f(v) \in L(v)$ and for every pair $(u,v) \in V \times V$, we have $(f(u), f(v)) \in C(u,v)$. Clearly, any instance of $\text{LHom}(H)$ can be seen as an instance of BCSP, where the domain D is $V(H)$, list function remains the same, and for every edge $uv \in E(G)$ we set $C(u,v) = \{(x,y) \mid xy \in E(H)\}$ and for every $uv \notin E(G)$ we set $C(u,v) = V(H) \times V(H)$. We will denote by $\text{BCSP}(H,G,L)$ the instance of BCSP corresponding to the instance (G,L) of $\text{LHom}(H)$. Standard approach of Edwards [11] with a reduction to 2-SAT implies that in polynomial time we can solve an instance of BCSP with all list of size at most two.

▶ **Theorem 5** (Edwards [11]). *Let (V, L, C) be an instance of BCSP over the domain D. Assume that for every $v \in V$ it holds $|L(v)| \leq 2$. Then we can solve the instance (V, L, C) in polynomial time.*

3 Reduction rules and basic observations

In this section we define reduction rules and show some basic observations.

Reduction rules

Let H be a graph and let (G, L) be an instance of $\text{LHom}(H)$. We define the following reduction rules.

(R1) If $H = C_{2k+1}$ and G contains an odd cycle of length at most $2k-1$, then return NO.
(R2) If $H = C_{2k+1}$ and in G there are two $(2k+1)$-cycles with consecutive vertices respectively $c_0, \ldots, c_{2k}$ and $c'_0, \ldots, c'_{2k}$ and such that $c_0 = c'_0$ and $c_i = c'_j$ for some $i, j \neq 0$, then a) if $i = j$, then identify c_ℓ with c'_ℓ for every $\ell \in [2k]$, b) if $i = -j$, then identify c_ℓ with $c'_{(-\ell)}$ for every $\ell \in [2k]$, c) otherwise return NO.
(R3) For every edge uv, if there is $x \in L(u)$ such that $N_H(x) \cap L(v) = \emptyset$, then remove x from $L(u)$.
(R4) If there is $v \in V(G)$ such that $L(v) = \emptyset$, then return NO.
(R5) For every $v \in V(G)$, if there are $x, y \in L(v)$ such that for every $u \in N_G(v)$, we have $N_H(x) \cap L(u) \subseteq N_H(y) \cap L(u)$, then remove x from $L(v)$.
(R6) For a vertex $x \in V(H)$, and vertices $u, v \in V(G)$ such that $L(u) = L(v) = \{x\}$, if $uv \in E(G)$, then return NO, otherwise contract u with v.

Clearly, each of the above reduction rules can be applied in polynomial time. The following lemma shows that the reduction rules are safe.

▶ **Lemma 6.** *After applying each reduction rule to an instance (G, L) of $\text{LHom}(H)$, we obtain an equivalent instance with diameter at most* $\text{diam}(G)$.

Proof. First, any odd cycle cannot be mapped to a larger odd cycle, so the reduction rule (R1) is safe. Furthermore, for any $(2k+1)$-cycle, in any list homomorphism to C_{2k+1}, its consecutive vertices have to be mapped to consecutive vertices of C_{2k+1}. Let $c_0, \ldots, c_{2k}$ and $c'_0, \ldots, c'_{2k}$ be the vertices of two $(2k+1)$-cycles such that $c_0 = c'_0$ and $c_i = c'_j$ for some $i, j \neq 0$. Suppose we are dealing with a yes-instance and let $\varphi : (G, L) \to C_{2k+1}$. Without loss of generality assume that $\varphi(c_0) = \varphi(c'_0) = 0$ and $\varphi(c_i) = \varphi(c'_j) = i$. Then $\varphi(c_s) = s$ for every $s \in [2k]_0$. Moreover, either $j = i$ or $j = -i$, and $\varphi(c'_s) = s$ for every $s \in [2k]_0$ in the first case, or $\varphi(c'_s) = -s$ for every $s \in [2k]_0$ in the second case. Therefore, the reduction rule (R2) is safe.

Let $uv \in E(G)$ be such that there is $x \in L(u)$ such that $N(x) \cap L(v) = \emptyset$. Suppose that there is a list homomorphism $\varphi : (G, L) \to C_{2k+1}$ such that $\varphi(u) = x$. Then v must be mapped to a vertex from $N(x) \cap L(v) = \emptyset$, a contradiction. Thus we can safely remove x from $L(u)$, and (R3) is safe. Clearly, if any list of a vertex is an empty set, then we are dealing with a no-instance and thus (R4) is safe. Finally, assume there is $v \in V(G)$ and $x, y \in L(v)$ such that for every $u \in N_G(v)$, it holds $N_H(x) \cap L(u) \subseteq N_H(y) \cap L(u)$, and suppose there is a list homomorphism $\varphi : (G, L) \to H$ such that $\varphi(v) = x$. Then φ' defined so that $\varphi'(v) = y$ and $\varphi'(w) = \varphi(w)$ for $w \in V(G) \setminus \{v\}$ is also a list homomorphism $(G, L) \to H$. Therefore, (R5) is safe. If two vertices have the same one-element list, then they must be mapped to the same vertex. Since we only consider loopless graphs H, adjacent vertices of G cannot be mapped to the same vertex. Therefore, if two vertices u, v of G have lists $L(v) = L(u) = \{x\}$ for some $x \in V(H)$, then if $uv \in E(G)$ we are dealing with a no-instance. Otherwise, we can identify u and v and thus (R6) is safe. ◀

In the following lemma we describe the lists of vertices that are at some small distance of a precolored vertex.

▶ **Lemma 7.** *Let $k \geq 2$ and let (G, L) be an instance of* LHOM(C_{2k+1})*. Let $u \in V(G)$ be such that $L(u) = \{i\}$ and let $v \in V(G)$ be such that $\operatorname{dist}(u, v) = d$. If none of the reduction rules can be applied, then*
a) *$L(v) \subseteq \{i-d, i-d+2, \ldots, i-2, i, i+2, \ldots, i+d-2, i+d\}$ if d is even,*
b) *$L(v) \subseteq \{i-d, i-d+2, \ldots, i-1, i+1, \ldots, i+d-2, i+d\}$ if d is odd.*

Proof. Let P be a shortest u-v path such that the consecutive vertices of P are $u = p_0, p_1, \ldots, p_d = v$. We have $L(p_0) = \{i\}$. Since the reduction rule (R3) cannot be applied for p_0p_1, we must have $L(p_1) \subseteq \{i-1, i+1\}$. Applying this reasoning to consecutive vertices of the path, for $j \in [d]$, we must have

$$L(v) \subseteq \{i-j, i-j+2, \ldots, i-2, i, i+2, \ldots, i+j-2, i+j\},$$

if j is even,

$$L(v) \subseteq \{i-j, i-j+2, \ldots, i-1, i+1, \ldots, i+j-2, i+j\},$$

if j is odd, which completes the proof. ◀

The next lemma immediately follows from Lemma 7.

▶ **Lemma 8.** *Let $k \geq 2$ and let (G, L) be an instance of* LHOM(C_{2k+1})*. Let $u, w \in V(G)$ be such that $L(u) = \{i\}$ and $L(w) = \{i+1\}$. Let $v \in V(G)$ be such that $\operatorname{dist}(u, v) = \operatorname{dist}(w, v) = k + \ell$. If none of the reduction rules can be applied, then $L(v) \subseteq \{i+k-\ell+1, i+k-\ell+2, \ldots, i+k+\ell+1\}$.*

In the following lemma we show that for a partial mapping $\varphi : V(G) \to [2k]_0$ for $k \geq 2$, for $v \in V(G)$, if every pair (a, b) of its neighbors is precolored so that $\varphi(a)$ and $\varphi(b)$ have a common neighbor in $L(v)$, then φ can be extended to v so that it preserves the edges containing v.

▶ **Lemma 9.** *Let $k \geq 2$, let (G, L) be an instance of* LHOM(C_{2k+1}) *and let $v \in V(G)$. Let $\varphi : N(v) \to [2k]_0$ be a mapping such that for every $a, b \in N(v)$, we have that $N_{C_{2k+1}}(\varphi(a)) \cap N_{C_{2k+1}}(\varphi(b)) \cap L(v) \neq \emptyset$. Then $\bigcap_{u \in N(v)} N_{C_{2k+1}}(\varphi(u)) \cap L(v) \neq \emptyset$.*

Proof. Define $A = \{\varphi(u) \mid u \in N(v)\}$. If $|A| \leq 2$, then the statement clearly follows. We will show that this is the only case. So suppose that $|A| \geq 3$. If two distinct vertices of C_{2k+1} for $k \geq 2$ have a common neighbor, then they must be at distance exactly two. Without loss of generality, let $0, 2 \in A$ and $1 \in L(v)$. Moreover, let $i \in A \setminus \{0, 2\}$. By assumption $N_{C_{2k+1}}(i) \cap N_{C_{2k+1}}(0) \neq \emptyset$, so $i = 2k-1$. On the other hand, $N_{C_{2k+1}}(i) \cap N_{C_{2k+1}}(2) \neq \emptyset$, so $i = 4$. Thus $2k-1 = 4$, a contradiction. ◀

In the next lemma we show that for an odd cycle C and a vertex v there is at least one pair of consecutive vertices of C with equal distances to v.

▶ **Lemma 10.** *Let G be a connected graph, let C be a cycle in G with consecutive vertices $c_0, \ldots, c_{2k}$, and let $v \in V(G) \setminus V(C)$. Then there is $i \in [2k]_0$ such that $\operatorname{dist}(v, c_i) = \operatorname{dist}(v, c_{i+1})$.*

Proof. First observe, that for all i, we have $|\operatorname{dist}(v, c_i) - \operatorname{dist}(v, c_{i+1})| \leq 1$ since $c_i c_{i+1} \in E(G)$. Therefore, going around the cycle the distance from v to c_i can increase by 1, decrease by 1, or remain the same. Since we have to end up with the same value at the end and the length of the cycle is odd, there is at least one pair of consecutive vertices c_i, c_{i+1} such that $\operatorname{dist}(v, c_i) = \operatorname{dist}(v, c_{i+1})$. ◀

4 Polynomial-time algorithm

In this section we prove Theorem 1. In fact we prove a stronger statement for the list version of the problem.

▶ **Theorem 11.** *Let $k \geq 2$. Then* LHOM(C_{2k+1}) *can be solved in polynomial time on diameter-$(k+1)$ graphs.*

Proof. Let (G, L) be an instance of LHOM(C_{2k+1}) such that G has diameter at most $k+1$. First for every $i \in [2k]_0$ we check whether there is a list homomorphism $\varphi : (G, L) \to C_{2k+1}$ such that no vertex is mapped to i, so we look for a list homomorphism to a path which can be done in polynomial time by [12].

If there is no such a list homomorphism, then we know that all colors have to be used and thus we guess $2k+1$ vertices that will be mapped to distinct vertices of C_{2k+1}. Let $c_0, \ldots, c_{2k}$ be the vertices such that c_i is precolored with i. We check whether such a partial assignment satisfies the edges with both endpoints precolored. Moreover, if c_i, c_{i+1} are non-adjacent, we add the edge $c_i c_{i+1}$ – this operation is safe as c_i, c_{i+1} are precolored with consecutive vertices of C_{2k+1} and adding an edge does not increase the diameter. Finally, we exhaustively apply the reduction rules.

Observe that the vertices $c_0, \ldots, c_{2k}$ induce a $(2k+1)$-cycle C. Suppose there is a vertex v that after the above procedure is not on C. By Lemma 10, there is $i \in [2k]_0$ such that $\operatorname{dist}(v, c_i) = \operatorname{dist}(v, c_{i+1}) =: \ell$.

First we show that we cannot have $\ell \leq k$. Suppose otherwise. Let P_1, P_2 be shortest v-c_i-, and v-c_{i+1}-paths, respectively. Let u be the their last common vertex (it cannot be c_i or c_{i+1} as the distances are the same and c_i, c_{i+1} are adjacent). Note that since P_1, P_2 are shortest, the u-c_i-path P_1' obtained from P_1 and the u-c_{i+1}-path P_2' obtained from P_2 have the same length. Therefore we can construct a cycle by taking P_1', P_2' and the edge $c_i c_{i+1}$. The length of the cycle is odd, and it is at most $2k+1$. This cycle contains at least two vertices from C, so either it should be contracted by (R2) or (R1) would return NO. Therefore we cannot have $\ell \leq k$, and thus, since $\operatorname{diam}(G) \leq k+1$, we have $\ell = k+1$.

By Lemma 8, for $v \in V_{\geq 3}$, we have that $L(v) \subseteq \{i+k, i+k+1, i+k+2\}$. Therefore, all lists of our instance have size at most 3. Moreover, each vertex of V_3 has list of type $(1,1)$. Furthermore, since (R3) cannot be applied, for a vertex with list $\{j, j+1, j+2\}$, the possible lists of its neighbors in $G[V_3]$ are then $\{j-1, j, j+1\}$, $\{j, j+1, j+2\}$, and $\{j+1, j+2, j+3\}$.

For a list $\{i, j, r\}$ of type $(1,1)$, where j is the vertex such that $j = i+1$ and $j = r-1$, we will call j the *middle vertex* of $\{i, j, r\}$. For a homomorphism φ we will say that a vertex $v \in V_3$ is *φ-middle*, if φ maps v to the middle vertex of its list.

Now consider a connected component S of $G[V_3]$, let $v \in V(S)$, and let $L(v) = \{j-1, j, j+1\}$. The following claim is straightforward.

▷ Claim 12. Suppose there is a list homomorphism $\varphi : (S, L) \to C_{2k+1}$. Then

(1.) if v is φ-middle, then any $u \in N_S(v)$ with list $\{j-1, j, j+1\}$ cannot be φ-middle, and every $w \in N_S(v)$ with list $\{j-2, j-1, j\}$ or $\{j, j+1, j+2\}$ has to be φ-middle,

(2.) if v is not φ-middle, then every $u \in N_S(v)$ with list $\{j-1, j, j+1\}$ has to be φ-middle, and any $w \in N_S(v)$ with list $\{j-2, j-1, j\}$ or $\{j, j+1, j+2\}$ cannot be φ-middle.

Thus deciding if one vertex of S is φ-middle, already determines for every vertex of S if it is φ-middle or not. It is described more formally in the following claim, whose proof can be found in the full version of the paper [29].

▷ Claim 13 (♠). In polynomial time we can either (1) construct a partition (U_1, U_2) of $V(S)$ (U_1, U_2 might be empty) such that for every list homomorphism $\varphi : (S, L) \to C_{2k+1}$, either all vertices of U_1 are φ-middle and no vertex of U_2 is φ-middle, or all vertices of U_2 are φ-middle and no vertex of U_1 is φ-middle, or (2) conclude that we are dealing with a no-instance.

Therefore, for every connected component S of $G[V_3]$ we solve two subinstances:

(I1) (S, L_1), where for every $v \in V(S)$ with list $\{i-1, i, i+1\}$ for some $i \in [2k]_0$, $L_1(v) = \{i\}$ if $v \in U_1$ and $L_1(v) = \{i-1, i+1\}$ if $v \in U_2$,

(I2) (S, L_2), where for every $v \in V(S)$ with list $\{i-1, i, i+1\}$ for some $i \in [2k]_0$, $L_2(v) = \{i\}$ if $v \in U_2$ and $L_2(v) = \{i-1, i+1\}$ if $v \in U_1$.

Note that both subinstances have all lists of size at most two and thus can be solved in polynomial time by Theorem 5. If for some component in both cases we obtain NO, then we return NO.

Creating a BCSP instance. Let $(V(G), L, C) = \text{BCSP}(C_{2k+1}, G[V_1 \cup V_2], L)$. We will modify now the instance $(V(G), L, C)$ so it is equivalent to the instance (G, L). For every $v \in V_3$ and for every pair of $a, b \in N(v) \cap V_2$, we leave in $C(a, b)$ only these pairs of vertices that have a common neighbor in $L(v)$ – recall that by Lemma 9 this ensures us that there will be color left for v. Furthermore, for every connected component S of $G[V_3]$, we add constraints according to which of the two possibilities S can be properly colored (possibly S can be colored in both cases) as follows. Let $v \in V(S)$ with $L(v) = \{i-1, i, i+1\}$, and without loss of generality assume that $v \in U_1$. The neighbors of v in V_2 have lists $\{i-1, i\}$ and $\{i, i+1\}$. For each such v:

- if S cannot be properly colored so that vertices of U_1 are middle (v is colored with i), we remove $i-1$ and $i+1$ from the lists of neigbors of v,
- if S cannot be properly colored so that vertices of U_1 are not middle (v is colored with one of $i-1, i+1$), we remove i from the lists of neigbors of v.
- Moreover, for every $u \in V(S)$ with list $\{j-1, j, j+1\}$, for every neighbor $u' \in V_2 \cap N(u)$, and for every $v' \in V_2 \cap N(v)$, if $u \in U_1$, then we remove from $C(u', v')$ pairs $(j, i+1)$, $(j, i-1)$, $(j-1, i)$, $(j+1, i)$, and if $u \in U_2$, then we remove from $C(u', v')$ the pairs (j, i), $(j-1, i-1)$, $(j-1, i+1)$, $(j+1, i-1)$, $(j+1, i+1)$.

This completes the construction of BCSP instance $(V(G), L, C)$. By Theorem 5 we solve $(V(G), L, C)$ in polynomial time.

Correctness. The detailed proof of correctness can be found in the full version of the paper [29].

▷ Claim 14 (♠). $(V(G), L, C)$ is a yes-instance of BCSP iff (G, L) is a yes-instance of LHom(C_{2k+1}).

Let us only mention here the idea behind each of introduced constraints. First, we start with $\text{BCSP}(C_{2k+1}, G[V_1 \cup V_2], L)$, so that the assignment restricted to $V_1 \cup V_2$ can be a list homomorphism φ on $(G[V_1 \cup V_2], L)$. The remaining constraints ensure us that we can extend φ to each connected component S of $G[V_3]$. First, we make sure that for every vertex $v \in V(S)$, there is a color left for v in $L(v)$ – it is a necessary condition. Observe that for a vertex v with list $\{i-1, i, i+1\}$, if the neighbors of v in V_2 are mapped so that there is a color left for v, then either they are colored with $i-1, i+1$ and the color left for v is i, or all such neighbors are colored with i, and both $i-1, i+1$ are left for v (unless one of them is not on $L(v)$). Therefore, we can assume that v has the same list as in one of the instances (I1) and (I2). Moreover, because of the last introduced constraint, all vertices of S have lists corresponding to the same instance, say (I1). Finally, if these lists are still present, then we know that (I1) is a yes-instance, and φ can be extended to S. This completes the proof. ◀

5 Subexponential-time algorithms

In this section we prove the following stronger version of Theorem 2.

▶ **Theorem 15.** *Let $k \geq 3$. Then* LHom(C_{2k+1}) *can be solved in time:*

(1.) $2^{\mathcal{O}((n\log n)^{\frac{k+1}{k+2}})}$ *on n-vertex diameter-$(k+2)$ graphs,*

(2.) $2^{\mathcal{O}((n\log n)^{\frac{k+2}{k+3}})}$ *on n-vertex diameter-$(k+3)$ graphs.*

We start with defining branching rules crucial for our algorithm.

Branching rules. Let $k \geq 2$, let (G, L) be an instance of LHom(C_{2k+1}) and let $d \geq \operatorname{diam}(G)$. Let $\mu = \sum_{\ell=2}^{2k+1} \ell \cdot |V_\ell|$. We define the following branching rules.

(B1) If there is a vertex $v \in V_{\geq 2}$ with at least $(\mu \log \mu)^{1/d}$ neighbors in $V_{\geq 2}$, for some $a \in L(v)$, we branch on coloring v with a or not, i.e., we create two instances $I_a = (G, L_a)$, $I'_a = (G, L'_a)$ such that $L_a(u) = L'_a(u) = L(u)$ for every $u \in V(G) \setminus \{v\}$, and $L_a(v) = \{a\}$ and $L'_a(v) = L(v) \setminus \{a\}$.

(B2) We pick a vertex v and branch on the coloring of $N^{\leq d-1}[v] \cap V_{\geq 2}$, i.e., for every mapping f of $N^{\leq d-1}[v] \cap V_{\geq 2}$ that respects the lists, we create a new instance $I_f = (G, L_f)$ such that $L_f(u) = L(u)$ for $u \notin N^{\leq d-1}[v] \cap V_{\geq 2}$ and $L_f(w) = \{f(w)\}$ for $w \in N^{\leq d-1}[v] \cap V_{\geq 2}$.

Algorithm `Recursion Tree`

Let us describe an algorithm that for fixed d takes an instance (G, L) of LHom(C_{2k+1}) with a fixed precolored $(2k+1)$-cycle C and such that $\operatorname{diam}(G) \leq d$, and returns a rooted tree $\mathcal{R}$ whose nodes are labelled with subinstances of (G, L). We first introduce the root r of $\mathcal{R}$ and we label it with (G, L). Then for every node we proceed recursively as follows. Let s be a node labelled with an instance (G', L') of LHom(C_{2k+1}). We first exhaustively apply to (G', L') reduction rules and if some of the reduction rules returns NO, then s does not have any children. Otherwise, if possible, we apply branching rule (B1). We choose $a \in L(v)$ for (B1) as follows. If on $N_{G'[V_{\geq 2}]}(v)$ there are no lists of type (2), then we take any $a \in L(v)$. Otherwise, let S be the most frequent list of type (2) on $N_{G'[V_{\geq 2}]}(v)$, and let $S = \{j-1, j+1\}$ for some $j \in [2k]_0$. Then we take any $a \in L(v) \setminus \{j\}$. After application of (B1), we exhaustively apply reduction rules to each instance. Furthermore, for each instance created by (B1), we create a child node of s and we label it with that instance.

So suppose that (B1) cannot be applied. We proceed as follows.

Case 1: there is no vertex v such that $\operatorname{dist}(v, c_i) = \operatorname{dist}(v, c_{i+1}) = k+3$ for some $i \in [2k]_0$. Then we apply the branching rule (B2) once, we exhaustively apply reduction rules, and again for each instance created by (B2), we introduce a child node of s. The choice of v is not completely arbitrary. If possible, we choose v so that $\operatorname{dist}(v, C) \geq \lceil \frac{d}{2} \rceil$ – note that the cycle C is present in all instances of $\mathcal{R}$.

Case 2: there exists at least one vertex v such that $\mathrm{dist}(v, c_i) = \mathrm{dist}(v, c_{i+1}) = k + 3$ for some $i \in [2k]_0$. For every $i \in [2k]_0$, we choose (if exists) a vertex v such that $\mathrm{dist}(v, c_i) = \mathrm{dist}(v, c_{i+1}) = k+3$, and again, if possible, we choose v so that $\mathrm{dist}(v, C) \geq \lceil \frac{d}{2} \rceil$. For each such v, we apply the branching rule (B2) and exhaustively reduction rules, and the children of s are those introduced for all instances created in all applications of (B2).

In both cases, we do not recurse on the children of s for which we applied (B2). Let us analyze the running time of `Recursion Tree` and properties of the tree $\mathcal{R}$.

▶ **Lemma 16.** *Given an instance (G, L) of* LHom(C_{2k+1}) *with a fixed precolored $(2k+1)$-cycle C and such that $n = |V(G)|$, $\mathrm{diam}(G) \leq d$, the algorithm `Recursion Tree` in time $2^{\mathcal{O}((n \log n)^{\frac{d-1}{d}})}$ returns a tree $\mathcal{R}$ whose nodes are labelled with instances of* LHom(C_{2k+1}) *and (G, L) is a yes-instance if and only if at least one instance corresponding to a leaf of $\mathcal{R}$ is a yes-instance.*

Proof. First we show that for every node s of $\mathcal{R}$ the corresponding instance is a yes-instance if and only if at least one instance corresponding to a child of s is a yes-instance. Let s be a node of $\mathcal{R}$ and let (G', L') be the corresponding instance. The algorithm `Recursion Tree` applies first reduction rules to (G', L') and by Lemma 6, we obtain equivalent instance. Furthermore, we applied to (G', L') either (B1) or (B2) where the branches correspond to all possible colorings of some set of vertices so indeed (G', L') is a yes-instance if and only if at least one instance corresponding to a child of s is a yes-instance. Since the root of $\mathcal{R}$ is labelled with (G, L), we conclude that (G, L) is a yes-instance if and only if at least one instance corresponding to a leaf of $\mathcal{R}$ is a yes-instance.

It remains to analyze the running time. Let $F(\mu)$ be the upper bound on the running time of `Recursion Tree` applied to an instance (G', L') with $\mu = \sum_{\ell=2}^{2k+1} \ell \cdot |V_\ell|$. Let $p(n)$ be a polynomial such that the reduction rules can be exhaustively applied to an instance on n vertices in time $p(n)$ – note that each reduction rule either decreases the number of vertices/sizes of lists or returns NO, so indeed exhaustive application of reduction rules can be performed in polynomial time. Observe that if we apply (B1) to (G', L'), then

$$F(\mu) \leq F\left(\mu - \frac{(\mu \log \mu)^{1/d}}{2k+1}\right) + F(\mu - 1) + 2 \cdot p(n).$$

Indeed, let v be the vertex to which we apply (B1). If there are no lists of type (2) on $N_{G'[V_{\geq 2}]}(v)$, then in the branch where we set $L(v) = \{a\}$, after application of reduction rules, every neighbor of v must have $L(v) \subseteq \{a-1, a+2\}$. If $|L(v)| \geq 2$ and $L(v) \neq \{a-1, a+1\}$, then $|L(v)| \cap \{a-1, a+1\}| < |L(v)|$. Therefore, in this case we decrease sizes of all lists on $N_{G'[V_{\geq 2}]}(v)$. Otherwise, we chose $a \in L(v) \setminus \{j\}$, where $\{j-1, j+1\}$ is the most frequent list of type (2) on $N_{G'[V_{\geq 2}]}(v)$. Since there are exactly $2k+1$ lists of type (2), at least $\frac{1}{2k+1}$-fraction of $N_{G'[V_{\geq 2}]}(v)$ has list of different type than (2) or has list $\{j-1, j+1\}$. Thus, for the branch where we set $L(v) = \{a\}$, the sizes of lists of at least $\frac{1}{2k+1} \cdot (\mu \log \mu)^{1/d}$ vertices decrease. In the branch where we remove a from $L(v)$, we decrease the size of $L(v)$ at least by one. In both branches we apply the reduction rules, so the desired inequality follows.

If we apply (B2) to (G', L') – recall that we stop recursing in this case – and if we are in Case 1, then we obtain

$$F(\mu) \leq (2k+1)^{(\mu \log \mu)^{\frac{d-1}{d}}} \cdot p(n),$$

since we guess the coloring on $N^{\leq d-1}_{G'[V_{\geq 2}]}(v)$ whose size is bounded by $(\mu \log \mu)^{\frac{d-1}{d}}$ (in this case we could not apply (B1) so the degrees in $G'[V_{\geq 2}]$ are bounded by $(\mu \log \mu)^{1/d}$) and the number of possible colors is at most $2k+1$.

In Case 2, we have:

$$F(\mu) \leq (2k+1)^{(2k+1)\cdot(\mu \log \mu)^{\frac{d-1}{d}}} \cdot p(n),$$

where additional $(2k+1)$ in the exponent comes from the fact that we applied (B2) possibly $(2k+1)$ times.

We can conclude that $F(\mu) \leq 2^{\mathcal{O}((\mu \log \mu)^{\frac{d-1}{d}})}$ (see for example [9], proof of Theorem 7) which combined with the inequality $\mu \leq (2k+1)n = \mathcal{O}(n)$ completes the proof. ◀

The following lemma shows that we can solve every instance corresponding to a leaf of $\mathcal{R}$ in polynomial time. We only sketch the proof here – for the full proof see [29].

▶ **Lemma 17** (♠). *Let (G', L') be an instance of* LHom(C_{2k+1}) *such that* $\operatorname{diam}(G') \leq k+3$ *and let C be a fixed precolored $(2k+1)$-cycle. Assume that we applied algorithm* `Recursion Tree` *to (G', L') and let $\mathcal{R}$ be the resulting recursion tree. Let (G, L) be as instance corresponding to a leaf in $\mathcal{R}$. Then (G, L) can be solved in polynomial time.*

Sketch of proof. By Lemma 10, for every vertex u outside the cycle C, there must be $j \in [2k]_0$ such that $\operatorname{dist}(u, c_j) = \operatorname{dist}(u, c_{j+1})$. Since the reduction rules (R1), (R2) cannot be applied and $\operatorname{diam}(G) \leq k+3$, then that distance is either $k+1$, $k+2$, or $k+3$. In case of $k+1$ or $k+2$ by Lemma 8 have that $L(u) \subseteq \{i-2, i-1, i, i+1, i+2\}$ for some $i \in [2k]_0$, and in case of $k+3$ we have $L(u) \subseteq \{i-3, i-2, i-1, i, i+1, i+2, i+3\}$ for some $i \in [2k]_0$.

Moreover, since for (B2), if we could, we chose vertex v whose distance from C is at least $\lceil \frac{d}{2} \rceil$, each vertex of $V_{\geq 3}$ is at distance $\lfloor \frac{d}{2} \rfloor$ from C. Indeed, every vertex u' that was in $N^{\leq d}_{G[V \geq 2]}(v)$ has list of size at most 2, as we guessed a color either for u' or at least one of its neighbors. So for any vertex u left in $V_{\geq 3}$, the shortest u-v path (whose length is at most the diameter d) should contain a vertex from C and length of that path is at least $\operatorname{dist}(v, C) + \operatorname{dist}(u, C)$. So either all vertices outside C were at distance at most $\lfloor \frac{d}{2} \rfloor$ or v was at distance at least $\lceil \frac{d}{2} \rceil$ and thus u is at distance at most $\lfloor \frac{k+3}{2} \rfloor$ from C. If $k > 3$, then $\lfloor \frac{k+3}{2} \rfloor < k$, which by Lemma 7 implies that $L(u)$ is an independent set. Combining the facts, for $u \in V_{\geq 3}$, diam $\leq k+2$, and $k > 3$, we obtain $L(u) = \{i-2, i, i+2\}$. By careful analysis we can prove that the same holds in the remaining cases (♠).

Furthermore, observe that if v is a neighbor of u with list $\{i-2, i, i+2\}$ and the reduction rule (R3) cannot be applied, then the possible list of v is $\{i-1, i+1\}$, $\{i-3, i+1\}$, $\{i-1, i+3\}$, $\{i-3, i-1, i+1\}$, or $\{i-1, i+1, i+3\}$. If for some vertex u with list $\{i-2, i, i+2\}$, some of possible lists is not present on $N(u)$, then we add v with such a list to G and make it adjacent to u. Note that now the diameter of G might increase, but we will not care about the diameter anymore. Moreover, any list homomorphism on $G - v$ can be extended to v, whose degree is 1. So since now, we can assume that for u with list $\{i-2, i, i+2\}$, all lists $\{i-1, i+1\}$, $\{i-3, i+1\}$, $\{i-1, i+3\}$ are present on $N(u)$ – this will allow us to encode coloring of u on its neighbors with lists of size 2.

BCSP instance. We start with $\operatorname{BCSP}(C_{2k+1}, G - V_3, L)$. Then, for every vertex $v \in V_3$ and for every $v', v'' \in V_2 \cap N(v)$, we leave in $C(v', v'')$ only such pairs that have a common neighbor in $L(v)$. Furthermore, for every edge uv with $u, v \in V_3$ and lists $L(u) = \{i-2, i, i+2\}$, $L(v) = \{i-1, i+1, i+3\}$, and for every pair $u', v' \in V_2$ such that $uu', vv' \in E(G)$, we remove (if they are present) from $C(u', v')$ the following pairs: $(i-3, i+2)$, $(i-1, i+4)$, and $(i+3, i-2)$. This completes the construction of (V, L, C), which is equivalent to (G, L) (♠).

Clearly (V, L, C) is constructed in polynomial time. Moreover, since all the lists have size at most 2, by Theorem 5, (V, L, C) can be solved in polynomial time, which completes the proof. ◀

Now we are ready to prove Theorem 15.

Proof of Theorem 15. Let (G, L) be an instance of LHOM(C_{2k+1}) such that diam(G) is at most $d \in \{k+2, k+3\}$. As in Theorem 1, first for every $i \in [2k]_0$, we check in polynomial time whether there is a list homomorphism $\varphi : (G, L) \to C_{2k+1}$ such that no vertex is mapped to i – this can be done by [12]. If there is no such list homomorphism, we guess $2k+1$ vertices $c_0, \ldots, c_{2k}$ which will be colored so that c_i is mapped to i. We add the edges $c_i c_{i+1}$ and we obtain an induced $(2k+1)$-cycle C (if not, then we are dealing with a no-instance). Note that adding edges cannot increase the diameter and since the edges are added between vertices precolored with consecutive vertices, we obtain an equivalent instance.

Now for (G, L) and C as the fixed precolored $(2k+1)$-cycle we use the algorithm `Recursion Tree`, which by Lemma 16 in time $2^{\mathcal{O}((n \log n)^{\frac{d-1}{d}})}$ returns a tree $\mathcal{R}$. Moreover, in order to solve the instance (G, L) it is enough to solve every instance corresponding to a leaf of $\mathcal{R}$ by Lemma 16, and by Lemma 17, we can solve each such instance in polynomial time. Furthermore, since the size of $\mathcal{R}$ is bounded by the running time, the instance (G, L) can be solved in time $2^{\mathcal{O}((n \log n)^{\frac{d-1}{d}})} \cdot n^{\mathcal{O}(1)} = 2^{\mathcal{O}((n \log n)^{\frac{d-1}{d}})}$, which completes the proof. ◀

6 Beyond odd cycles

In this section we consider target graphs other than odd cycles. Instead, we focus on input graphs with diameter at most 2. Since homomorphisms preserve edges, for graphs G, H, a homomorphism $\varphi : G \to H$ and a sequence of vertices $v_1, \ldots, v_k$ forming a path in G, the sequence $\varphi(v_1), \ldots, \varphi(v_k)$ forms a walk in H. Therefore, if G has diameter at most 2, we can assume that H has also diameter at most 2. The following observation is straightforward.

▶ **Observation 18.** *Let G, H be graphs such that G is connected. If there exists a homomorphism $\varphi : G \to H$, then the image $\varphi(V(G))$ induces in H a subgraph with diameter at most* diam(G)*.*

We prove the following stronger version of Theorem 4.

▶ **Theorem 19.** *Let H be a simple triangle-free graph. Then LHOM(H) is polynomial-time solvable on diameter-2 graphs.*

Proof. Let (G, L) be an instance of LHOM(H). We guess the set of colors that will be used – by Observation 18 they should induce a diameter-2 subgraph H' of H. For each such H', we guess $h' = |H'|$ vertices $v_1, \ldots, v_{h'}$ of G that will be injectively mapped to $V(H') = \{x_1, \ldots, x_{h'}\}$. For each tuple $(H', v_1, \ldots, v_{h'})$, we solve the instance (G, L') of LHOM(H'), where $L'(v) = \{x_i\}$ for $v = v_i$, $i \in [h']$ and $L'(v) = L(v)$ otherwise. Note that (G, L) is a yes-instance if and only if at least one instance (G, L') is a yes-instance.

First for every edge $x_i x_j \in E(H')$, if v_i, v_j are non-adjacent, we add the edge $v_i v_j$ to G – note that this operation is safe, since we cannot increase the diameter by adding edges and we only add edges between vertices that must be mapped to neighbors in H'. Therefore, we can assume that the set $V' = \{v_1, \ldots, v_{h'}\}$ induces a copy of H' in G (if not, then we have an extra edge, which means that we are dealing with a no-instance and we reject immediately). Furthermore, we exhaustively apply reduction rules.

So from now on we assume that the instance (G, L') is reduced. We claim that either (G, L') is a no-instance or $V(G) = \{v_1, \ldots, v_{h'}\}$, i.e., after exhaustive application of the reduction rules the graph G is isomorphic to H'. Note that in the latter case we can return YES as an answer.

Suppose there is $v \in V(G) \setminus V'$. Moreover, we choose such v which is adjacent to some vertex of V' (see Figure 1). Suppose that there exists $\varphi : (G, L') \to H'$ and let $x_i = \varphi(v)$. Then v cannot be adjacent to v_i since there are no loops in H'. Furthermore, the only neighbors of v in V' can be the neighbors of v_i. Suppose that there is $v_j \in N_G(v_i) \cap V'$ which is non-adjacent to v. Since the diameter of G is at most 2, then there must be $u \in N_G(v) \cap N_G(v_j)$. Observe that $u \notin V'$. Indeed, v does not have any neighbors in $V' \setminus N_G(v_i)$ and if $u \in N_G(v)$, then there is a triangle uv_iv_j in a copy of H', a contradiction. Furthermore, it must hold that $\varphi(u)$ is adjacent to x_i in H' as u is adjacent to v and $\varphi(v) = x_i$, and similarly, $\varphi(u)$ must be adjacent to x_j as u is adjacent to v_j. Then $\varphi(u)x_ix_j$ forms a triangle in H', a contradiction. Thus v must be adjacent to all vertices of $N(v_i) \cap V'$.

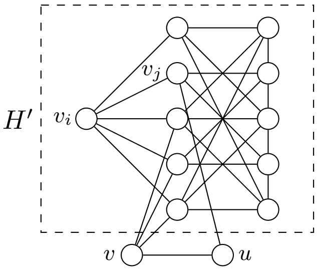


Figure 1 Copy of H' in G and a vertex v such that for some homomorphism φ, it holds $\varphi(v_i) = \varphi(v)$. We show that a vertex u which is a common neighbor of v and some neighbor v_j of v_i in the copy of H' cannot exist.

Since (R3) cannot be applied, each vertex of $L'(v)$ is adjacent to all vertices of $N_H(x_i)$. Moreover, since (R5) cannot be applied, it holds that $L'(v) = \{x_i\}$. Indeed, otherwise there is $x_{i'} \neq x_i$ such that $x_{i'} \in L'(v)$. Recall that $x_{i'}$ is adjacent to all vertices of $N_H(x_i)$. Therefore, $N_H(x_i) \subseteq N_H(x_{i'})$, and thus one of $x_i, x_{i'}$ should have been removed from $L'(v)$ by (R5). Furthermore, since (R6) cannot be applied, we must have $v = v_i \in V'$, a contradiction. This completes the proof. ◀

7 Conclusion

In this paper we studied the computational complexity of Hom(C_{2k+1}) problem on bounded-diameter graphs. We proved that for $k \geq 2$, the Hom(C_{2k+1}) problem can be solved in polynomial-time on diameter-$(k+1)$ graphs and we gave subexponential-time algorithms for diameter-$(k+2)$ and -$(k+3)$ graphs. We also proved that Hom(H) for triangle-free graph H, can be solved in polynomial time on diameter-2 graphs.

The main open problem in this area remains the question whether 3-Coloring on diameter-2 graphs can be solved in polynomial time. However, as more reachable, we propose the following future research directions. (i) Is the Hom(C_{2k+1}) problem NP-hard for the subexponential-time cases, i.e., diameter-$(k+2)$ or -$(k+3)$ graphs? (2) Let H be a diameter-2 graph such that H contains a triangle but $H \not\to K_3$. Is the Hom(H) problem NP-hard?

References

1 Laurent Beaudou, Florent Foucaud, and Reza Naserasr. Smallest c_{2l+1}-critical graphs of odd-girth 2k+1. *Discret. Appl. Math.*, 319:564–575, 2022. doi:10.1016/J.DAM.2021.08.040.

2 Marthe Bonamy, Konrad K. Dabrowski, Carl Feghali, Matthew Johnson, and Daniël Paulusma. Independent feedback vertex sets for graphs of bounded diameter. *Inf. Process. Lett.*, 131:26–32, 2018. doi:10.1016/J.IPL.2017.11.004.

3 Christoph Brause, Petr A. Golovach, Barnaby Martin, Pascal Ochem, Daniël Paulusma, and Siani Smith. Acyclic, star, and injective colouring: Bounding the diameter. *Electron. J. Comb.*, 29(2), 2022. doi:10.37236/10738.

4 Andrei A. Bulatov and Amirhossein Kazeminia. Complexity classification of counting graph homomorphisms modulo a prime number. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 – 24, 2022*, pages 1024–1037. ACM, 2022. doi:10.1145/3519935.3520075.

5 Jin-Yi Cai and Ashwin Maran. The complexity of counting planar graph homomorphisms of domain size 3. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1285–1297. ACM, 2023. doi:10.1145/3564246.3585173.

6 Victor A. Campos, Guilherme de C. M. Gomes, Allen Ibiapina, Raul Lopes, Ignasi Sau, and Ana Silva. Coloring problems on bipartite graphs of small diameter. *Electron. J. Comb.*, 28(2):2, 2021. doi:10.37236/9931.

7 Rajesh Chitnis, László Egri, and Dániel Marx. List h-coloring a graph by removing few vertices. *Algorithmica*, 78(1):110–146, 2017. doi:10.1007/S00453-016-0139-6.

8 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. 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 210–223. ACM, 2017. doi:10.1145/3055399.3055502.

9 Michal Dębski, Marta Piecyk, and Paweł Rzążewski. Faster 3-coloring of small-diameter graphs. *SIAM J. Discret. Math.*, 36(3):2205–2224, 2022. doi:10.1137/21M1447714.

10 Oliver Ebsen and Mathias Schacht. Homomorphism thresholds for odd cycles. *Comb.*, 40(1):39–62, 2020. doi:10.1007/S00493-019-3920-8.

11 Keith Edwards. The complexity of colouring problems on dense graphs. *Theor. Comput. Sci.*, 43:337–343, 1986. doi:10.1016/0304-3975(86)90184-2.

12 Tomás Feder, Pavol Hell, and Jing Huang. List homomorphisms and circular arc graphs. *Comb.*, 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. *J. Graph Theory*, 42(1):61–80, 2003. doi:10.1002/JGT.10073.

14 Jacob Focke, Dániel Marx, and Paweł Rzążewski. Counting list homomorphisms from graphs of bounded treewidth: tight complexity bounds. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 431–458. SIAM, 2022. doi:10.1137/1.9781611977073.22.

15 Robert Ganian, Thekla Hamm, Viktoriia Korchemna, Karolina Okrasa, and Kirill Simonov. The fine-grained complexity of graph homomorphism parameterized by clique-width. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 66:1–66:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.66.

16 A. M. H. Gerards. Homomorphisms of graphs into odd cycles. *J. Graph Theory*, 12(1):73–83, 1988. doi:10.1002/JGT.3190120108.

17 Carla Groenland, Isja Mannens, Jesper Nederlof, Marta Piecyk, and Paweł Rzążewski. Towards tight bounds for the graph homomorphism problem parameterized by cutwidth via asymptotic rank parameters. *CoRR*, abs/2312.03859, 2023. doi:10.48550/arXiv.2312.03859.

18 Pavol Hell and Jaroslav Nešetřil. On the complexity of h-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

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 Tereza Klimosová and Vibha Sahlot. 3-coloring c_4 or c_3-free diameter two graphs. In Pat Morin and Subhash Suri, editors, *Algorithms and Data Structures – 18th International Symposium, WADS 2023, Montreal, QC, Canada, July 31 – August 2, 2023, Proceedings*, volume 14079 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2023. doi:10.1007/978-3-031-38906-1_36.

21 Aleksej Dmitrievich Korshunov. On the diameter of graphs. *Soviet Math*, 12:302:305, 1971.

22 Hong-Jian Lai. Unique graph homomorphisms onto odd cycles, II. *J. Comb. Theory, Ser. B*, 46(3):363–376, 1989. doi:10.1016/0095-8956(89)90056-7.

23 Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring h-free graphs of bounded diameter. 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 14:1–14:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.MFCS.2019.14.

24 Barnaby Martin, Daniël Paulusma, and Siani Smith. Colouring graphs of bounded diameter in the absence of small cycles. *Discret. Appl. Math.*, 314:150–161, 2022. doi:10.1016/J.DAM.2022.02.026.

25 George B. Mertzios and Paul G. Spirakis. Algorithms and almost tight results for 3-colorability of small diameter graphs. *Algorithmica*, 74(1):385–414, 2016. doi:10.1007/S00453-014-9949-6.

26 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.

27 Karolina Okrasa and Paweł Rzążewski. Fine-grained complexity of the graph homomorphism problem for bounded-treewidth graphs. *SIAM J. Comput.*, 50(2):487–508, 2021. doi:10.1137/20M1320146.

28 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

29 Marta Piecyk. C_{2k+1}-coloring of bounded-diameter graphs. *CoRR*, abs/2403.06694, 2024. doi:10.48550/arXiv.2403.06694.

30 Sebastian Schnettler. A structured overview of 50 years of small-world research. *Soc. Networks*, 31(3):165–178, 2009. doi:10.1016/J.SOCNET.2008.12.004.

31 Petra Sparl and Janez Zerovnik. Homomorphisms of hexagonal graphs to odd cycles. *Discret. Math.*, 283(1-3):273–277, 2004. doi:10.1016/J.DISC.2003.12.012.

Demonic Variance and a Non-Determinism Score for Markov Decision Processes

Jakob Piribauer
Technische Universität Dresden, Germany
Universität Leipzig, Germany

Abstract

This paper studies the influence of probabilism and non-determinism on some quantitative aspect X of the execution of a system modeled as a Markov decision process (MDP). To this end, the novel notion of *demonic variance* is introduced: For a random variable X in an MDP $\mathcal{M}$, it is defined as $1/2$ times the maximal expected squared distance of the values of X in two independent execution of $\mathcal{M}$ in which also the non-deterministic choices are resolved independently by two distinct schedulers.

It is shown that the demonic variance is between 1 and 2 times as large as the maximal variance of X in $\mathcal{M}$ that can be achieved by a single scheduler. This allows defining a non-determinism score for $\mathcal{M}$ and X measuring how strongly the difference of X in two executions of $\mathcal{M}$ can be influenced by the non-deterministic choices. Properties of MDPs $\mathcal{M}$ with extremal values of the non-determinism score are established. Further, the algorithmic problems of computing the maximal variance and the demonic variance are investigated for two random variables, namely weighted reachability and accumulated rewards. In the process, also the structure of schedulers maximizing the variance and of scheduler pairs realizing the demonic variance is analyzed.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Markov decision processes, variance, non-determinism, probabilism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.79

Related Version *Full Version*: `https://arxiv.org/abs/2406.18727` [26]

Funding This work was partly funded by the DFG Grant 389792660 as part of TRR 248 (Foundations of Perspicuous Software Systems).

1 Introduction

In software and hardware systems, uncertainty manifests in two distinct forms: *non-determinism* and *probabilism*. Non-determinism emerges from, e.g., unknown operating environments, user interactions, or concurrent processes. Probabilistic behavior arises through deliberate randomization in algorithms or can be inferred, e.g., from probabilities of component failures. In this paper, we investigate the uncertainty in the value X of some quantitative aspect of a system whose behavior is subject to non-determinism *and* probabilism. On the one hand, we aim to quantify this uncertainty. In the spirit of the variance that quantifies uncertainty in purely probabilistic settings, we introduce the notion of *demonic variance* that generalizes the variance in the presence of non-determinism. On the other hand, we provide a *non-determinism score* (NDS) based on this demonic variance that measures the extent to which the uncertainty of X can be ascribed to the non-determinism.

As formal models, we use Markov decision processes (MDPs, see, e.g., [29]), one of the most prominent models combining non-determinism and probabilism, heavily used in verification, operations research, and artificial intelligence. The non-deterministic choices in an MDP are resolved by a *scheduler*. Once a scheduler is fixed, the system behaves purely probabilistically.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 79; pp. 79:1–79:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

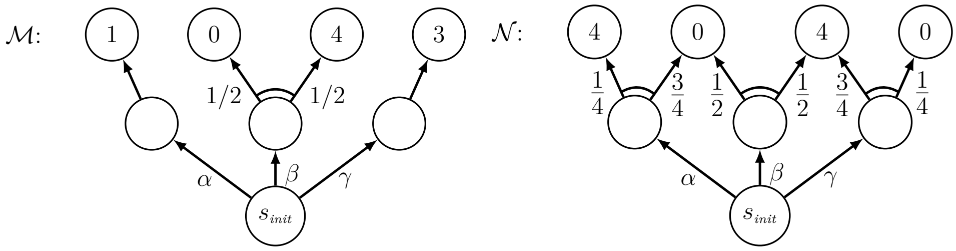


Figure 1 MDPs modeling a communication protocol and the time required to process a message.

Demonic variance. For a random variable Y, the variance is equal to half the expected squared deviation of two independent copies Y_1 and Y_2 of Y:

$$\mathbb{V}(Y) \overset{\text{def}}{=} \mathbb{E}((Y-\mathbb{E}(Y))^2) = \mathbb{E}(Y^2) - \mathbb{E}(Y)^2 = \frac{1}{2}\mathbb{E}(Y_1^2 - 2Y_1Y_2 + Y_2^2) = \frac{1}{2}\mathbb{E}((Y_1-Y_2)^2).$$

For a quantity X in an MDP $\mathcal{M}$, we obtain a random variable $X_{\mathcal{M}}^{\mathfrak{S}}$ for each scheduler $\mathfrak{S}$.[1] The maximal variance $\mathbb{V}_{\mathcal{M}}^{\max}(X) \overset{\text{def}}{=} \sup_{\mathfrak{S}} \mathbb{V}(X_{\mathcal{M}}^{\mathfrak{S}})$ can serve as a measure for the "amount of probabilistic uncertainty" regarding X present in the MDP. However, in the presence of non-determinism, quantifying the spread of outcomes in terms of the squared deviation of two independent executions of a system gives rise to a whole new meaning: We can allow the non-determinism to be resolved independently as well. To this end, we consider two different scheduler $\mathfrak{S}_1$ and $\mathfrak{S}_2$ in two independent copies $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\mathcal{M}$ and define

$$\mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_1,\mathfrak{S}_2}(X) \overset{\text{def}}{=} \frac{1}{2}\mathbb{E}((X_{\mathcal{M}_1}^{\mathfrak{S}_1} - X_{\mathcal{M}_2}^{\mathfrak{S}_2})^2).$$

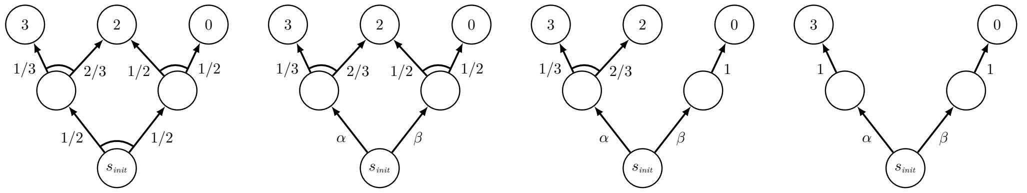
If we now allow for a *demonic* choice of the two schedulers making this uncertainty as large as possible, we arrive at the *demonic variance* $\mathbb{V}_{\mathcal{M}}^{dem}(X) \overset{\text{def}}{=} \sup_{\mathfrak{S}_1,\mathfrak{S}_2} \mathbb{V}_{\mathcal{M}}^{\mathfrak{S}_1,\mathfrak{S}_2}(X)$ of X in $\mathcal{M}$.

▶ **Example 1.1.** To illustrate a potential use case, consider a communication network in which messages are processed according to a randomized protocol employed on different hardware at the different nodes of the network. A low worst-case expected processing time of the protocol is clearly desirable. In addition, however, large differences in the processing time at different nodes make buffering necessary and increase the risk of package losses.

Consider the MDPs $\mathcal{M}$ and $\mathcal{N}$ in Fig. 1 modeling such a communication protocol. Initially, a non-deterministic choice between α, β, and γ is made. Then, a final node containing the processing time X is reached according to the depicted distributions. In both MDPs, the expected value of X lies between 1 and 3 for all schedulers $\mathfrak{S}$ – with the values 1 and 3 being realized by α and γ. Furthermore, as the outcomes lie between 0 and 4, the distribution over outcomes leading to the highest possible variance of 4 is the one that takes value 0 and 4 with probability $\frac{1}{2}$ each, which is realized by a scheduler choosing β. So, $\mathbb{V}_{\mathcal{M}}^{\max}(X) = \mathbb{V}_{\mathcal{N}}^{\max}(X) = 4$.

However, the demonic variances are different: Our results will show that the demonic variance is obtained by a pair of *deterministic* schedulers that do not randomize over the non-deterministic choices. In $\mathcal{M}$, we can easily check that no combination of such schedulers $\mathfrak{S}$ and $\mathfrak{T}$ leads to a value $\mathbb{V}_{\mathcal{M}}^{\mathfrak{S},\mathfrak{T}}(X)$ of more than $4 = \mathbb{V}_{\mathcal{M}}^{\beta,\beta}(X)$ where β denotes the scheduler that chooses β with probability 1. In $\mathcal{N}$, on the other hand, the demonic variance is $\mathbb{V}_{\mathcal{N}}^{dem}(X) = \mathbb{V}_{\mathcal{N}}^{\alpha,\gamma}(X) = \frac{1}{2}\mathbb{E}((X_{\mathcal{N}_1}^{\alpha} - X_{\mathcal{N}_2}^{\gamma})^2) = \frac{1}{2}\left(\frac{10}{16}\cdot 16\right) = 5$.

[1] Note that the notation $X_{\mathcal{M}}^{\mathfrak{S}}$ here differs from the notation used in the technical part of the paper.



(a) NDS $= 0$. **(b)** NDS ≈ 0.32. **(c)** NDS ≈ 0.92. **(d)** NDS $= 1$.

Figure 2 Example MDPs with different non-determinism scores (NDSs).

So, despite the same maximal variance and range of expected values, the worst-case expected squared deviation between two values of X in independent executions is worse in $\mathcal{N}$ than in $\mathcal{M}$. Hence, we argue that the protocol modeled by $\mathcal{M}$ should be preferred.

Non-determinism score (NDS). By the definition of the demonic variance, it is clear that $\mathbb{V}^{dem}_{\mathcal{M}}(X) \geq \mathbb{V}^{\max}_{\mathcal{M}}(X)$. Under mild assumptions ensuring the well-definedness, we will prove that $\mathbb{V}^{dem}_{\mathcal{M}}(X) \leq 2\mathbb{V}^{\max}_{\mathcal{M}}(X)$, too. So, the demonic variance is between 1 and 2 times as large as the maximal variance. We use this to define the *non-determinism score (NDS)*

$$\mathrm{NDS}(\mathcal{M}, X) \overset{\text{def}}{=} \frac{\mathbb{V}^{dem}_{\mathcal{M}}(X) - \mathbb{V}^{\max}_{\mathcal{M}}(X)}{\mathbb{V}^{\max}_{\mathcal{M}}(X)} \in [0,1].$$

The NDS captures how much larger the expected squared deviation of two outcomes can be made by resolving the non-determinism in two executions independently compared to how large it can be solely due to the probabilism under a single resolution of the non-determinism.

▶ **Example 1.2.** For an illustration of the NDS, four simple MDPs and their NDSs are depicted in Figure 2. In all of the MDPs except for the first one, a scheduler has to make a (randomized) choice over actions α and β in the initial state s_{init}. Afterwards one of the terminal states is reached according to the specified probabilities. The terminal states are equipped with a weight that specifies the value of X at the end of the execution. For all of these MDPs, the maximal variance can be computed by expressing the variance in terms of the probability p that α is chosen and maximizing the resulting quadratic function. In the interest of brevity, we do not present these computations. The pair of (deterministic) schedulers realizing the demonic variance always consists of the scheduler choosing α and the scheduler choosing β making it easy to compute the demonic variance in these examples.

Potential applications. First of all, the demonic variance might serve as the basis for refined guarantees on the behavior of systems, in particular, when employed in different environments. As a first result in this direction, we will prove an analogue to Chebyshev's Inequality using the demonic variance. Further, as illustrated in Example 1.1, achieving a low demonic variance or NDS can be desirable when designing systems. Hence, a reasonable synthesis task could be to design a system ensuring a high expected value of a quantity X while keeping the demonic variance of X below a threshold.

Secondly, the demonic variance and the NDS can serve to enhance the explainability of a system's behavior, a topic of growing importance in the area of formal verification (see, e.g., [4] for an overview). More concretely, the NDS can be understood as a measure assigning *responsibility* for the scattering of a quantity X in different executions to the non-determinism and the probabilism present in the system, respectively. Further, considering the NDS for different starting states makes it possible to pinpoint regions of the state space in which the non-determinism has a particularly high influence. Notions of responsibility that quantify to which extent certain facets of the behavior of a system can be ascribed to certain components, states, or events have been studied in various settings [10, 33, 6, 25, 5].

Finally, the NDS can also be understood as a measure for the power of control when non-determinism models controllable aspects of a system. This interpretation could be useful, e.g., when designing exploration strategies in reinforcement learning. Here, the task is to learn good strategies as fast as possible by interacting with a system. One of the main challenges is to decide which regions of the state space to explore (see [21] for a recent survey). Estimations for the NDS starting from different states could be useful here: States from which the NDS is high might be more promising to explore than states from which the NDS is low as the difference in received rewards from such a state is largely subject to randomness.

Contributions. Besides establishing general results for the demonic variance and the NDS, we investigate the two notions for weighted reachability and accumulated rewards. For weighted reachability, terminal states of an MDP are equipped with a weight that is received if an execution ends in this state. For accumulated rewards, all states are assigned rewards that are summed up along an execution. The main contributions of this paper are as follows.

- We introduce the novel notions of demonic variance and non-determinism score. For general random variables X, we prove that the demonic variance is at most twice as large as the maximal variance. Furthermore, we prove an analogue of Chebyshev's inequality. For the non-determinism score, we establish consequences of a score of 0 or 1.
- In the process, we prove a result of independent interest using a topology on the space of schedulers that states that convergence with respect to this topology implies convergence of the corresponding probability measures.
- For weighted reachability, we show that the maximal and the demonic variance can be computed via quadratic programs. For the maximal variance, this results in a polynomial-time algorithm; for the demonic variance, in a separable bilinear program of polynomial size yielding an exponential time upper bound. Further, we establish that there is a memoryless scheduler maximizing the variance and a pair of memoryless deterministic schedulers realizing the demonic variance.
- For accumulated rewards, we prove that the maximal variance and an optimal finite-memory scheduler can be computed in exponential time. Further, we prove that the demonic variance is realized by a pair of deterministic finite-memory schedulers which can be computed via a bilinear program of exponential size.

Related work. We are not aware of investigations of notions similar to the demonic variance for MDPs. Previous work on the variance in MDPs usually focused on the minimization of the variance. In [24], the problem to find schedulers that ensure a certain expected value while keeping the variance below a threshold is investigated for accumulated rewards in the finite horizon setting. It is shown that deciding whether there is a scheduler ensuring variance 0 is NP-hard. In [22], the minimization of the variance of accumulated rewards and of the mean payoff is addressed with a focus on optimality equations and no algorithmic results. The variance of accumulated weights in Markov chains is shown to be computable in polynomial time in [32]. For the mean payoff, algorithms were given to compute schedulers that achieve given bounds on the expectation and notions of variance and variability in [9].

One objective incorporating the variance that has been studied on MDPs is the variance-penalized expectation (VPE) [16, 13, 28]. Here, the goal is to find a scheduler that maximizes the expected reward minus a penalty factor times the variance. In [28], the objective is studied for accumulated rewards. Methodically, our results for the maximal and demonic variance of accumulated rewards share similarities with the techniques of [28] and we make use of some results proved there, such as the result that among expectation-optimal schedulers a variance-optimal memoryless deterministic scheduler can be computed in polynomial time. Nevertheless, the optimization of the VPE inherently requires the minimization of the variance. In particular, it is shown in [28] that deterministic schedulers are optimal for the VPE, while randomization is necessary for the maximization of the variance.

Besides the variance, several other notions that aim to bound the uncertainty of the outcome of some quantitative aspect in MDPs have been studied – in particular, in the context of risk-averse optimization: Given a probability p, quantiles for a quantity X are the best bound B such that X exceeds B with probability at most p in the worst or best case. For accumulated rewards in MDPs, quantiles have been studied in [31, 3, 17, 30]. The *conditional value-at-risk* is a more involved measures that quantifies how far the probability mass of the tail of the probability distribution lies above a quantile. In [20], this notion has been investigated for weighted reachability and mean payoff; in [27] for accumulated rewards. A further measure incentivizing a high expected value while keeping the probability of low outcomes small is the entropic risk measure. For accumulated rewards, this measure has been studied in [2] in stochastic games that extend MDPs with an adversarial player.

Finally, as the demonic variance is a measure that looks at a system across different executions, there is a conceptual similarity to hyperproperties [12, 11]. For probabilistic systems, logics expressing hyperproperties that allow to quantify over different executions or schedulers have been introduced in [1, 15].

2 Preliminaries

Notations for Markov decision processes. A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, \mathit{Act}, P, s_{\mathit{init}})$ where S is a finite set of states, Act a finite set of actions, $P\colon S \times \mathit{Act} \times S \to [0,1] \cap \mathbb{Q}$ the transition probability function, and $s_{\mathit{init}} \in S$ the initial state. We require that $\sum_{t\in S} P(s,\alpha,t) \in \{0,1\}$ for all $(s,\alpha) \in S \times \mathit{Act}$. We say that action α is *enabled* in state s iff $\sum_{t\in S} P(s,\alpha,t) = 1$ and denote the set of all actions that are enabled in state s by $\mathit{Act}(s)$. We further require that $\mathit{Act}(s) \neq \emptyset$ for all $s \in S$. If for a state s and all actions $\alpha \in \mathit{Act}(s)$, we have $P(s,\alpha,s) = 1$, we say that s is *absorbing*. The paths of $\mathcal{M}$ are finite or infinite sequences $s_0\,\alpha_0\,s_1\,\alpha_1\dots$ where states and actions alternate such that $P(s_i,\alpha_i,s_{i+1}) > 0$ for all $i \geq 0$. For $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\dots\alpha_{k-1}\,s_k$, $P(\pi) = P(s_0,\alpha_0,s_1)\cdot\ldots\cdot P(s_{k-1},\alpha_{k-1},s_k)$ denotes the probability of π and $\mathit{last}(\pi) = s_k$ its last state. Often, we equip MDPs with a reward function $\mathit{rew}\colon S \times \mathit{Act} \to \mathbb{N}$. The *size* of $\mathcal{M}$ is the sum of the number of states plus the total sum of the encoding lengths in binary of the non-zero probability values $P(s,\alpha,s')$ as fractions of co-prime integers as well as the encoding length in binary of the rewards if a reward function is used. A *Markov chain* is an MDP in which the set of actions is a singleton. In this case, we can drop the set of actions and consider a Markov chain as a tuple $\mathcal{M} = (S, P, s_{\mathit{init}}, \mathit{rew})$ where P now is a function from $S \times S$ to $[0,1]$ and rew a function from S to $\mathbb{N}$.

An *end component* of $\mathcal{M}$ is a strongly connected sub-MDP formalized by a subset $S' \subseteq S$ of states and a non-empty subset $\mathfrak{A}(s) \subseteq \mathit{Act}(s)$ for each state $s \in S'$ such that for each $s \in S'$, $t \in S$ and $\alpha \in \mathfrak{A}(s)$ with $P(s,\alpha,t) > 0$, we have $t \in S'$ and such that in the resulting sub-MDP all states are reachable from each other. An end-component is a 0-end-component if it only contains state-action-pairs with reward 0. Given two MDPs $\mathcal{M} = (S, \mathit{Act}, P, s_{\mathit{init}})$ and $\mathcal{N} = (S', \mathit{Act}', P', s'_{\mathit{init}})$, we define the (synchronous) product $\mathcal{M} \otimes \mathcal{N}$ as the tuple $(S \times S', \mathit{Act} \times \mathit{Act}', P^{\otimes}, (s_{\mathit{init}}, s'_{\mathit{init}}))$ where we define $P^{\otimes}((s,s'),(\alpha,\beta),(t,t')) = P(s,\alpha,t)\cdot P(s',\beta,t')$ for all $(s,s'),(t,t') \in S \times S'$ and $(\alpha,\beta) \in \mathit{Act} \times \mathit{Act}'$.

Schedulers. A *scheduler* (also called *policy*) for $\mathcal{M}$ is a function $\mathfrak{S}$ that assigns to each finite path π a probability distribution over $\mathit{Act}(\mathit{last}(\pi))$. If $\mathfrak{S}(\pi) = \mathfrak{S}(\pi')$ for all finite paths π and π' with $\mathit{last}(\pi) = \mathit{last}(\pi')$, we say that $\mathfrak{S}$ is *memoryless.* In this case, we also view schedulers as functions mapping states $s \in S$ to probability distributions over $\mathit{Act}(s)$. A scheduler $\mathfrak{S}$ is called deterministic if $\mathfrak{S}(\pi)$ is a Dirac distribution for each finite path π, in which case we also view the scheduler as a mapping to actions in $\mathit{Act}(\mathit{last}(\pi))$. Given two MDPs $\mathcal{M} = (S, \mathit{Act}, P, s_{\mathit{init}})$ and $\mathcal{N} = (S', \mathit{Act}', P', s'_{\mathit{init}})$ and two schedulers $\mathfrak{S}$ and $\mathfrak{T}$ for $\mathcal{M}$ and $\mathcal{N}$, respectively, we define the product scheduler $\mathfrak{S} \otimes \mathfrak{T}$ for $\mathcal{M} \otimes \mathcal{N}$ by defining for a finite path $\pi = (s_0, t_0)\,(\alpha_0, \beta_0)\,(s_1, t_1)\,\ldots\,(s_k, t_k)$: $\mathfrak{S} \otimes \mathfrak{T}(\pi)(\alpha, \beta) = \mathfrak{S}(s_0\,\alpha_0\,\ldots\,s_k)(\alpha) \cdot \mathfrak{T}(t_0\,\beta_0\,\ldots\,t_k)(\beta)$ for all $(\alpha, \beta) \in \mathit{Act} \times \mathit{Act}'$.

Probability measure. We write $\Pr^{\mathfrak{S}}_{\mathcal{M},s}$ to denote the probability measure induced by a scheduler $\mathfrak{S}$ and a state s of an MDP $\mathcal{M}$. It is defined on the σ-algebra generated by the cylinder sets $\mathit{Cyl}(\pi)$ of all infinite extensions of a finite path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\alpha_{k-1}\,s_k$ starting in state s, i.e., $s_0 = s$, by assigning to $\mathit{Cyl}(\pi)$ the probability that π is realized under $\mathfrak{S}$, which is $P^{\mathfrak{S}}(\pi) \overset{\text{def}}{=} \prod_{i=0}^{k-1} \mathfrak{S}(s_0\,\alpha_0\ldots\,s_i)(\alpha_i) \cdot P(s_i, \alpha_0, s_{i+1})$. This can be extended to a unique probability measure on the mentioned σ-algebra. For details, see [29]. For a random variable X, i.e., a measurable function defined on infinite paths in $\mathcal{M}$, we denote the expected value of X under a scheduler $\mathfrak{S}$ and state s by $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$. We define $\mathbb{E}^{\min}_{\mathcal{M},s}(X) \overset{\text{def}}{=} \inf_{\mathfrak{S}} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$ and $\mathbb{E}^{\max}_{\mathcal{M},s}(X) \overset{\text{def}}{=} \sup_{\mathfrak{S}} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$. The variance of X under the probability measure determined by $\mathfrak{S}$ and s in $\mathcal{M}$ is denoted by $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}(X)$ and defined by $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}(X) \overset{\text{def}}{=} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}((X - \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X))^2) = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X^2) - \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)^2$. We define $\mathbb{V}^{\max}_{\mathcal{M},s}(X) \overset{\text{def}}{=} \sup_{\mathfrak{S}} \mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}(X)$. If $s = s_{\mathit{init}}$, we sometimes drop the subscript s in $\Pr^{\mathfrak{S}}_{\mathcal{M},s}$, $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}$ and $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M},s}(X)$.

Mixing schedulers. Intuitively, we often want to use a scheduler that initially decides to behave like a scheduler $\mathfrak{S}$ and then to stick to this scheduler with probability p and to behave like a scheduler $\mathfrak{T}$ with probability $1 - p$. As this intuitive description does not match the definition of schedulers as functions from finite paths[2], we provide a formal definition: For two schedulers $\mathfrak{S}$ and $\mathfrak{T}$ and $p \in [0, 1]$, we use $p\mathfrak{S} \oplus (1-p)\mathfrak{T}$ to denote the following scheduler. For a path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\alpha_{k-1}\,s_k$, we define for an action α enabled in s_k

$$(p\mathfrak{S} \oplus (1-p)\mathfrak{T})(\pi)(\alpha) \overset{\text{def}}{=} \frac{p \cdot P^{\mathfrak{S}}(\pi) \cdot \mathfrak{S}(\pi)(\alpha)}{p \cdot P^{\mathfrak{S}}(\pi) + (1-p) \cdot P^{\mathfrak{T}}(\pi)} + \frac{(1-p) \cdot P^{\mathfrak{T}}(\pi) \cdot \mathfrak{T}(\pi)(\alpha)}{p \cdot P^{\mathfrak{S}}(\pi) + (1-p) \cdot P^{\mathfrak{T}}(\pi)}.$$

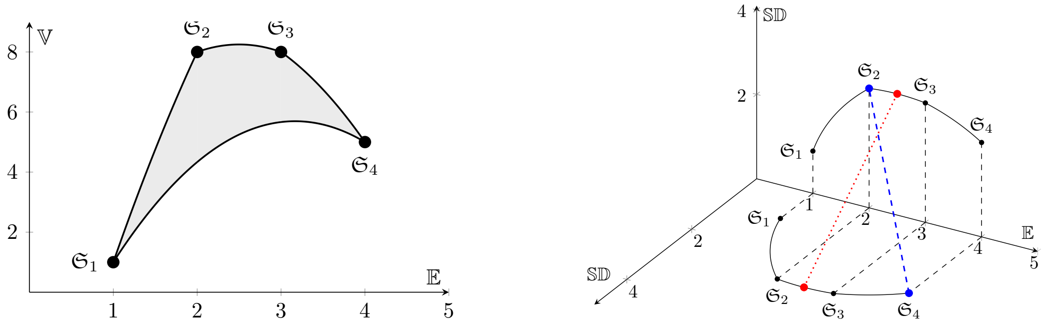
This is well-defined for any path that has positive probability under $\mathfrak{S}$ or $\mathfrak{T}$. The following result is folklore; a proof is included in the full version [26].

▶ **Proposition 2.1.** *Let the schedulers $\mathfrak{S}$ and $\mathfrak{T}$ and the value p be as above. Then, for any path $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\ldots\alpha_{k-1}\,s_k$, we have $P^{p\mathfrak{S}\oplus(1-p)\mathfrak{T}(\pi)}(\pi) = pP^{\mathfrak{S}}(\pi) + (1-p)P^{\mathfrak{T}}(\pi)$.*

We conclude $\Pr^{p\mathfrak{S}\oplus(1-p)\mathfrak{T}}_{\mathcal{M},s}(A) = p\Pr^{\mathfrak{S}}_{\mathcal{M},s}(A) + (1-p)\Pr^{\mathfrak{T}}_{\mathcal{M},s}(A)$ for any measurable set of paths A. Hence, we can think of the scheduler $p\mathfrak{S} \oplus (1-p)\mathfrak{T}$ as behaving like $\mathfrak{S}$ with probability p and like $\mathfrak{T}$ with probability $(1-p)$. In particular, we can also conclude that for a random variable X, we have $\mathbb{E}^{p\mathfrak{S}\oplus(1-p)\mathfrak{T}}_{\mathcal{M},s}(X) = p\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X) + (1-p)\mathbb{E}^{\mathfrak{T}}_{\mathcal{M},s}(X)$. For the variance, we obtain the following as shown in full version [26].

▶ **Lemma 2.2.** *Given $\mathcal{M}$, X, and two schedulers $\mathfrak{S}_1$ and $\mathfrak{S}_2$, as well as $p \in [0, 1]$, let $\mathfrak{T} = p\mathfrak{S}_1 \oplus (1-p)\mathfrak{S}_2$. Then, $\mathbb{V}^{\mathfrak{T}}_{\mathcal{M}}(X) = p\mathbb{V}^{\mathfrak{S}_1}_{\mathcal{M}}(X) + (1-p)\mathbb{V}^{\mathfrak{S}_2}_{\mathcal{M}}(X) + p(1-p)(\mathbb{E}^{\mathfrak{S}_1}_{\mathcal{M}}(X) - \mathbb{E}^{\mathfrak{S}_2}_{\mathcal{M}}(X))^2$.*

[2] This description would be admissible if we allowed stochastic memory updates (see, e.g., [8]).



(a) Possible combinations of variance and expectation in Example 3.2.

(b) Plot of the standard deviation over the expectation on two orthogonal planes.

Figure 3 Graphical illustration of the task to find the demonic variance (see Example 3.2).

Topology and convergence of measures. Given a family of topological spaces $((S_i, \tau_i))_{i\in I}$, the product topology τ on $\prod_{i\in I} S_i$ is the coarsest topology such that the projections $p_i \colon \prod_{i\in I} S_i \to S_i, \quad (s_i)_{i\in I} \mapsto s_i$ are continuous for all $i \in I$. For measures $(\mu_j)_{j\in\mathbb{N}}$ and μ on a measure space (Ω, Σ) where Ω is a metrizable topological space and Σ the Borel σ-algebra on Ω, we say that the sequence $(\mu_j)_{j\in\mathbb{N}}$ *weakly converges* to μ if for all bounded continuous functions $f \colon \Omega \to \mathbb{R}$, we have $\lim_{j\to\infty} \int f \mathrm{d}\mu_j = \int f \mathrm{d}\mu$. The set of infinite paths $\Pi_{\mathcal{M}}$ of an MDP $\mathcal{M}$ with the topology generated by the cylinder sets is metrizable as we can define the metric $d(\pi, \pi') = 2^{-\ell}$ where ℓ is the length of the longest common prefix of π and π'.

3 Demonic variance and non-determinism score

In this section, we formally define the demonic variance. After proving first auxiliary results, we prove an analogue of Chebyshev's Inequality using the demonic variance. Then, we introduce the non-determinism score and investigate necessary and sufficient conditions for this score to be 0 or 1. Proofs omitted here can be found in the full version [26].

Throughout this section, let $\mathcal{M} = (S, Act, P, s_{init})$ be an MDP and let X be a random variable, i.e., a Borel measurable function on the infinite paths of $\mathcal{M}$. We will work under two assumptions that ensure that all notions are well-defined: First, note that $\mathbb{V}^{\max}_{\mathcal{M}}(X) = 0$ implies that there is a constant c such that under all schedulers $\mathfrak{S}$, we have $\Pr^{\mathfrak{S}}_{\mathcal{M}}(X = c) = 1$ – an uninteresting case. Furthermore, for meaningful definitions of demonic variance and non-determinism score, we need that the expected value and the variance of X in $\mathcal{M}$ are finite. Hence, we work under the following assumption:

▶ **Assumption 3.1.** *We assume that* $0 < \mathbb{V}^{\max}_{\mathcal{M}}(X) < \infty$ *and that* $\sup_{\mathfrak{S}} \left|\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(X)\right| < \infty$.

3.1 Demonic variance

As described in the introduction, the idea behind the demonic variance is to quantify the expected squared deviation of X in two independent executions of $\mathcal{M}$, in which the non-determinism is resolved independently as well. We use the following notation: Given a path in $\mathcal{M} \otimes \mathcal{M}$ consisting of a sequence of pairs of states and pairs of actions, we denote by X_1 and X_2 the function X applied to the projection of the path on the first component and on the second component, respectively. Given two schedulers $\mathfrak{S}_1$ and $\mathfrak{S}_2$ for $\mathcal{M}$, we define

$$\mathbb{V}^{\mathfrak{S}_1,\mathfrak{S}_2}_{\mathcal{M}}(X) \overset{\text{def}}{=} \frac{1}{2}\mathbb{E}^{\mathfrak{S}_1\otimes\mathfrak{S}_2}_{\mathcal{M}\otimes\mathcal{M}}((X_1 - X_2)^2).$$

Intuitively, in this definition two independent executions of $\mathcal{M}$ are run in parallel while the non-determinism is resolved by $\mathfrak{S}_1$ in the first execution and by $\mathfrak{S}_2$ in the second component. As the two components in the products $\mathcal{M} \otimes \mathcal{M}$ and $\mathfrak{S}_1 \otimes \mathfrak{S}_2$ are independent, the resulting distributions of X in the two components, i.e., X_1 and X_2 are independent as well. The factor $\frac{1}{2}$ is included as for a random variable Y, this factor also appears in the representation $\mathbb{V}(Y) = \frac{1}{2}\mathbb{E}((Y_1 - Y_2)^2)$ for two independent copies Y_1 and Y_2 of Y.

The *demonic variance* is now the worst-case value when ranging over all pairs of schedulers:

$$\mathbb{V}^{dem}_{\mathcal{M}}(X) \overset{\text{def}}{=} \sup_{\mathfrak{S}_1,\mathfrak{S}_2} \frac{1}{2}\mathbb{E}^{\mathfrak{S}_1\otimes\mathfrak{S}_2}_{\mathcal{M}\otimes\mathcal{M}}((X_1 - X_2)^2).$$

A first simple, but useful, result allows us to express $\mathbb{V}^{\mathfrak{S}_1,\mathfrak{S}_2}_{\mathcal{M}}(X)$ in terms of the expected values and variances of X under $\mathfrak{S}_1$ and $\mathfrak{S}_2$.

▶ **Lemma 3.1.** *Given two schedulers $\mathfrak{S}_1$ and $\mathfrak{S}_2$ for $\mathcal{M}$, we have*

$$\mathbb{V}^{\mathfrak{S}_1,\mathfrak{S}_2}_{\mathcal{M}}(X) = \frac{1}{2}\left(\mathbb{V}^{\mathfrak{S}_1}_{\mathcal{M}}(X) + \mathbb{V}^{\mathfrak{S}_2}_{\mathcal{M}}(X) + (\mathbb{E}^{\mathfrak{S}_1}_{\mathcal{M}}(X) - \mathbb{E}^{\mathfrak{S}_2}_{\mathcal{M}}(X))^2\right).$$

This lemma allows us to provide an insightful graphical interpretation of the demonic variance using the standard deviation $\mathbb{SD}(X) \overset{\text{def}}{=} \sqrt{\mathbb{V}(X)}$ of a random variable X:

▶ **Example 3.2.** Suppose in an MDP $\mathcal{M}$, there are four deterministic scheduler $\mathfrak{S}_1, \dots, \mathfrak{S}_4$ with expected values 1, 2, 3, and 4 and variances 1, 8, 8, and 5 for a random variable X. Lemma 2.2 allows us to compute the variances of schedulers obtained by randomization leading to parabolic line segments in the expectation-variance-plane as depicted in Figure 3a (see also [28]). Further randomizations also make it possible to realize any combination of expectation and variance in the interior of the resulting shape. When looking for the maximal variance and the demonic variance, only the upper bound of this shape is relevant.

In Figure 3b, we now depict the standard deviations of schedulers on this upper bound over the expectation twice on two orthogonal planes. Clearly, the highest standard deviation (and consequently variance) is obtained for the expected value 2.5 in this example. The red dotted line of length $\sqrt{2\mathbb{V}^{\max}_{\mathcal{M}}(X)}$ connects the two points corresponding to this maximum on the two planes. Considering $\mathfrak{S}_2$ and $\mathfrak{S}_4$, we can also find the value $\sqrt{2\mathbb{V}^{\mathfrak{S}_2,\mathfrak{S}_4}_{\mathcal{M}}(X)}$: The blue dashed line connects the point corresponding to $\mathfrak{S}_2$ on one of the planes to the point corresponding to $\mathfrak{S}_4$ on the other plane. By the Pythagorean theorem, its length is

$$\sqrt{\sqrt{\mathbb{V}^{\mathfrak{S}_2}_{\mathcal{M}}(X)}^2 + (\mathbb{E}^{\mathfrak{S}_2}_{\mathcal{M}}(X) - (\mathbb{E}^{\mathfrak{S}_4}_{\mathcal{M}}(X))^2 + \sqrt{\mathbb{V}^{\mathfrak{S}_4}_{\mathcal{M}}(X)}^2} = \sqrt{2\mathbb{V}^{\mathfrak{S}_2,\mathfrak{S}_4}_{\mathcal{M}}(X)}.$$

So, finding $\sqrt{2}$ times the "demonic standard deviation" and hence the demonic variance corresponds to finding two points on the two orthogonal graphs with maximal distance.

The relation between maximal and demonic variance is shown in the following proposition.

▶ **Proposition 3.3.** *We have $\mathbb{V}^{\max}_{\mathcal{M}}(X) \leq \mathbb{V}^{dem}_{\mathcal{M}}(X) \leq 2\mathbb{V}^{\max}_{\mathcal{M}}(X)$.*

By means of Chebyshev's Inequality, the variance can be used to bound the probability that a random variable Y lies far from its expected value. Using the demonic variance, we can prove an analogous result providing bounds on the probability that the outcomes of X in two independent executions of the MDP $\mathcal{M}$ lie far apart. This can be seen as a first step in the direction of using the demonic variance to provide guarantees on the behavior of a system.

▶ **Theorem 3.4.** *We have* $\Pr^{\mathfrak{S}\otimes\mathfrak{T}}_{\mathcal{M}\otimes\mathcal{M}}\left(|X_1 - X_2| \geq k \cdot \sqrt{\mathbb{V}^{dem}_{\mathcal{M}}(X)}\right) \leq \frac{2}{k^2}$ *for any* $k \in \mathbb{R}_{>0}$ *and schedulers* $\mathfrak{S}$ *and* $\mathfrak{T}$ *for* $\mathcal{M}$*.*

Using the result that $\mathbb{V}^{dem}_{\mathcal{M}}(X) \leq 2\mathbb{V}^{\max}_{\mathcal{M}}(X)$, we obtain the following variant of the inequality providing a weaker bound in terms of the maximal variance.

▶ **Corollary 3.5.** *We have* $\Pr^{\mathfrak{S}\otimes\mathfrak{T}}_{\mathcal{M}\otimes\mathcal{M}}\left(|X_1 - X_2| \geq k \cdot \sqrt{\mathbb{V}^{\max}_{\mathcal{M}}(X)}\right) \leq \frac{4}{k^2}$ *for any* $k \in \mathbb{R}_{>0}$ *and schedulers* $\mathfrak{S}$ *and* $\mathfrak{T}$ *for* $\mathcal{M}$*.*

3.2 Non-determinism score

We have seen that the demonic variance is larger than the maximal variance by a factor between 1 and 2. As described in the introduction, we use this insight as the basis for a score quantifying how much worse the "uncertainty" of X is when non-determinism can be resolved differently in two executions of an MDP compared to how bad it can be in a single execution. We define the non-determinism score (NDS)

$$\mathrm{NDS}(\mathcal{M}, X) \overset{\text{def}}{=} \frac{\mathbb{V}^{dem}_{\mathcal{M}}(X) - \mathbb{V}^{\max}_{\mathcal{M}}(X)}{\mathbb{V}^{\max}_{\mathcal{M}}(X)}.$$

By Assumption 3.1, the NDS is well-defined. By Proposition 3.3, the NDS always returns a value in $[0, 1]$. Clearly, in Markov chains, the NDS is 0. A bit more general, we can show:

▶ **Proposition 3.6.** *If* $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(X) = \mathbb{E}^{\mathfrak{T}}_{\mathcal{M}}(X)$ *for all schedulers* $\mathfrak{S}$ *and* $\mathfrak{T}$*, then* $\mathrm{NDS}(\mathcal{M}, X) = 0$*.*

In transition systems viewed as MDPs in which all transition probabilities are 0 or 1, the NDS is 1: Under Assumption 3.1 in a transition system the value of X must be bounded, i.e., $X \in [a, b]$ for some $a, b \in \mathbb{R}$ such that $\sup_\pi X(\pi) = b$ and $\inf_\pi X(\pi) = a$ where π ranges over all paths. Any path can be realized by a scheduler with probability 1. So, for any $\varepsilon > 0$, there are schedulers $\mathfrak{S}$ and $\mathfrak{T}$ with $\Pr^{\mathfrak{S}}_{\mathcal{M}}(X < a + \varepsilon) = 1$ and $\Pr^{\mathfrak{T}}_{\mathcal{M}}(X > b - \varepsilon) = 1$. Then, $\mathbb{V}^{\mathfrak{S},\mathfrak{T}}_{\mathcal{M}}(X) \geq \frac{1}{2}(b - a - 2\varepsilon)^2$. For $\varepsilon \to 0$, this converges to $\frac{(a-b)^2}{2}$. It is well-known that the variance of random variables taking values in $[a, b]$ is maximal for the random variable taking values a and b with probability $\frac{1}{2}$ each. The variance in this case is $\frac{(a-b)^2}{4}$. So, the maximal variance is (at most) half the demonic variance in this case. Consequently, the NDS is 1.

Of course, a NDS of 1 does not imply that there are no probabilistic transitions in $\mathcal{M}$. Nevertheless, a NDS of 1 has severe implications showing that the outcome of X can be heavily influenced by the non-determinism in this case as the following theorem shows:

▶ **Theorem 3.7.** *If* $\mathrm{NDS}(\mathcal{M}, X) = 1$*, the following statements hold:*

1. *For every* $\varepsilon > 0$*, there are schedulers* $\mathfrak{Min}_\varepsilon$ *and* $\mathfrak{Max}_\varepsilon$ *with* $\mathbb{E}^{\mathfrak{Min}_\varepsilon}_{\mathcal{M}}(X) \leq \mathbb{E}^{\min}_{\mathcal{M}}(X) + \varepsilon$ *and* $\mathbb{V}^{\mathfrak{Min}_\varepsilon}_{\mathcal{M}}(X) \leq \varepsilon$*, and* $\mathbb{E}^{\mathfrak{Max}_\varepsilon}_{\mathcal{M}}(X) \geq \mathbb{E}^{\max}_{\mathcal{M}}(X) - \varepsilon$ *and* $\mathbb{V}^{\mathfrak{Max}_\varepsilon}_{\mathcal{M}}(X) \leq \varepsilon$*.*
2. *If there are schedulers* $\mathfrak{S}_0$ *and* $\mathfrak{S}_1$*, with* $\mathbb{V}^{dem}_{\mathcal{M}}(X) = \mathbb{V}^{\mathfrak{S}_0,\mathfrak{S}_1}_{\mathcal{M}}(X)$*, then, for* $i = 0$ *or* $i = 1$*,* $\Pr^{\mathfrak{S}_i}_{\mathcal{M}}(X = \mathbb{E}^{\min}_{\mathcal{M}}(X)) = 1$ *and* $\Pr^{\mathfrak{S}_{1-i}}_{\mathcal{M}}(X = \mathbb{E}^{\max}_{\mathcal{M}}(X)) = 1$*.*
3. *If* X *is bounded and continuous wrt the topology generated by the cylinder sets, there are schedulers* $\mathfrak{Min}$ *and* $\mathfrak{Max}$ *with* $\Pr^{\mathfrak{Min}}_{\mathcal{M}}(X = \mathbb{E}^{\min}_{\mathcal{M}}(X)) = 1$ *and* $\Pr^{\mathfrak{Max}}_{\mathcal{M}}(X = \mathbb{E}^{\max}_{\mathcal{M}}(X)) = 1$*.*

The first two statements can be shown by elementary calculations. For the third statement, we use topological arguments. We view schedulers as elements of $\prod_{k=0}^{\infty} \mathrm{Distr}(Act)^{\mathrm{Paths}^k_{\mathcal{M}}}$ where $\mathrm{Paths}^k_{\mathcal{M}}$ is the set of paths of length k in $\mathcal{M}$ and prove the following result:

▶ **Proposition 3.8.** *The space of schedulers* $\mathrm{Sched}(\mathcal{M}) = \prod_{k=0}^{\infty} \mathrm{Distr}(Act)^{\mathrm{Paths}^k_{\mathcal{M}}}$ *with the product topology is compact. So, every sequence of schedulers has a converging subsequence in this space. Further, for a sequence* $(\mathfrak{S}_j)_{j\in\mathbb{N}}$ *converging to a scheduler* $\mathfrak{S}$ *in this space, the sequence of probability measures* $(\Pr^{\mathfrak{S}_j}_{\mathcal{M}})_{j\in\mathbb{N}}$ *weakly converges to the probability measure* $\Pr^{\mathfrak{S}}_{\mathcal{M}}$*.*

An example for a random variable that is bounded and continuous wrt the topology generated by the cylinder sets is the discounted reward: Given a reward function $rew\colon S \to \mathbb{R}$, the discounted reward of a path $\pi = s_0\alpha_0 s_1 \dots$ is defined as $DR_\lambda(\pi) \overset{\text{def}}{=} \sum_{j=0}^{\infty} \lambda^j rew(s_j)$ for some discount factor $\lambda \in (0,1)$. First, $|DR_\lambda|$ is bounded by $\max_{s\in S} |rew(s)| \cdot \frac{1}{1-\lambda}$. Further, for any $\varepsilon > 0$, let N be a natural number such that $\max_{s\in S} |rew(s)| \cdot \frac{\lambda^N}{1-\lambda} < \varepsilon$. Then, $|DR_\lambda(\pi) - DR_\lambda(\rho)| < \varepsilon$ for all paths π and ρ that share a prefix of length more than N.

4 Weighted reachability

We now address the problems to compute the demonic and the maximal variance for weighted reachability where a weight is collected on a run depending on which absorbing state is reached. As the NDS is defined via these two quantities, we do not address it separately here. Throughout this section, let $\mathcal{M} = (S, Act, P, s_{init})$ be an MDP with set of absorbing states $T \subseteq S$ and let $wgt\colon T \to \mathbb{Q}$ be a weight function. We define the random variable WR on infinite paths π by $\mathrm{WR}(\pi) = wgt(t)$ if π reaches the absorbing state $t \in T$, and $\mathrm{WR}(\pi) = 0$ if π does not reach T. The main result we are going to establish is the following:

Main result. *The maximal variance $\mathbb{V}^{\max}_{\mathcal{M}}(\mathrm{WR})$ and an optimal memoryless randomized scheduler can be computed in polynomial time.*

The demonic variance $\mathbb{V}^{dem}_{\mathcal{M}}(\mathrm{WR})$ can be computed as the solution to a bilinear program that can be constructed in polynomial time. Furthermore, there is a pair of memoryless deterministic schedulers realizing the demonic variance.

The following standard model transformation collapsing end components (see [14]) allows us to assume that T is reached almost surely under any scheduler: We add a new absorbing state t^* and set $wgt(t^*) = 0$ and collapse all maximal end components $\mathcal{E}$ in $S \setminus T$ to single states $s_{\mathcal{E}}$. In $s_{\mathcal{E}}$, all actions that were enabled in some state in $\mathcal{E}$ and that did not belong to $\mathcal{E}$ as well as a new action τ leading to t^* with probability 1 are enabled. In the resulting MDP $\mathcal{N}$, the set of absorbing states $T \cup \{t^*\}$ is reached almost surely under any scheduler. Further, for any scheduler $\mathfrak{S}$ for $\mathcal{M}$, there is a scheduler $\mathfrak{T}$ for $\mathcal{N}$ such that the distribution of WR is the same under $\mathfrak{S}$ in $\mathcal{M}$ and under $\mathfrak{T}$ in $\mathcal{N}$, and vice versa. So, w.l.o.g., assume the following:

▶ **Assumption 4.1.** *The set T is reached almost surely under any scheduler $\mathfrak{S}$ for $\mathcal{M}$.*

In the sequel, we first address the computation of the maximal variance and afterwards of the demonic variance of WR in $\mathcal{M}$. Omitted proofs can be found in the full version [26].

Computation of the maximal variance. It is well-known that the set of vectors $(\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\Diamond q))_{q\in T}$ of combinations of reachability probabilities for states in T that can be realized by a scheduler $\mathfrak{S}$ can be described by a system of linear inequalities (see, e.g., [18]). We provide such a system of inequalities below in equations (1) – (3). The equations use variables $x_{s.\alpha}$ for all state-action pairs (s, α) encoding the expected number of times action α is taken in state s. Setting $\mathbb{1}_{s=s_{init}} = 1$ if $s = s_{init}$ and $\mathbb{1}_{s=s_{init}} = 0$ otherwise, we require

$$x_{s,\alpha} \geq 0 \qquad \text{for all } (s,\alpha), \tag{1}$$

$$\sum_{\alpha\in Act(s)} x_{s,\alpha} = \sum_{t\in S, \beta\in Act(t)} x_{t,\beta} \cdot P(t,\beta,s) + \mathbb{1}_{s=s_{init}} \qquad \text{for all } s \in S\setminus T, \tag{2}$$

$$y_q = \sum_{t\in S, \beta\in Act(t)} x_{t,\beta} \cdot P(t,\beta,q) \qquad \text{for all } q \in T. \tag{3}$$

The variables y_q for $q \in T$ represent the probabilities that state q is reached. We can now express the expected value of WR and WR^2 via variables e_1 and e_2 via the constraints:

$$e_1 = \sum_{q \in T} y_q \cdot wgt(q) \quad \text{and} \quad e_2 = \sum_{q \in T} y_q \cdot wgt(q)^2. \tag{4}$$

The variance can now be written as a quadratic objective function in e_1 and e_2:

$$\text{maximize} \quad e_2 - e_1^2. \tag{5}$$

▶ **Theorem 4.1.** *The maximal value in objective (5) under constraints (1) – (4) is* $\mathbb{V}^{\max}_{\mathcal{M}}(\mathrm{WR})$.

Due to the concavity of the objective function, we conclude:

▶ **Corollary 4.2.** *The maximal variance* $\mathbb{V}^{\max}_{\mathcal{M}}(\mathrm{WR})$ *can be computed in polynomial time. Furthermore, there is a memoryless randomized scheduler* $\mathfrak{S}$ *with* $\mathbb{V}^{\mathfrak{S}}_{\mathcal{M}}(\mathrm{WR}) = \mathbb{V}^{\max}_{\mathcal{M}}(\mathrm{WR})$, *which can also be computed in polynomial time.*

Computation of the demonic variance. The demonic variance can also be expressed as the solution to a quadratic program. To encode the reachability probabilities for states in T under two distinct schedulers, we use variables $x_{s,\alpha}$ for all state weight pairs (s,α) and y_q for $q \in T$ subject to constraints (1) – (3) as before. Additionally, we use variables $x'_{s,\alpha}$ for all state weight pairs (s,α) and y'_q for $q \in T$ subject to the analogue constraints $(1')$ – $(3')$ using these primed variables. The maximization of the demonic variance can be expressed as

$$\text{maximize} \quad \frac{1}{2} \sum_{q,r \in T} y_q \cdot y'_r \cdot (wgt(q) - wgt(r))^2. \tag{6}$$

▶ **Theorem 4.3.** *The maximum in (6) under constraints (1) – (3), (1′) – (3′) is* $\mathbb{V}^{dem}_{\mathcal{M}}(\mathrm{WR})$.

The quadratic objective function (6) is not concave. However, it is *bilinear* and *separable*. This means that the variables can be split into two sets, the primed and the unprimed variables, such that the quadratic terms only contain products of variables from different sets and each constraint contains only variables from the same set. In general, checking whether the solution to a separable bilinear program exceeds a given threshold is NP-hard [23]. Nevertheless, solution methods tailored for bilinear programs that perform well in practice have been developed (see, e.g., [19]). Further, bilinearity allows us to conclude:

▶ **Corollary 4.4.** *There is a pair of memoryless deterministic schedulers* $\mathfrak{S}$ *and* $\mathfrak{T}$ *for* $\mathcal{M}$ *such that* $\mathbb{V}^{dem}_{\mathcal{M}}(\mathrm{WR}) = \mathbb{V}^{\mathfrak{S},\mathfrak{T}}_{\mathcal{M}}(\mathrm{WR})$.

For the complexity of the threshold problem, we can conclude an NP upper bound. Whether the computation of the demonic variance is possible in polynomial time is left open.

▶ **Corollary 4.5.** *Given* $\mathcal{M}$, *wgt and* $\vartheta \in \mathbb{Q}$, *deciding whether* $\mathbb{V}^{dem}_{\mathcal{M}}(\mathrm{WR}) \geq \vartheta$ *in in NP.*

5 Accumulated rewards

One of the most important random variables studied on MDPs are accumulated rewards: Let $\mathcal{M} = (S, Act, P, s_{init})$ be an MDP and let $rew\colon S \to \mathbb{N}$ be a reward function. We extend the reward function to paths $\pi = s_0\alpha_0 s_1 \ldots$ by $rew(\pi) = \sum_{i=0}^{\infty} rew(s_i)$. For this random variable, we prove the following result:

Main result. *The maximal variance $\mathbb{V}^{\max}_{\mathcal{M}}(rew)$ and an optimal randomized finite-memory scheduler can be computed in exponential time.*

The demonic variance $\mathbb{V}^{dem}_{\mathcal{M}}(rew)$ can be computed as the solution to a bilinear program that can be constructed in exponential time. Furthermore, there is a pair of deterministic finite-memory schedulers realizing the demonic variance.

We provide a sketch outlining the proof strategy. For a detailed exposition, see [26].

Proof sketch for the main result. It can be checked in polynomial time whether $\mathbb{E}^{\max}_{\mathcal{M}}(rew) < \infty$ [14]. If this is the case, this allows us to perform the same preprocessing as in Section 4 that removes all end components without changing the possible distributions of *rew* [14].

Bounding expected values and expectation maximizing actions. After the pre-processing, a terminal state is reached almost surely. As shown in [28], this allows to obtain a bound Q on $\mathbb{E}^{\max}_{\mathcal{M}}(rew^2)$ in polynomial time. Further, the maximal expectation $\mathbb{E}^{\max}_{\mathcal{M},s}(rew)$ from each state s can be computed in polynomial time [7, 14]. From these values, a set of *maximizing actions* $Act^{\max}(s)$ for each state s can be computed. After the preprocessing, a scheduler is expectation optimal iff it only chooses actions from these sets. If a scheduler $\mathfrak{S}$ initially chooses a non-maximizing action in a state s, the expected value $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(rew)$ is strictly smaller than $\mathbb{E}^{\max}_{\mathcal{M},s}(rew)$. We define δ to be the minimal difference between these values ranging over all starting states and non-maximizing actions. So, δ is the "minimal loss" in expected value of *rew* received by choosing a non-maximizing action.

Switching to expectation maximization. Using the values Q and δ, we provide a bound B such that any scheduler choosing a non-maximizing action with positive probability after a path π with $rew(\pi) \geq B$ cannot realize the maximal variance. Intuitively, the reason is that the influence of accumulating future rewards on the variance grows with the amount of rewards already accumulated due to the quadratic nature of variance. The bound B can be computed in polynomial time and its numerical value is exponential in the size of the input.

It follows that variance maximizing schedulers have to maximize the future expected rewards after a reward of at least B has been accumulated. Furthermore, we can show that among all expectation maximizing schedulers, a variance maximizing scheduler has to be used above the reward bound B. In [28], it is shown that a memoryless deterministic expectation maximizing scheduler $\mathfrak{U}$ that maximizes the variance among all expectation maximizing schedulers can be computed in polynomial time. So, schedulers maximizing the variance of *rew* can be chosen to behave like $\mathfrak{U}$ once a reward of at least B has been accumulated.

Quadratic program. Now, we can unfold the MDP $\mathcal{M}$ by storing in the state space how much reward has been accumulated up to the bound B. This results on an exponentially larger MDP $\mathcal{M}'$. Using the expected values $\mathbb{E}^{\mathfrak{U}}_{\mathcal{M},s}(rew)$ and the variances $\mathbb{V}^{\mathfrak{U}}_{\mathcal{M},s}(rew)$ under $\mathfrak{U}$ from each state s, we can formulate a quadratic program similar to the one for weighted reachability in Section 4 for this unfolded MDP $\mathcal{M}'$. From the solution to this quadratic program, the maximal variance and an optimal memoryless scheduler $\mathfrak{S}$ for $\mathcal{M}'$ can be extracted. Transferred back to $\mathcal{M}$, the scheduler $\mathfrak{S}$ corresponds to a reward-based finite-memory scheduler that keeps track of the accumulated reward up to bound B. As the quadratic program is convex, these computations can be carried out in exponential time.

Demonic variance. For the demonic variance, the overall proof follows the same steps. Similar to the bound B above, a bound B' can be provided such that in any pair of scheduler $\mathfrak{S}$ and $\mathfrak{T}$ realizing the demonic variance, both schedulers can be assumed to switch to the behavior of the memoryless deterministic scheduler $\mathfrak{U}$ above the reward bound B'.

Again by unfolding the state space up to this reward bound, the demonic variance can be computed via a bilinear program of exponential size similar to the one used in Section 4 for weighted reachability. Furthermore, the pair of optimal memoryless deterministic schedulers in the unfolded MDP, which can be extracted from the solution, corresponds to a pair of deterministic reward-based finite-memory schedulers in the original MDP $\mathcal{M}$. ◀

6 Conclusion

We introduced the notion of demonic variance that quantifies the uncertainty under probabilism *and* non-determinism of a random variable X in an MDP $\mathcal{M}$. As this demonic variance is at most twice as big as the maximal variance of X, we used it to define the NDS for MDPs.

The demonic variance can be used to provide new types of guarantees on the behavior of systems. A first step in this direction is the variant of Chebyshev's Inequality using the demonic variance proved in this paper. Furthermore, the demonic variance and the NDS can serve as the basis for notions of responsibility. On the one hand, such notions could ascribe responsibility for the uncertainty to non-determinism and probabilism. On the other hand, comparing the NDS from different starting states can be used to identify regions of the state space in which the non-deterministic choices are of high importance.

For weighted reachability and accumulated rewards, we proved that randomized finite-memory schedulers are sufficient to maximize the variance. For the demonic variance, even pairs of deterministic finite-memory schedulers are sufficient. While we obtained upper bounds via the formulation of the computation problems as quadratic programs, determining the precise complexities is left as future work. In the case of accumulated rewards, we restricted to non-negative rewards. When dropping this restriction, severe difficulties have to be expected as several related problems on MDPs exhibit inherent number-theoretic difficulties rendering the decidability status of the corresponding decision problems open [27].

Of course the investigation of the demonic variance and NDS for further random variables constitutes an interesting direction for future work. For practical purposes, studying also the approximability of the maximal and demonic variance is important.

Finally, In the spirit of the demonic variance, further notions can be defined to quantify the uncertainty in X if the non-determinism in two executions of $\mathcal{M}$ is not resolved independently, but information can be passed between the two executions. This could be useful, e.g., to analyze the potential power of coordinated attacks on a network. Formally, such a notion could be defined as $\sup_{\mathfrak{S}} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}\otimes\mathcal{M}}((X_1 - X_2)^2)$ where $\mathfrak{S}$ ranges over all schedulers for $\mathcal{M}\otimes\mathcal{M}$. In this context, also using an asynchronous product of $\mathcal{M}$ with $\mathcal{M}$ could be reasonable.

References

1 Erika Ábrahám and Borzoo Bonakdarpour. Hyperpctl: A temporal logic for probabilistic hyperproperties. In *International Conference on Quantitative Evaluation of Systems*, pages 20–35. Springer, 2018. doi:10.1007/978-3-319-99154-2_2.

2 Christel Baier, Krishnendu Chatterjee, Tobias Meggendorfer, and Jakob Piribauer. Entropic risk for turn-based stochastic games. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.MFCS.2023.15.

3 Christel Baier, Marcus Daum, Clemens Dubslaff, Joachim Klein, and Sascha Klüppelholz. Energy-utility quantiles. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*, volume 8430 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2014. doi:10.1007/978-3-319-06200-6_24.

4 Christel Baier, Clemens Dubslaff, Florian Funke, Simon Jantsch, Rupak Majumdar, Jakob Piribauer, and Robin Ziemek. From verification to causality-based explications (invited talk). In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 1:1–1:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.1.

5 Christel Baier, Florian Funke, and Rupak Majumdar. A game-theoretic account of responsibility allocation. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1773–1779. ijcai.org, 2021. doi:10.24963/IJCAI.2021/244.

6 Christel Baier, Florian Funke, and Rupak Majumdar. Responsibility attribution in parameterized markovian models. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 11734–11743. AAAI Press, 2021. doi:10.1609/aaai.v35i13.17395.

7 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991. doi:10.1287/moor.16.3.580.

8 Tomáš Brázdil, Václav Brožek, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Markov decision processes with multiple long-run average objectives. *Logical Methods in Computer Science*, 10, 2014. doi:10.2168/LMCS-10(1:13)2014.

9 Tomáš Brázdil, Krishnendu Chatterjee, Vojtěch Forejt, and Antonín Kučera. Trading performance for stability in Markov decision processes. *Journal of Computer and System Sciences*, 84:144–170, 2017. doi:10.1016/j.jcss.2016.09.009.

10 Hana Chockler and Joseph Y. Halpern. Responsibility and Blame: A Structural-Model Approach. *J. Artif. Int. Res.*, 22(1):93–115, October 2004. doi:10.1613/jair.1391.

11 Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Principles of Security and Trust: Third International Conference, POST*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

12 Michael R Clarkson and Fred B Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

13 EJ Collins. Finite-horizon variance penalised Markov decision processes. *Operations-Research-Spektrum*, 19(1):35–39, 1997.

14 Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In *10th International Conference on Concurrency Theory (CONCUR)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81, 1999. doi:10.1007/3-540-48320-9_7.

15 Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. Probabilistic hyperproperties of Markov decision processes. In *International Symposium on Automated Technology for Verification and Analysis, ATVA*, pages 484–500. Springer, 2020. doi:10.1007/978-3-030-59152-6_27.

16 Jerzy A Filar, Lodewijk CM Kallenberg, and Huey-Miin Lee. Variance-penalized Markov decision processes. *Mathematics of Operations Research*, 14(1):147–161, 1989. doi:10.1287/moor.14.1.147.

17 Christoph Haase and Stefan Kiefer. The odds of staying on budget. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9135 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2015. doi:10.1007/978-3-662-47666-6_19.

18 Lodewijk Kallenberg. *Markov Decision Processes.* Lecture Notes. University of Leiden, 2016.

19 Scott Kolodziej, Pedro M Castro, and Ignacio E Grossmann. Global optimization of bilinear programs with a multiparametric disaggregation technique. *Journal of Global Optimization*, 57:1039–1063, 2013. doi:10.1007/s10898-012-0022-1.

20 Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 609–618. ACM, 2018. doi:10.1145/3209108.3209176.

21 Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. Exploration in deep reinforcement learning: A survey. *Inf. Fusion*, 85(C):1–22, September 2022. doi:10.1016/j.inffus.2022.03.003.

22 Petr Mandl. On the variance in controlled Markov chains. *Kybernetika*, 7(1):1–12, 1971. URL: http://www.kybernetika.cz/content/1971/1/1.

23 Olvi L Mangasarian. The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization*, 6(2):153–161, 1995. doi:10.1007/BF01096765.

24 Shie Mannor and John N. Tsitsiklis. Mean-variance optimization in Markov decision processes. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pages 177–184, Madison, WI, USA, 2011. Omnipress. URL: https://icml.cc/2011/papers/156_icmlpaper.pdf.

25 Corto Mascle, Christel Baier, Florian Funke, Simon Jantsch, and Stefan Kiefer. Responsibility and verification: Importance value in temporal logics. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470597.

26 Jakob Piribauer. Demonic variance and a non-determinism score for Markov decision processes, 2024. doi:10.48550/arXiv.2406.18727.

27 Jakob Piribauer and Christel Baier. On Skolem-hardness and saturation points in Markov decision processes. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *LIPIcs*, pages 138:1–138:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.138.

28 Jakob Piribauer, Ocan Sankur, and Christel Baier. The variance-penalized stochastic shortest path problem. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 129:1–129:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.129.

29 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994. doi:10.1002/9780470316887.

30 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.

31 Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In Frank Pfenning, editor, *16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013. doi:10.1007/978-3-642-37075-5_23.

32 Tom Verhoeff. Reward variance in Markov chains: A calculational approach. In *Proceedings of Eindhoven FASTAR Days*. Citeseer, 2004.

33 Vahid Yazdanpanah, Mehdi Dastani, Wojciech Jamroga, Natasha Alechina, and Brian Logan. Strategic Responsibility Under Imperfect Information. In *Proc. of the 18th Intern. Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 592–600. AAMAS Foundation, 2019. URL: http://dl.acm.org/citation.cfm?id=3331745.

Computational Model for Parsing Expression Grammars

Alexander Rubtsov
HSE University, Moscow, Russia
MIPT, Moscow, Russia

Nikita Chudinov
Google, Zürich, Switzerland

Abstract

We present a computational model for Parsing Expression Grammars (PEGs). The predecessor of PEGs top-down parsing languages (TDPLs) were discovered by A. Birman and J. Ullman in the 1960-s, B. Ford showed in 2004 that both formalisms recognize the same class named Parsing Expression Languages (PELs). A. Birman and J. Ullman established such important properties like TDPLs generate any DCFL and some non-context-free languages like $a^nb^nc^n$, a linear-time parsing algorithm was constructed as well. But since this parsing algorithm was impractical in the 60-s TDPLs were abandoned and then upgraded by B. Ford to PEGs, so the parsing algorithm was improved (from the practical point of view) as well. Now PEGs are actively used in compilers (eg., Python replaced LL(1)-parser with a PEG one) so as for text processing as well. In this paper, we present a computational model for PEG, obtain structural properties of PELs, namely proof that PELs contain Boolean closure of regular closure of DCFLs and PELs are closed over left concatenation with regular closure of DCFLs. We present an extension of the PELs class based on the extension of our computational model. Our model is an upgrade of deterministic pushdown automata (DPDA) such that during the pop of a symbol it is allowed to return the head to the position of the push of the symbol. We provide a linear-time simulation algorithm for the 2-way version of this model, which is similar to the famous S. Cook linear-time simulation algorithm of 2-way DPDA.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases PEG, formal languages, pushdown automata, two-way pushdown automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.80

Related Version *Full Version*: `https://arxiv.org/abs/2406.14911` [13]

Funding *Alexander Rubtsov*: This paper was prepared within the framework of the HSE University Basic Research Program. Supported in part by RFBR grant 20–01–00645.

Acknowledgements Authors thank Mikhail Iumanov for helpful discussions and anonymous referees for the helpful comments.

1 Introduction

We present a computational model for Parsing Expression Grammars (PEGs) presented by B. Ford in [6]. The predecessor of PEGs top-down parsing languages (TDPLs) was discovered by A. Birman and J. Ullman in the 1960s (so as generalized TDPLs) [4]. While the PEGs formalism has more operations, it has the same power as TDPLs and generalized TDPLs which was shown by B. Ford in [7]. We refer to the class of languages generated by PEGs (and TDPLs, and generalized TDPLs) as *Parsing Expression Languages* (PELs).

Little is known about the structural properties of PELs. From the 60's it is known that PELs contain DCFLs as a subclass and some non-context-free languages like $a^nb^nc^n$ as well. A linear-time parsing algorithm (in RAM) had been constructed for TDPLs, but it was

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 80; pp. 80:1–80:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

impractical in the 1960s since it required too much memory for memoization and TDPLs had been abandoned. B. Ford upgraded the TDPLs formalism to PEGs and presented a linear-time practical algorithm in 2002 [6]. Now PEGs are being actively used in compilers (eg., Python replaced an LL(1)-parser with a PEG one [16]) so as for text processing as well. In this paper, we present a computational model for PELs and obtain some interesting properties for this class, analyze (some of) its subclasses, and generalize the PELs class as well.

A computational model for PELs was presented in [12], but this model significantly differs from classical models of computations, so it is hard to clarify the place of PELs among known classes of formal languages, based on this model. So we present a simpler and more convenient model that discovers the place of PEGs in the variety of formal language classes. Namely, the computational model is a modified deterministic pushdown automaton (DPDA) that puts to the stack a symbol with the pointer of the head's position on the tape (from which the push has been performed). During the pop, the automaton has two options: either leave the head in the current position or move the head to the position stored in the pointer (retrieved during the pop of the symbol). We call this model a deterministic pointer pushdown automaton (DPPDA). This description of PELs from the automata point of view helped us to obtain other important results not only for the PELs but for the general area of formal languages as well. Namely, we prove that boolean closure of regular closure of DCFLs is linear-time recognizable (in RAM), what extends the nontrivial result by E. Bertsch and M.-J. Nederhof [3] that regular closure of DCFLs is linear-time recognizable.

To describe our results we shall mention the following important results in the area of formal languages and automata theory. Donald Knuth invented LR(k) grammars that describe DCFLs for $k \geq 1$ and were widely used in practice. It is easier to design an LL(k) grammar for practical purposes, so despite the power of LR, LL grammars are widely used for parsing (and some artificial modification of recursive descent parsing as well). Top-down parsing languages (TDPLs, predecessor of PEGs) cover LL(1) grammars and even contain DCFLs as a subclass, but their linear-time parsing algorithm was impractical in the 1970s, so TDPLs had been abandoned till B. Ford upgraded them to PEGs and presented a practically reasonable linear-time parser (Packrat). So, linear-time recognizable classes of formal languages are used in compilers, and LR (DCFLs) parsers now compete with PEGs which cover a wider class of formal languages that is almost undiscovered. There are no comprehensive results on the structure of PELs, so we make a contribution to this open question. Another wide linear-time recognizable class of formal languages is languages recognizable by two-way deterministic pushdown automata (2DPDA). S. Cook obtained in [5] a famous linear-time simulation algorithm for this model. There also was an amazing story about how D. Knuth used S. Cook's algorithm to discover the Knuth-Morris-Pratt algorithm ([10], Section 7).

We modify 2DPDA in the same way as we did for DPDA: we add symbols to stack with a pointer that allows returning the head to the cell from which the push had been performed. S. Cook's linear-time simulation algorithm applies to this model as well (with a little modification). So we extend the important class of formal languages (recognizable by 2DPDAs) preserving linear-time parsing. This extension can be used to generalize PEGs. Also, this algorithm provides another approach to linear time recognition of languages generated by PEGs described via DPPDAs. Note that there are not many structural results about PELs. Moreover, even equivalence of TDPLs and generalized TDPLs (with PEGs) had been proved by B. Ford [7] decades after these classes had been invented. In our opinion, one of the reasons for that is that TDPL-based formalisms are hard. So even the proof of inclusion DCFLs in PELs [4] is complicated, while it directly follows from the equivalence of PEGs with our model.

So we hope that our model will raise interest in investigations of PELs and will help with these investigations as well. Our results also clarify the place of another interesting result (we also improved it, as described below). It was shown by E. Bertsch and M.-J. Nederhof [3] that regular closure of DCFLs is linear-time recognizable. We show that this class is recognizable by DPPDAs which simplifies the original proof [3] and shows the place of this class in the formal languages classes.

There are many linear-time recognizable classes of formal languages. Recently Rubtsov showed [14] that Hibbard's hierarchy (the subclass of CFLs) is linear-time recognizable. So there are many open questions related to the systematization of linear time recognizable classes of formal languages and particularly the relation of Hibbard's hierarchy with languages recognizable by 1-2 DPPDAs.

1.1 Results

In this paper, we present a new computational model DPPDA which is equivalent to PEGs. We also consider the two-way model 2DPPDA and provide a linear time simulation algorithm for this model following S. Cook's construction. Via DPPDA we show that the PEGs class is closed over left concatenation with regular closure of DCFLs, so PELs contain the regular closure of DCFLs as a subclass. With the linear-time simulation algorithm for 2DPPDA, we obtain another linear-time recognition algorithm for the regular closure of DCFLs and since PELs are closed over Boolean operation we prove that the Boolean closure of regular closure of DCFLs is linear-time recognizable. Note that the last result not only generalizes well known result of linear-time recognizability of regular closure of DCFLs [3], but also our proof is significantly simpler as well.

The full version of this paper is available on arXiv [13]. We put the reference to [13] when the proof is omitted due to the space limitations.

1.2 Basic Notation

We follow the notation from [9] on formal languages, especially on context-free grammars (CFGs) and pushdown automata. We denote the input alphabet as Σ and its elements (letters, terminals) are denoted by small letters $a, b, c, \ldots$, while letters w, x, y, z denote words. The empty word is denoted by ε. We denote nonterminals N by capital letters $A, B, C, \ldots$, and X, Y, Z can be used for both nonterminals and terminals. The axiom is denoted by $S \in N$. Words over the alphabet $N \cup \Sigma$ are called sentential forms and are denoted by small Greek letters.

1.3 Informal Description of PEGs

The formal definition of PEGs is not well intuitive, so we begin with an informal one that clarifies a simple idea behind this formal model. The intuition behind PEGs lies in recursive descent parsing.

One of the parsing methods for CF-grammars is a recursive descent parsing that is a process when the derivation tree is built top-down (starting from the axiom S) and then each nonterminal is substituted according to the associated function. A rollback is possible as well, where by rollback we mean the replacement of one production rule by another or even the replacement of the rule higher above the current node with the deletion of subtrees. This method is very general and we do not go deep into details. For our needs, we describe a recursive descent parsing of LL(1) grammars and its modification that defines PEGs.

For LL(1) grammar, the following assertion holds. Fix a leftmost derivation of a word $w\triangleleft = uav\triangleleft$ and let $uA\alpha\triangleleft$ be a derivation step (here $\triangleleft$ is a right end marker of the input). The next leftmost derivation step is determined by the nonterminal A and the terminal a, so the rule is the function $R(A, a)$. So, the recursive descent algorithm for an LL(1)-parser is as follows. An input $w\triangleleft$ is written in the one-way read-only tape called the *input tape*. The pointer in the (constructing) derivation tree points to the leftmost nonterminal node (without children), initially the axiom S. This node is replaced according to the function R. In the fixed above derivation step $uA\alpha\triangleleft$ the pointer is over the nonterminal A, $R(A, a) = xB\beta$, where $A \to xB\beta$ is a grammar rule. So, $xB\beta$ is glued into A as a subtree, x is a prefix of av and the head of the input tape moves while scanning x. If $R(A, a)$ does not contain a nonterminal, then (after replacement) the tree is traversed via DFS until the next (leftmost!) nonterminal is met. Each terminal during this traversal shifts the head of the input tape. If the symbol under the head differs from the traversed terminal, the input word is rejected. We illustrated the described process in Fig. 1. Note that u, x, α, β are the subtrees and u, v, x, v' in fact occupies several cells of the input tape.

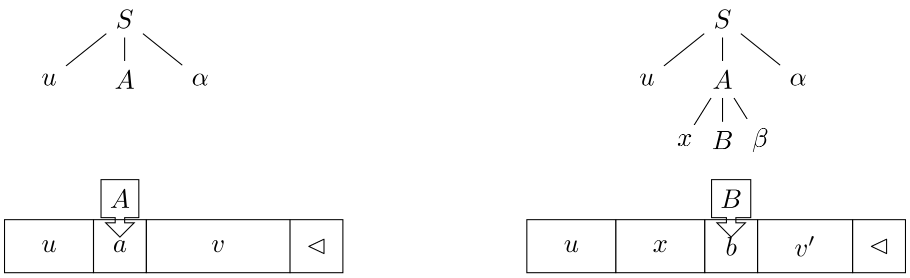


Figure 1 Example of LL(1) recursive descent parsing.

So now we move to the description of PEGs via modification of recursive descent parsing. In the first example, we will provide similar PEG and CFG (Fig. 2) and explain their similarity and differences.

PEG	CFG
$S \leftarrow AB \,/\, BC$	$S \to AB \mid BC$
$A \leftarrow aA \,/\, a$	$A \to aA \mid a$
$B \leftarrow abb \,/\, b$	$B \to abb \mid b$
$C \leftarrow cC \,/\, \varepsilon$	$C \to cC \mid \varepsilon$

Figure 2 PEG and CFG for comparison.

PEGs look similar to context-free grammars, but the meaning of almost all concepts are different, therefore the arrow ← is used to separate the left part of a rule from the right part. The difference comes from the following approach to recursive descent parsing. We describe the PEG via the transformation of the CFG. Let us order all the rules of the CFG for each nonterminal. During recursive descent parsing, we will try each rule according to this order. If a failure happens, let us try the next rule in the order. If the last rule leads us to the failure too, propagate the failure to the parent and try using the next rule in the order on the previous tree level. So, that is the reason why all right-hand sides of the rules in PEG are separated by the delimiter /, but not by |. The order of rules in PEGs matters, unlike CFGs. Consider the parsing (Fig. 3) of the word aab by the PEG defined on Fig. 2.

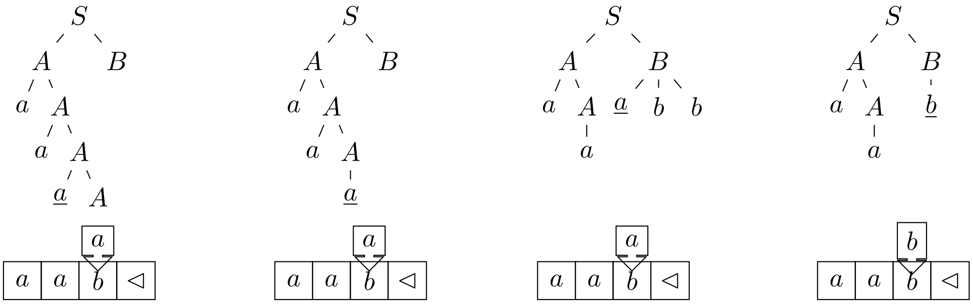


Figure 3 Parsing of aab by PEG.

The rule $A \leftarrow aA$ is applied while the content of the input tape matches the crown (the leafs) of the tree. So, when the last application is unsuccessful, it is replaced by the following rule $A \leftarrow a$ which is unsuccessful too. So failure signal goes to the level above and the second rule $A \leftarrow aA$ is replaced by $A \leftarrow a$. After that, the control goes to the nonterminal B for which firstly the rule $B \leftarrow abb$ is applied, but since it leads to the failure, finally the rule $B \leftarrow b$ is applied and it finishes the parsing since the whole word has been matched.

So PEGs are similar to CFGs since they share the idea of recursive descent parsing. But the difference is significant. Since all the rules for each nonterminal are ordered, the classical notion of concatenation does not apply to PEGs. We cannot say that if a word u is derived from A and v is derived from B, then uv is derived from AB as explained below. In the PEG example above, a word abb is never derived from B because A from AB will always parse all a's from the input. Note that the failure during the parsing occurs only because of a mismatch. So, the input $abbc$ will be parsed by the PEG as following. The prefix ab will be successfully parsed by AB and by S as well, but since the whole word has not been parsed, the input is rejected. Since there was no failure, the rule $S \leftarrow AB$ was not replaced by $S \leftarrow BC$. So the word $abbc$ is not accepted by the PEG while it is derived from the CFG.

▶ Remark 1. It is an open question, whether PELs are closed over concatenation.

Note that the patterns of iteration $A \leftarrow aA\,/\,a$ and $C \leftarrow cC\,/\,\varepsilon$ work in a greedy way. In the case of concatenation Ce (with an expression e), all c's from the prefix of the input would be parsed by C.

In the considered example we have not mentioned an important PEG's operation. There is a unary operator **!** that is applied as follows. In the case **!**e the following happens. Firstly the parsing goes to the expression e. If e parsed the following input successfully (i.e., a subtree for e that matches the prefix of the unprocessed part of the input has been constructed without a failure), then **!**e produces failure. If a failure happens, then **!**e is considered to parse the empty word ε and the parsing process continues. For example, consider the following PEG:

$$S \leftarrow A(\mathbf{!}C)\,/\,B \quad A \leftarrow aAb\,/\,\varepsilon \quad B \leftarrow aBc\,/\,\varepsilon \quad C \leftarrow a\,/\,b$$

($\mathbf{!}C$) guarantees that if $A(\mathbf{!}C)$ finished without failure, then it parsed the whole input. So, in the case of the input a^nb^n for $n \geq 0$, the input will be parsed by $A(\mathbf{!}C)$ and there will be no switch to the rule $S \leftarrow B$. For any other input, the parsing of $A(\mathbf{!}C)$ fails and the rule is switched to $S \leftarrow B$. So, this PEG generates the language $\{a^nb^n \mid n \geq 0\} \cup \{a^nc^n \mid n \geq 0\}$.

Another common use of the operator **!** is its double application that has its name: $\&e = \mathbf{!}(\mathbf{!}e)$. This construction checks whether the prefix of the (unprocessed part of the) input matches e: in the positive case, & parses an empty word and computation continues, in the negative case, it returns failure. So & acts similarly to **!**, but the conditions are flipped. Consider the following example:

$$S \leftarrow (\&(Ac))BC \quad A \leftarrow aAb \,/\, \varepsilon \quad B \leftarrow aB \,/\, a \quad C \leftarrow bCc \,/\, \varepsilon$$

This PEG checks that the input has the prefix $a^n b^n c$ and then parses the input if it has the form $a^+ b^n c^n$, so the PEG generates the language $\{a^n b^n c^n \mid n \geq 1\}$.

So it is known that PEGs generate non CFLs and it is still an open question whether PEGs generate all CFLs. The conditional answer is no: there exists a linear-time parsing algorithm for PEG, while the work of L. Lee [11] and Abboud et al. [1] proves that it is very unlikely for CFLs due to theoretical-complexity assumptions: any CFG parser with time complexity $O(gn^{3-\varepsilon})$, where g is the size of the grammar and n is the length of the input word, can be efficiently converted into an algorithm to multiply $m \times m$ Boolean matrices in time $O(m^{3-\varepsilon/3})$. Note that this conditional result shows that it is unlikely that 2DPPDAs recognize all CFLs as well.

2 Formal Definition of PEGs

Our definition slightly differs from the standard definition of PEG from [7] (Section 3) due to technical reasons. We discuss the difference after the formal definition.

▶ **Definition 2.** *A parsing expression grammar G is defined by a tuple (N, Σ, P, S), where N is a finite set of symbols called* nonterminals*, Σ is a finite input alphabet (a set of* terminals*), $N \cap \Sigma = \varnothing$, $S \in N$ is* the axiom*, and P is a set of* production rules *of the form $A \leftarrow e$ such that each nonterminal $A \in N$ has the only corresponding rule, and e is an expression that is defined recursively as follows. The empty word ε, a terminal $a \in \Sigma$, and a nonterminal $A \in N$ are expressions. If e and e' are expressions, then so are (e) which is equivalent to e, a* sequence ee'*, a* prioritized choice $e \,/\, e'$*, a* not predicate $\mathbf{!}e$*. We assume that **!** has the highest priority, the next priority has the sequence operation and the prioritized choice has the lowest one. We denote the set of all expressions over G by E_G or by E if the grammar is fixed.*

To define the language generated by a PEG G we define recursively a partial function $R : E \times \Sigma^* \to (\Sigma^* \cup \{\mathbf{F}\})$ that takes as input the expression e, the input word w, and if $R(e, w) = s \in \Sigma^*$, then s is the suffix of $w = ps$ such that the prefix p has been parsed by e during the processing of w; if $R(e, w) = \mathbf{F}$ it indicates a failure that happens during the parsing process. So, the function R is defined recursively as follows:

- $R(\varepsilon, w) = w$, $R(a, as) = s$, $R(a, bs) = \mathbf{F}$ (where $a \neq b$)
- $R(e_1 e_2, w) = R(e_2, R(e_1, w))$ if $R(e_1, w) \neq \mathbf{F}$, otherwise $R(e_1 e_2, w) = \mathbf{F}$
- $R(A, w) = R(e, w)$, where $A \leftarrow e \in P$
- $R(e_1 \,/\, e_2, w) = R(e_1, w)$ if $R(e_1, w) \neq \mathbf{F}$, otherwise $R(e_1 \,/\, e_2, w) = R(e_2, w)$
- $R(\mathbf{!}e, w) = w$ if $R(e, w) = \mathbf{F}$, otherwise $R(\mathbf{!}e, w) = \mathbf{F}$

Note that $R(e, w)$ is undefined if during the recursive computation, R comes to an infinite loop. In fact, we will never meet this case because for each PEG there exists an equivalent form for which R is a total function (see Subsection 2.1).

We say that a PEG G *generates* the language $L(G) = \{w \mid R(S, w) = \varepsilon\}$; if $R(S, w) = \varepsilon$ we say that w is *generated* by G.

2.1 Difference with other standard definitions and forms of PEGs

Note that our definition of $L(G)$ differs from [7] (Section 3). The difference is about the operations allowed in PEG and the acceptance condition as well. In this subsection, we explain the difference and provide an overview of different forms of PEGs.

In the case of practical parsing, it is convenient to have more operations in the definition of PEG, but theoretically, it is more convenient to have fewer operations for the sake of the proofs' simplicity. In [7] B. Ford investigated different forms of PEGs and proved their equivalence, so as the equivalence with (generalized) top-down parsing languages. We begin our overview with operations that are so easy to express via operations from our definitions that they can be considered (as programmers say) syntactic sugar:

- **Iterations:** e^* is equivalent to $A \leftarrow eA \,/\, \varepsilon$; $e^+ = ee^*$
- **Option expression:** $e?$ is equivalent to $A \leftarrow e \,/\, \varepsilon$
- **And predicate:** $\&e = \mathbf{!}(\mathbf{!}e)$
- **Any character:** $\bullet = a_1 \,/\, a_2 \,/\, \dots \,/\, a_k$ where $\Sigma = \{a_1, \dots, a_k\}$
- **Failure:** $\mathbf{F} = \mathbf{!}\varepsilon$ (we use the same notation as for the failure result)

We can use these constructions below. In this case, the reader can assume that they are reduced to the operations from Definition 2 as we have described.

So by adding to the definition (or removing) syntactic sugar operations, one obviously obtains an equivalent definition (in terms of recognizable languages' class). Now we move to the nontrivial cases proved in [7].

A PEG G is *complete* if for each $w \in \Sigma^*$ the function $R(S, w)$ is defined. A PEG G is *well-formed* if it does not contain directly or mutually left-recursive rules, such as $A \leftarrow Aa \,/\, a$. It is easy to see that a well-formed grammar is complete. It was proved in [7] that each PEG has an equivalent well-formed one and the algorithm of the transformation had been provided as well. So from now on we assume that each PEG in our constructions is well-formed. Note that most PEGs that are used in practice are well formed by construction.

Another interesting result from [7] is that each PEG has an equivalent one without predicate **!**. Despite this fact, we decided to include **!** in our definition since unlike substitutions for syntactical sugar operations, removing **!** predicate requires significant transformations of the PEG. Since **!** predicate is widely used in practice and it does not affect our constructions, by including **!** in the definition we achieve the constructions that can be used in practice.

As we have already mentioned our condition of the input acceptance also differs from [7]. We used the provided approach since if $R(S, w) = \varepsilon$ we can reconstruct the parsing tree with the root S that generates w. We use this property for the transformation of PEG to the computational model and the inverse transformation as well. Firstly, in [7] there is no axiom in PEG, but there is a starting expression e_S. This difference is insignificant since one can state $e_S = S$ and $S \leftarrow e_S$ for the opposite direction. A PEG from [7] generates the input w if $R(e_S, w) \neq \mathbf{F}$, so $R(e_S, w) = y$, where $w = xy$. So to translate PEG from [7] to ours one needs to set $S \leftarrow e_S(\bullet)^*$. The transformation in the other direction is $e_S = S(\mathbf{!}\bullet)$.

3 Definition of the Computational Model

We call our model *deterministic pointer pushdown automata* (DPPDA). We consider a one-way model (1DPPDA or just DPPDA) as a restricted case of a two-way model (2DPPDA), so we define the two-way model only.

▶ **Definition 3.** *A* 2*-way deterministic pointer pushdown automata M is defined by a tuple*

$$\langle Q, \Sigma_{\triangleright\triangleleft}, \Gamma, F, q_0, z_0, \delta \rangle$$

- Q *is the finite set of automaton states.*
- $\Sigma_{\triangleright\triangleleft} = \Sigma \cup \{\triangleright, \triangleleft\}$*, where* Σ *is the finite input alphabet and* $\triangleright, \triangleleft$ *are the endmarkers. The input has the form* $\triangleright w \triangleleft$, $w \in \Sigma^*$.
- Γ *is the alphabet of the pushdown storage.*
- $F \subseteq Q$ *is the set of the final states.*
- $q_0 \in Q$ *is the initial state.*
- $Z_0 \in \Gamma$ *is the initial symbol in the pushdown storage.*
- δ *is the partial transition function defined as* $\delta : Q \times \Sigma_{\triangleright\triangleleft} \times \Gamma \to Q \times \Gamma_\varepsilon^* \times \{\leftarrow, \downarrow, \uparrow, \to\}$*, where* $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$*. Moreover, if* $\delta(q, a, z) = (q', \alpha, \uparrow)$*, then* $\alpha = \varepsilon$.

To define and operate with configurations of automata we introduce some notation. We denoted by $\alpha \times \vec{i} = (Z_m, i_m), \ldots, (Z_0, i_0)$ the zip of the sequences α and $\vec{i}$, which are of the same length by the definition. A right associative operation $x : \vec{l}$ prepends an element x to the beginning of the vector $\vec{l}$ (we adopt this operator from Haskell programming language). E.g., if $\vec{l} = 1, 2, 3$ and $\vec{r} = 2, 3$, we write $\vec{l} = 1 : \vec{r}$.

A configuration of M on a word w is a quadruple $c \in Q \times (\Gamma \times I)^* \times I$, where $I = \{0, \ldots, |w| + 1\}$; we refer to $w_i, i \in I$ as the i-th input symbol; $w_0 = \triangleright$, $w_{|w|+1} = \triangleleft$. A configuration $c = (q, \alpha \times \vec{i}, j)$ has the following meaning. The head of 2DPPDA M is over the symbol w_j in the state q; the pushdown contains $\alpha = Z_m Z_{m-1} \cdots Z_0$ (the stack grows from right to left) and there is also additional information vector $\vec{i} = i_m, i_{m-1}, \ldots, i_0$, $i_k \in I$ such that Z_k was pushed to the pushdown store when the head was over the i_k-th cell.

The automaton's move is defined via the relation $\vdash$ as follows. Let $\delta(q, a, Z_n) = (q', \beta, d)$. The relation

$$(q, Z_n\alpha \times i_n : \vec{i}, j) \vdash (q', \alpha' \times \vec{i}', j')$$

is defined according to the following case analysis.

- If $d \in \{\leftarrow, \downarrow, \to\}$, then $j' = j - 1$, $j' = j$, $j' = j + 1$ respectively. The cases $a = \triangleright$, $d = \leftarrow$ and $a = \triangleleft$, $d = \to$ are forbidden.
- If $\beta = \varepsilon$ and $d \in \{\leftarrow, \downarrow, \to\}$, then $\alpha' = \alpha$, $\vec{i}' = \vec{i}$
- If $\beta = \varepsilon$ and $d = \uparrow$, then $\alpha' = \alpha$, $\vec{i}' = \vec{i}$, $j' = i_n$
- If $\beta = X_1 \cdots X_k$, $k > 0$, then $\alpha' = \beta Z_n \alpha$, $\vec{i}' = \underbrace{j' : j' : \cdots : j'}_{k} : i_n : \vec{i}$

The initial configuration is $(q_0, Z_0 \times 0, 0)$ and an accepting configuration is $(q_f, \varepsilon \times (), |w| + 1)$, where $q_f \in F$ and by $()$ we have denoted the empty sequence of integers. I.e., M reaches the right end marker $\triangleleft$ empties the stack and finishes the computation in an accepting state. Formally, a word w is accepted by M if there exists a computational path from the initial configuration to an accepting one.

In the case of 1DPPDA (or just DPPDA), the moves $\leftarrow$ are forbidden.

3.1 Properties of DPPDA

Now we discuss the properties of the model and provide some shortcuts for the following needs. Note that each move of a 2DPPDA is either push- or pop-move due to the sake of convenience in the proofs (induction invariants are simpler). At the same time, in constructions, it is convenient to have right, left, and even stay moves that do not change the stack. So we add moves $\hookrightarrow$, $\hookleftarrow$, and $\downarrow$ that are syntactic sugar for such moves. So, when we write $\delta(q, a, z) = (p, \hookrightarrow)$, we mean the sequence of moves:

$$\delta(q, a, z) = (p', Z', \to); \forall \sigma \in \Sigma_{\triangleright\triangleleft} : \delta(p', \sigma, Z') = (p, \varepsilon, \downarrow).$$

The construction for $\hookleftarrow$ and $\downarrow$ are similar.

Due to the definition of δ, a DPPDA can move only if the stack is non-empty and since each move is either push or pop, we have that Z_0 lies at the bottom of the stack till the last move of a computation or even after the last move in the case of unsuccessful computation. In the case of a successful computation, Z_0 is popped at the last move.

4 Equivalence of DPPDAs and PEGs

In this section, we provide an algorithm that transforms a PEG into a DPPDA. The algorithm of the inverse transformation provided in [13] due to space limitations. Our construction is similar to the well-known proof of equivalence between CFGs and DPDA for CFLs, but since both DPPDAs and PEGs are more complicated than DPDAs and CFLs, our constructions are technically harder. We refer the reader to [15] for the detailed explanation of the proof idea, where it was provided for CFLs (so as the proof for CFLs as well).

In this section we assume that PEGs have a special form. We call it Chomsky's normal form since it is similar to such a form for CFGs.

▶ **Definition 4.** *A PEG G has a* Chomsky normal form *if the axiom S never occurs on the right side of the rules and the rules are of the following form:*

$$A \leftarrow B\,/\,C, \quad A \leftarrow BC, \quad A \leftarrow\, !B, \quad A \leftarrow a, \quad A \leftarrow \varepsilon.$$

▶ **Lemma 5** ([13]). *Each PEG G has an equivalent PEG G' in Chomsky's normal form which is complete if so was G.*

▶ **Theorem 6.** *For a PEG G there exists an equivalent DPPDA M.*

Proof. We assume that G is a well-formed PEG in a Chomsky normal form (by Lemma 5). We construct an equivalent DPPDA $M = \langle Q, \Sigma_{\rhd\lhd}, \Gamma, \{q_f\}, q_0, Z_0, \delta\rangle$ by the PEGs description. We formally describe δ on Fig. 4; we do not provide a full list of states Q and pushdown alphabet Γ since most of the states and symbols depend on rules listed in δ's construction and can be easily restored from it. Since the construction is straightforward, we describe here only the main details.

The DPPDA M simulates the parsing process of a PEG G on the input w. Firstly M performs a series of technical moves to come from the initial configuration to the initial simulation configuration:

$$(q_0, Z_0 \times 0, 0) \overset{*}{\vdash} (q, SZ_0 \times (1:0), 1),$$

where q is the *main work state* and S is the axiom of the PEG.

During the simulation the following invariants hold. Below A is a nonterminal of the PEG.

1. If the automaton is in the main work state q and on the top of the stack is the pair $A \times i$, then the head is over the cell i.
2. If the head is over the cell $r+1$ in a state $q_{A_\pm}$ (hereinafter $q_{A_\pm} \in \{q_{A_+}, q_{A_-}\}$) and the topmost symbol had been added at the position l, then it means the following.

   q_{A_+} A subword $s = w_l \cdots w_r$ would be parsed by the PEG from A (starting from the position l); when $r+1 = l$, we have $s = \varepsilon$. In the other direction: if the PEG parses $w_l \cdots w_r$ from A starting from the position l, then the DPPDA that starts computation from the position l in the main work state q with A on the top on the stack finishes at the position $r+1$ with (the same) A on the top of the stack, i.e.,

$$(q, A\alpha \times l : \vec{i}, l) \overset{*}{\vdash} (q_{A_+}, A\alpha \times l : \vec{i}, r+1).$$

q_{A_-} After the PEG started parsing from A from the position l, the computation ended up with a failure at some point in the case of q_{A_-}. In the other direction: if the PEG fails, then for some $r \geq l-1$:

$$(q, A\alpha \times l : \vec{i}, l) \overset{*}{\vdash} (q_{A_-}, A\alpha \times l : \vec{i}, r+1).$$

DPPDA M accepts the input only if the head reaches the symbol $\lhd$ in the state q_{S+} (note that the axiom does not occur on the right side of the rules). Formally, we add the rule

$$\delta(q_{S_+}, \lhd, Z_0) = (q_f, \varepsilon, \downarrow),$$

where q_f is the only final state of the DPPDA. So, from the invariant it follows that the DPPDA accepts the input iff the PEG parses the input.

The rest of the construction is the delta's description in Fig. 4. The proof is a straightforward induction on the recursion depth of the PEGs computation. So we describe the behavior of the automaton corresponding to formal construction in two main cases and check that the invariants hold (the remaining cases are simple).

Construction of δ

We denote by $Z \in \Gamma$ and $\sigma \in \Sigma_{\rhd\lhd}$ arbitrary symbols. The rules are grouped with respect to the PEG's operations. Note that the states q, $q_{A_\pm}$ are the same for all rules, while other states and stack symbols depend on the rule, i.e., stack symbols A_1's from different rules are different even if they correspond to the same nonterminal A. When we use nonterminals (and states) with signs $\pm$ or $\mp$, the signs have corresponding matching, i.e., if in a rule we have $A_\pm$ and $B_\mp$, then when $A_\pm = A_+$, $B_\mp$ equals to B_- and when $A_\pm = A_-$, $B_\mp$ equals to B_+.

0. General rules
- $\delta(q_0, \rhd, Z_0) = (q_0, \hookrightarrow)$; $\forall \sigma' \in \Sigma \cup \{\lhd\} : \delta(q_0, \sigma', Z_0) = (q, S, \downarrow)$;
- $\delta(q_{A_\pm}, \sigma, A) = (q_{A_\pm}, \varepsilon, \downarrow)$; $\delta(q_{S_+}, \lhd, Z_0) = (q_f, \varepsilon, \downarrow)$.

1. $A \leftarrow BC$
- $\delta(q, \sigma, A) = (q, BA_1, \downarrow)$
- $\delta(q_{B_+}, \sigma, A_1) = (q, CA_2, \downarrow)$;
- $\delta(q_{B_-}, \sigma, A_1) = (q_{A_-}, \varepsilon, \uparrow)$;
- $\delta(q_{C_+}, \sigma, A_2) = (q'_{A_2}, \varepsilon, \downarrow)$;
- $\delta(q'_{A_2}, \sigma, A_1) = (q_{A_+}, \varepsilon, \downarrow)$;
- $\delta(q_{C_-}, \sigma, A_2) = (q'_{A_{2-}}, \varepsilon, \uparrow)$;
- $\delta(q'_{A_{2-}}, \sigma, A_1) = (q_{A_-}, \varepsilon, \uparrow)$.

2. $A \leftarrow B \,/\, C$
- $\delta(q, \sigma, A) = (q, BA_1, \downarrow)$
- $\delta(q_{B_+}, \sigma, A_1) = (q_{A_+}, \varepsilon, \downarrow)$
- $\delta(q_{B_-}, \sigma, A_1) = (q_{A_2}, \varepsilon, \uparrow)$
- $\delta(q_{A_2}, \sigma, Z) = (q, CA_2, \downarrow)$
- $\delta(q_{C_+}, \sigma, A_2) = (q_{A_+}, \varepsilon, \downarrow)$
- $\delta(q_{C_-}, \sigma, A_2) = (q_{A_-}, \varepsilon, \uparrow)$;

3. $A \leftarrow \varepsilon$
- $\delta(q, \sigma, A) = (q_{A_+}, \lcurvearrowdown)$

4. $A \leftarrow \mathbf{!}B$
- $\delta(q, \sigma, A) = (q, BA_1, \downarrow)$;
- $\delta(q_{B_\pm}, \sigma, A_1) = (q_{A_\mp}, \varepsilon, \uparrow)$

5. $A \leftarrow a$
- $\delta(q, a, A) = (q_{A_+}, \hookrightarrow)$;
- $\delta(q, b, A) = (q_{A_-}, \lcurvearrowdown)$, here $b \neq a$;

Figure 4 Construction of δ by the PEG G.

Each rule is applied to a configuration of the form $(q, A\alpha \times l : \vec{i}, l)$. In the first case (of concatenation) the automaton pushes the auxiliary symbol A_1 at the same position that A has been pushed (since the invariant 1 holds) and then pushes B. If it reached a configuration of the form $(q_{B_+}, BA_1A\alpha \times l : l : l : \vec{i}, r' + 1)$, then B has successfully parsed the subword $w_l \cdots w_{r'}$ due to invariant 2, then B is popped due to General rules and DPPDA pushes C at the position $r' + 1$ and goes to the main work state q. If then the DPPDA reaches a configuration of the form $(q_{C_+}, CA_2A_1A\alpha \times (r'+1) : (r'+1) : l : l : \vec{i}, r+1)$ we have that the PEG parsed $w_{r'+1} \cdots w_r$ from C and after the sequences of technical pops the automaton comes to the configuration $(q_{A_+}, A\alpha \times l : \vec{i}, r + 1)$ that proves that invariant 2-q_{A_+} holds (the arguments for the other direction are similar).

In the case of reaching the configuration $(q_{C_-}, CA_2A_1A\alpha \times (r'+1) : (r'+1) : l : l : \vec{i}, r+1)$ or $(q_{B_-}, BA_1A\alpha \times l : l : l : \vec{i}, r' + 1)$ earlier, the sequence of pops lead the DPPDA to the configuration $(q_{A_-}, A\alpha \times l : \vec{i}, l)$ that proves that invariant 2-q_{A_-} holds (the arguments for the other direction are similar).

The case of the ordered choice is similar to the case of concatenation. The difference is, that in the case of a configuration $(q_{B_+}, BA_1A\alpha \times l : l : l : \vec{i}, r + 1)$, the automaton reaches the configuration $(q_{A_+}, A\beta \times l : \vec{i}, r + 1)$ via the technical moves, and in the case of $(q_{B_-}, BA_1A\alpha \times l : l : l : \vec{i}, r + 1)$ the automaton reaches the configuration $(q, CA_2A\alpha \times l : l : l : \vec{i}, l)$ after which

$$\text{either } (q, CA_2A\alpha \times l : l : l : \vec{i}, l) \vdash^* (q_{C_+}, CA_2A\alpha \times l : l : l : \vec{i}, r+1) \vdash (q_{A_+}, A\alpha \times l : \vec{i}, r+1),$$

$$\text{or } (q, CA_2A\alpha \times l : l : l : \vec{i}, l) \vdash^* (q_{C_-}, CA_2A\alpha \times l : l : l : \vec{i}, r+1) \vdash^* (q_{A_-}, A\alpha \times l : \vec{i}, l).$$

The analysis of the remaining cases directly follows from the definitions, so we omit it. ◀

5 Linear-Time Simulation of 2DPPDA

Our linear-time simulation algorithm for 2DPPDA is almost the same as S. Cook's algorithm for 2DPDA [5]. One can find the detailed exposition in [2] and [8].

▶ **Theorem 7** ([13]). *Let M be a 2DPPDA. The language $L(M)$ is recognizable in time $O(n)$ in the RAM model. Moreover, there exists an $O(|w|)$ (in RAM) simulation algorithm for M on the input w.*

6 Structural Results

We use the computational model to obtain new structural results about the PELs.

▶ **Lemma 8.** *Let X be a DCFL and Y be a PEL. Then XY is a PEL.*

Proof. We describe a DPPDA M recognizing XY that simulates a DPDA M_X recognizing X and a DPPDA M_Y recognizing Y, constructed by a (well-formed) PEG.

DPPDA M simulates M_X until it reaches an accepting state. Then it pushes the information of the state to the stack, then pushes Z_0 (of M_Y) and simulates M_Y. If M_Y accepts the rest of the input, then the whole input is accepted. Otherwise, M pops symbols from the stack until it reaches the info about the M_X state and continues the simulation until it reaches an accepting state again. This process is continued until either M_Y accepts, or M_X reaches the end of the input (and M_Y rejects ε).

The correctness easily follows from the construction. During the process M tests all the prefixes of the input from X and checks whether the corresponding suffixes belong to Y. ◀

We use the notation PEL, DCFL, and REG for the language classes in formulas (the last one denotes regular languages). Denote by $\Gamma_{\mathsf{REG}}(\mathsf{DCFL})$ the regular closure of DCFLs; this class is defined as follows. $L \in \Gamma_{\mathsf{REG}}(\mathsf{DCFL})$ if there exists a regular expression (RE) R over an alphabet $\Sigma_k = \{a_1, \ldots, a_k\}$ and DCFLs $L_1, \ldots, L_k$ such that if we replace a_i by L_i in R the resulting expression $\psi(R)$ describes L.

▶ **Lemma 9** ([13])**.** $\Gamma_{\mathsf{REG}}(\mathsf{DCFL}) \subseteq \mathsf{PEL}$.

We describe only the proof idea. We provided the proof of Lemma 8 to generalize it as follows. In the case of a single concatenation, we have a kind of linear order for an exhaustive search. In the case of $\Gamma_{\mathsf{REG}}(\mathsf{DCFL})$ we will perform an exhaustive search in the order corresponding to a (graph of) deterministic finite automaton (DFA) recognizing R. If a word w on the input belongs to $L \in \Gamma_{\mathsf{REG}}(\mathsf{DCFL})$, then it can be split into subwords $w_1 \cdots w_k = w$ such that there exists a word $\alpha_1 \cdots \alpha_k \in R$ such that $w_i \in L_{\alpha_i}$, where L_{α_i} is the DCFL from the substitution that maps α_i to L_{α_i}. So, the exhaustive search finds the split of w by considering $\alpha_1 \cdots \alpha_k$ in the length-lexicographic order and considering w's subwords $w_i \in L_{\alpha_i}$ ordered by the length. If a word $w_1 \in L_{\alpha_1}$ is the shortest prefix, the DPPDA tries to find the shortest $w_2 \in L_{\alpha_2}$ and so on. If at some point the DPPDA failed to find $w_{j+1} \in L_{\alpha_{j+1}}$, it rollbacks to α_j and tries to find a longer word $w_j \in L_{\alpha_j}$. If it fails, then it rollbacks to α_{j-1} and so on. During the search of w_j, the DPPDA simulates a DPDA M_j recognizing L_{α_j}.

Denote by $\Gamma_{\mathsf{Bool}}(\mathscr{L})$ the *Boolean closure* of the language's class $\mathscr{L}$, i.e., $\Gamma_{\mathsf{Bool}}(\mathscr{L})$ is a minimal class satisfying the conditions:

- $\mathscr{L} \subseteq \Gamma_{\mathsf{Bool}}(\mathscr{L})$
- $\forall A, B \in \Gamma_{\mathsf{Bool}}(\mathscr{L}) : A \cup B, A \cap B, \overline{A} \in \Gamma_{\mathsf{Bool}}(\mathscr{L})$

▶ **Theorem 10.** *The following assertions hold.*

1. $\Gamma_{\mathsf{Bool}}(\Gamma_{\mathsf{REG}}(\mathsf{DCFL})) \subseteq \mathsf{PEL}$.
2. $\Gamma_{\mathsf{REG}}(\mathsf{DCFL}) \cdot \mathsf{PEL} = \mathsf{PEL}$.

Proof. It was shown in [7] that $\Gamma_{\mathsf{Bool}}(\mathsf{PEL}) = \mathsf{PEL}$. We proved that $\Gamma_{\mathsf{REG}}(\mathsf{DCFL}) \subseteq \mathsf{PEL}$, so $\Gamma_{\mathsf{Bool}}(\Gamma_{\mathsf{REG}}(\mathsf{DCFL})) \subseteq \mathsf{PEL}$.

The inclusion $\Gamma_{\mathsf{REG}}(\mathsf{DCFL}) \cdot \mathsf{PEL} \supseteq \mathsf{PEL}$ is obvious ($\{\varepsilon\} \in \Gamma_{\mathsf{REG}}(\mathsf{DCFL})$). The inclusion $\Gamma_{\mathsf{REG}}(\mathsf{DCFL}) \cdot \mathsf{PEL} \subseteq \mathsf{PEL}$ follows from the modification of the simulation algorithm from the proof of Lemma 9 by the simulation step from the proof of Lemma 8: when M_j reaches an accepting state and the corresponding state of $\mathcal{A}$ is an accepting state, M simulates the DPPDA for the PEG. If it successfully parses the suffix, the input is accepted, otherwise, the simulation continues as in the proof of Lemma 9. Recall that it is not known, whether PELs are closed over concatenation (Remark Remark:ConcatClosure), so we have to use Lemma 9 to for left concatenation with $\Gamma_{\mathsf{REG}}(\mathsf{DCFL})$. ◀

▶ **Corollary 11.** *For each $L \in \Gamma_{\mathsf{Bool}}(\Gamma_{\mathsf{REG}}(\mathsf{DCFL}))$ there exists a RAM-machine M that decides, whether $w \in L$ in time $O(|w|)$. In other words, the class $\Gamma_{\mathsf{Bool}}(\Gamma_{\mathsf{REG}}(\mathsf{DCFL}))$ is linear-time recognizable.*

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the Current Clique Algorithms Are Optimal, So is Valiant's Parser. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, FOCS '15, pages 98–117, USA, 2015. IEEE Computer Society. `doi:10.1109/FOCS.2015.16`.
2. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley Publishing Company, 1974.
3. Eberhard Bertsch and Mark-Jan Nederhof. Regular Closure of Deterministic Languages. *SIAM J. Comput.*, 29:81–102, 1999.
4. A. Birman and J. D. Ullman. Parsing algorithms with backtrack. In *11th Annual Symposium on Switching and Automata Theory (SWAT 1970)*, pages 153–174, 1970.

5 Stephen A Cook. *Linear time simulation of deterministic two-way pushdown automata.* Department of Computer Science, University of Toronto, 1970.

6 Bryan Ford. Packrat Parsing: Simple, Powerful, Lazy, Linear Time, Functional Pearl. In *Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming*, ICFP '02, pages 36–47, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/581478.581483.

7 Bryan Ford. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, pages 111–122, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/964001.964011.

8 Robert Glück. Simulation of Two-Way Pushdown Automata Revisited. In *Electronic Proceedings in Theoretical Computer Science*, volume 129, pages 250–258. Open Publishing Association, September 2013.

9 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.

10 Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. doi:10.1137/0206024.

11 Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, 2002.

12 Bruno Loff, Nelma Moreira, and Rogério Reis. The computational power of parsing expression grammars. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 491–502. Springer, 2018. doi:10.1007/978-3-319-98654-8_40.

13 Alexander Rubtsov and Nikita Chudinov. Computational Model for Parsing Expression Grammars, 2024. arXiv:2406.14911.

14 Alexander A. Rubtsov. A Linear-Time Simulation of Deterministic d-Limited Automata. In *Developments in Language Theory: 25th International Conference, DLT 2021, Porto, Portugal, August 16–20, 2021, Proceedings*, pages 342–354, Berlin, Heidelberg, 2021. Springer-Verlag. doi:10.1007/978-3-030-81508-0_28.

15 Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, Boston, MA, third edition, 2013.

16 Guido van Rossum, Pablo Galindo, and Lysandros Nikolaou. New PEG parser for CPython, 2020. URL: https://peps.python.org/pep-0617/.

Monoids of Upper Triangular Matrices over the Boolean Semiring

Andrew Ryzhikov
Department of Computer Science, University of Oxford, UK

Petra Wolf
LaBRI, CNRS, Université de Bordeaux, Bordeaux INP, France

Abstract

Given a finite set $\mathcal{A}$ of square matrices and a square matrix B, all of the same dimension, the membership problem asks if B belongs to the monoid $\mathcal{M}(\mathcal{A})$ generated by $\mathcal{A}$. The rank one problem asks if there is a matrix of rank one in $\mathcal{M}(\mathcal{A})$. We study the membership and the rank one problems in the case where all matrices are upper triangular matrices over the Boolean semiring. We characterize the computational complexity of these problems, and identify their PSPACE-complete and NP-complete special cases.

We then consider, for a set $\mathcal{A}$ of matrices from the same class, the problem of finding in $\mathcal{M}(\mathcal{A})$ a matrix of minimum rank with no zero rows. We show that the minimum rank of such matrix can be computed in linear time.We also characterize the space complexity of this problem depending on the size of $\mathcal{A}$, and apply all these results to the ergodicity problem asking if $\mathcal{M}(\mathcal{A})$ contains a matrix with a column consisting of all ones. Finally, we show that our results give better upper bounds for the case where each row of every matrix in $\mathcal{A}$ contains at most one non-zero entry than for the general case.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases matrix monoids, membership, rank, ergodicity, partially ordered automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.81

Funding *Andrew Ryzhikov*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).
Petra Wolf: Supported by the French ANR, project ANR-22-CE48-0001 (TEMPOGRAL).

Acknowledgements We thank anonymous reviewers for their comments that improved the content and presentation of the paper.

1 Introduction

Membership in matrix monoids. Given a finite set $\mathcal{A}$ of $n \times n$ matrices and an $n \times n$ matrix B, *the membership problem* asks if B belongs to the monoid $\mathcal{M}(\mathcal{A})$ generated by $\mathcal{A}$, which is the set of all products of matrices from $\mathcal{A}$. For matrices over rational numbers, membership is solvable in polynomial time if $\mathcal{A}$ consists of one matrix [22]. This result was later extended to the case where $\mathcal{A}$ is a set of commuting matrices [1], and then to a special class of non-commutative matrices [28]. Membership is also decidable for 2×2 non-singular integer matrices [34]. On the other hand, membership is already undecidable for 3×3 integer matrices with determinant equal to 0 [32].

For sets of matrices over the Boolean semiring (the set $\{0, 1\}$ with addition and multiplication defined the same way as for integer numbers, except that $1 + 1 = 1$), many natural problems, including membership, are trivially in PSPACE, and are thus more tractable. However, membership in this case is actually PSPACE-complete [25]. In this paper, we further restrict the setting and study upper triangular matrices over the Boolean semiring. Everywhere below in this paper, we assume all matrices to be square and over the Boolean

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 81; pp. 81:1–81:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

semiring, unless specified otherwise. Upper triangular matrices may seem to be a simple and restrictive class, but sets of such matrices and the corresponding automata have been studied extensively, and many problems for them are known to be PSPACE-complete, see e.g. [33, 26, 31, 37, 30]. Nevertheless, we are not aware of any results on the four problems we consider in this paper (membership, rank one, minimum rank of a total word and ergodicity) specifically for the case of upper triangular matrices, except for the results of [43] on the ergodicity problem, which we explain below.

Matrices and sets of matrices over the Boolean semiring play an important role in automata theory [4], discrete-time Markov chains [38], graph theory [2], variable-length codes [12], and symbolic dynamics [27]. They are often used in the theory of nonnegative matrices, since in many problems one is not interested in the actual values of the entries, but only in which values are zero and which are strictly positive [3].

Boolean square matrices can also be viewed as binary relations on a finite set Q. Matrices with at most one non-zero entry in each row thus correspond to partial transformations, and if they have exactly one non-zero entry in every row, to complete transformations. We call such matrices transformation and complete transformation matrices respectively. For monoids of complete transformation matrices, membership is PSPACE-complete [25], and its complexity was studied for several subsclasses [7, 6, 9, 8]. Connections of membership with the automata intersection problem are explored in [13].

The rank one problem. Instead of asking if the monoid $\mathcal{M}(\mathcal{A})$ contains a given matrix, one can ask if it contains a matrix from a certain class. Given a set $\mathcal{A}$ of matrices, *the rank one problem* asks if there is a matrix of rank one in $\mathcal{M}(\mathcal{A})$.

The rank one problem is known to be solvable in polynomial time for monoids of complete transformation matrices (see e.g. [41]), strongly connected monoids of transformation matrices [11] and, more generally, for strongly connected monoids of unambiguous relations and their subclasses [36, 14, 24, 4] (a monoid of $n \times n$ matrices is called *strongly connected* if for each pair (i, j), $1 \leq i, j \leq n$, it contains a matrix whose entry (i, j) is equal to one). The latter case plays an important role in the theory of variable-length codes because of its connections with synchronizing codes [12]. On the other hand, the rank one problem is PSPACE-complete already for monoids of transformation matrices which are not strongly connected [10]. The complexity of deciding if a monoid of complete transformation matrices contains a matrix of given rank is studied in [18].

The rank one problem is related to the Černý conjecture and its generalizations. Indeed, the case of monoids of complete transformations corresponds exactly to the case of complete DFAs, and hence in this case the rank one problem is nothing else than the problem of checking if a complete DFA is synchronizing. We refer to the surveys [41, 42, 23, 5] for the vast literature on synchronizing DFAs.

Careful synchronization and ergodicity. A similar problem is, given a set $\mathcal{A}$ of matrices, to find in $\mathcal{M}(\mathcal{A})$ a minimum rank matrix without zero rows. We call matrices with no zero rows *total*, since they correspond to total relations. For sets of transformation matrices, monoids of matrices containing a total matrix of rank one correspond to carefully synchronizing DFAs [29]. Deciding if $\mathcal{M}(\mathcal{A})$ contains a total matrix of rank one is PSPACE-complete even if $\mathcal{A}$ consists of two matrices [29], and the length of a shortest product of matrices from $\mathcal{A}$ resulting in a total matrix of rank one can be exponential [29]. If $\mathcal{A}$ is a set of upper triangular transformation matrices, deciding if $\mathcal{M}(\mathcal{A})$ contains a total matrix of rank one and finding a product resulting in such a matrix is solvable in polynomial time [43]. We discuss the contributions of [43] and their connection to the results of this paper in more details in the beginning of Section 4.

A closely related notion is that of ergodic matrices (also known as column-primitive [16] or D3-directing [20, 21]). A matrix is called *ergodic* [44] if it has a column consisting of all ones. Note that a transformation matrix is ergodic if and only if it is a total matrix of rank one, that is, this case also coincides with carefully synchronizing DFAs. *The ergodicity problem* asks if a given matrix monoid contains an ergodic matrix. The ergodicity problem is PSPACE-complete [29], but is solvable in polynomial time for monoids of matrices with no zero rows [35, 44]. An application of ergodicity to deciding if a discrete-time multi-agent system can be driven to consensus is presented in [16]. We emphasize that the existing polynomial-time solvability results on ergodicity [44, 16, 15] (except for [43]) consider monoids of matrices with no zero rows, while our results for this problem do not have this requirement.

2 Main definitions and our contributions

Boolean rank. *The Boolean semiring* is the set $\{0, 1\}$ with addition and multiplication defined in the usual way except that $1 + 1 = 1$. All matrices in this paper are over the Boolean semiring. The *Boolean rank* [17] (which we call just *rank* for brevity in this paper) of an $n \times n$ matrix A is the smallest number r such that $A = CR$, where C is an $n \times r$ matrix, and R is an $r \times n$ matrix, with usual matrix multiplication over the Boolean semiring.

For example, the following equality shows that the rank of A is at most two (moreover, it cannot be one since there are two different non-zero rows in A):

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = CR.$$

Matrices and automata. Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ be a set of $n \times n$ matrices over the Boolean semiring. The *monoid generated by* $\mathcal{A}$ is the set of all possible products (with respect to the usual matrix multiplication) of matrices from $\mathcal{A}$, including the empty product corresponding to the identity matrix. By assigning to each matrix $A_i \in \mathcal{A}$ a letter a_i from a finite alphabet Σ of size m, the set $\mathcal{A}$ can be equivalently seen as a nondeterministic finite automaton (NFA) with state set $Q = \{1, 2, \ldots, n\}$, alphabet Σ and transition relation $\Delta : Q \times \Sigma \to 2^Q$ defined for each letter a_i by the action of the corresponding matrix A_i. That is, we have $j \in \Delta(i, a_k)$ if and only if the entry (i, j) in A_k is one. If the transition relation is clear from the context, we denote the set $\Delta(q, a)$ as $q \cdot a$. Equivalently, given an NFA (Q, Σ, Δ), we can consider for each $a \in \Sigma$ the matrix such that its entry (p, q), $p, q \in Q$, is non-zero if and only if $q \in \Delta(p, a)$. This allows to consider an NFA as a set of matrices corresponding to its letters. The transition relation can be homomorphically extended to the set Σ^* of words over Σ. Words over Σ thus naturally correspond to products of matrices. In particular, by the matrix of a word we mean the result of this product, and by the rank of a word we mean the rank of its matrix. In this paper we make use of this correspondence between NFAs and matrix monoids and switch between the languages of matrices and automata whenever convenient. We remark that the described construction is nothing more than a homomorphism from the free monoid Σ^* to the monoid of matrices over the Boolean semiring.

In NFA terms, the matrix M of a word w has rank at most r if and only if there exist r subsets $Q_1, \ldots, Q_r$ of Q such that the image of every state of Q under w is a union of some of these subsets. If r is the smallest such number, the rank of M is equal to r. In particular, the rank of a word is one if and only if there is a subset $Q_1 \subseteq Q$ such that the image of every state under w is either Q_1 or the empty set, and not all images are empty.

For a set $S \subseteq Q$ of states and a word $w \in \Sigma^*$, we denote $S \cdot w = \{p \in Q \mid p \in q \cdot w$ for some $q \in S\}$. We say that the action of a word w is *defined* for a state q if $q \cdot w$ is non-empty. A word is called *total* if its action is defined for every state. Equivalently, it means that the matrix of a word does not have zero rows.

An NFA is a *deterministic finite automaton* (DFA) if for every $p \in Q$ and $a \in \Sigma$ we have that $|\Delta(p, a)| \leq 1$. In this case we use δ instead of Δ to emphasize that the transition relation is a (partial) transition function. If moreover $\delta(p, a)$ is defined for every $p \in Q$ and $a \in \Sigma$, the DFA is called *complete*. For DFAs, the rank of a word w is simply the size $|Q \cdot w|$.

Ergodic words. A word w is called *ergodic* for an NFA $\mathcal{A} = (Q, \Sigma, \Delta)$ if there is a state $q \in Q$ such that for every state $p \in Q$ we have $q \in p \cdot w$. Equivalently, a word is ergodic if its matrix contains a column of all ones. Clearly, every ergodic word is total. We call an NFA *ergodic* if it admits an ergodic word.

Partially ordered automata. An NFA $\mathcal{A} = (Q, \Sigma, \Delta)$ is called *partially ordered* (poNFA) if there exists an order $\prec$ on its set of states Q which is preserved by all its transitions. That is, if $p \in \Delta(q, a)$ for $p, q \in Q$ and $a \in \Sigma$, then $q \prec p$ or $q = p$. Such order can be found in time $\mathcal{O}(mn)$ by topological sorting, and in deterministic logarithmic space [40], hence we always assume that a poNFA is provided together with the order $\prec$. Equivalently, an NFA is a poNFA if the underlying digraph of $\mathcal{A}$ does not have any cycles other than self-loops, or, alternatively, if the matrices of all letters of $\mathcal{A}$ are upper triangular. A poNFA $\mathcal{A}$ is called *self-loop deterministic* if for every letter $a \in \Sigma$ such that $q \in q \cdot a$ for a state $q \in Q$ we have $q \cdot a = \{q\}$. Following [26], we denote self-loop deterministic partially ordered NFAs as rpoNFAs. We denote partially ordered DFAs as poDFAs.

Decision problems. An *NFA acceptor* $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ is an NFA (Q, Σ, Δ) with distinguished sets $I \subseteq Q$ and $F \subseteq Q$ of, respectively, initial and final states. The *language* $L(\mathcal{A})$ of an NFA acceptor is the set of words $w \in \Sigma^*$ such that there exist states $i \in I$, $f \in F$ with $f \in i \cdot w$. Given an NFA acceptor, *the universality problem* asks if it accepts all words over its alphabet. Universality of poNFAs is PSPACE-complete even over a binary alphabet [26], and remains PSPACE-complete for rpoNFAs over polynomial size alphabet [26, 30].

Given an NFA $\mathcal{A}$ and a matrix B, *the membership problem* asks if there exists a word such that its matrix in $\mathcal{A}$ is equal to B. Given an NFA $\mathcal{A}$, the *rank one problem* asks if there is a word whose matrix has rank one in $\mathcal{A}$.

We assume that the reader is familiar with basic notions of automata theory and computational complexity, see e.g. [39].

Our contributions. In the first half of this paper, Section 3, we analyze the computational complexity of the membership and rank one problems for poNFAs (equivalently, for monoids of upper triangular matrices). We show that they are PSPACE-complete for rpoNFAs and ternary poNFAs (Proposition 6). The main results of Section 3 are that for rpoNFAs over a fixed alphabet the membership problem is in NP (Theorem 1), and remains NP-complete even for complete poDFAs over a binary alphabet (Theorem 8). We show containment in NP by proving that for each word there is a short word with the same matrix (Proposition 2). We complement this result with a lower bound on the length of words with a given matrix (Lemma 7).

In the second half of the paper, Section 4, we study the problem of finding the minimum rank of a total word in a poNFA. We show that, depending on the size of the alphabet, this problem lies between L and P-complete, and moreover can be solved in linear time. These results are summarized in Theorem 11. All these results transfer to the problem of ergodicity of a set of upper triangular matrices (Theorem 15), which asks if a matrix monoid contains a matrix with a column consisting of all ones. We then further extend these results to get improved bounds for ergodicity of poDFAs (Theorem 16).

3 Membership and minimum rank

3.1 Upper bounds for rpoNFAs

The goal of this subsection is to prove the following theorem.

▶ **Theorem 1.** *Membership for rpoNFAs over a fixed alphabet is in* NP.

We start with the following key result which, intuitively, shows that if $\mathcal{A}$ is an rpoNFA, a product of matrices from $\mathcal{A}$ cannot contain too many partial products.

▶ **Proposition 2.** *Let $\mathcal{A} = (Q, \Sigma, \Delta)$ be an rpoNFA with n states and m letters. For every word $w \in \Sigma^*$ there exist at most $n(n+1)^{m-1}$ prefixes of w with pairwise different matrices.*

Proof. We prove the statement by induction on m. Clearly, if $m = 1$, the matrix of every word w of length more than n is equal to the matrix of some word v of length at most n, and thus the matrix of every prefix of w is equal to the matrix of some prefix of v, and there are only at most n prefixes.

Assume now that $m \geq 2$ and the statement is true for rpoNFAs over all alphabets $\Sigma' \subset \Sigma$, $|\Sigma'| \leq m - 1$. We can assume that every letter from Σ appears in w, otherwise the statement is proved. Let $Q = Q_1 \cup Q_1'$, where Q_1 is the set of states p such that there exists a letter $a \in \Sigma$ with $p \notin p \cdot a$, and $Q_1' = Q \setminus Q_1$. Let q be a smallest (with respect to $\prec$) state in Q_1, and let $a \in \Sigma$ be a letter such that $q \notin q \cdot a$.

Let w' be the prefix of w before the first appearance of a, that is, w' is the prefix of w such that w' does not contain a, and $w'a$ is also a prefix of w. Clearly, $w' \in (\Sigma \setminus \{a\})^*$. By the induction assumption, there are at most $n(n+1)^{m-2}$ prefixes of w' with pairwise different matrices. After the first application of a, for each state $p \in Q_1 \cdot w'a$ we have that $q \prec p$.

Now we perform the following iterative process. If the suffix of w after the first appearance of a does not contain all letters from Σ, we stop. Otherwise, we consider the smallest state q' in $Q_1 \cdot w'a$ and a letter a' such that $q' \notin q' \cdot a'$, and repeat the argument above with $Q_1 \cdot w'a$ taken as Q_1, q' taken as q, a' taken as a, and w' taken as w. This argument will be repeated at most n times in total, since after n times Q_1 must be empty, and thus the remaining suffix does not change the matrix of the word. Hence we get that there are at most $n \cdot (n(n+1)^{m-2} + 1) \leq n(n+1)^{m-1}$ prefixes of w with pairwise different matrices. ◀

▶ **Lemma 3.** *Let $\mathcal{A}$ be an rpoNFA and w be a word. Let P be the set of matrices of all prefixes of w, and $|P| = k$. Then there exist words $w_1, \ldots, w_k$ such that $w = w_1 w_2 \ldots w_k$, every matrix in P is equal to the matrix of $w_1 \ldots w_i$ for some i, and for each i, $0 \leq i < k$, for every non-empty prefix v of w_{i+1} the matrices of $w_1 w_2 \ldots w_i v$ and $w_1 w_2 \ldots w_i w_{i+1}$ are equal.*

Proof. Represent w as a concatenation $w_1 w_2 \ldots w_k$ of words w_i such that for every i, $0 \leq i < k$, the matrices of $w_1 w_2 \ldots w_i$ and $w_1 w_2 \ldots w_i w_{i+1}$ are different, but the matrix of $w_1 w_2 \ldots w_i v$ is equal to the matrix of $w_1 w_2 \ldots w_i w_{i+1}$ for every non-empty prefix v of w_{i+1}. We only need to show that for $i \neq j$ the matrices of $w_1 w_2 \ldots w_i$ and $w_1 w_2 \ldots w_j$ cannot be equal. This follows from the fact that in rpoNFAs, if for some non-empty words u_1, u_2 the matrices of u_1 and $u_1 u_2$ are different, there is no word u_3 such that the matrices of u_1 and $u_1 u_2 u_3$ are equal. ◀

The following statement implies that membership is in NP for rpoNFAs over a fixed alphabet, thus proving Theorem 1. It also implies that the rank one problem for rpoNFAs over a fixed alphabet is in NP.

▶ **Proposition 4.** *Let $\mathcal{A}$ be an rpoNFA with n states and m letters. Then for every word there exists a word of length at most $n(n+1)^{m-1}$ with the same matrix.*

Proof. Let $w_1, \ldots, w_k$ be the words from the statement of Lemma 3. By Proposition 2, $k \leq n(n+1)^{m-1}$. For each i, $1 \leq i \leq k$, let a_i be the first letter of w_i. Then the matrix of $a_1 a_2 \ldots a_k$ is equal to the matrix of w. ◀

As another consequence of our results, we get an elementary combinatorial proof of a theorem from [26] which originally required some non-trivial formal languages machinery.

▶ **Corollary 5** (Theorem 23 in [26]). *Let $\mathcal{A}$ be an NFA acceptor, and $\mathcal{B}$ be an rpoNFA acceptor, both over the same fixed alphabet. Checking if $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is in* coNP.

Proof. If $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$, there exists a word w accepted by $\mathcal{A}$ and not accepted by $\mathcal{B}$. Let $w_1, \ldots, w_k$ be the words from the statement of Lemma 3. By Proposition 2, $k \leq n(n+1)^{m-1}$. For each i let a_i be the first letter of w_i, and let $w_i = a_i w_i'$. Furthermore, let Σ_i be the set of letters occurring in w_i'. Consider the regular expression $e = a_1 \Sigma_1^* a_2 \Sigma_2^* a_3 \ldots \Sigma_{k-1}^* a_k \Sigma_k^*$. By construction, w is accepted by e, and for every word accepted by e its matrix in $\mathcal{B}$ is equal to the matrix of w in $\mathcal{B}$, and thus every such word is not accepted by $\mathcal{B}$. Moreover, e can be straightforwardly transformed into a poNFA acceptor $\mathcal{E}$ with at most $n(n+1)^{m-1}$ states and m letters. It remains to note that the intersection of $\mathcal{A}$ and $\mathcal{E}$ can be decided in polynomial time, and hence $\mathcal{E}$ can be used as a certificate for coNP. ◀

3.2 Lower bounds for rpoNFAs

We now complement the positive results from the previous subsection with lower bounds. We first show that if the alphabet is not fixed, membership for rpoNFAs is as hard as for general NFAs. The proof of that is done by constructing a reduction from the universality problem for rpoNFA acceptors, which is PSPACE-complete [26]. The same proof works for ternary poNFAs.

▶ **Proposition 6.** *The membership and rank one problems are* PSPACE*-complete for rpoNFAs and ternary poNFAs.*

Proof. To prove both statements, we use the following reduction from the universality problem for poNFA acceptors. Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ be a poNFA acceptor. Construct a poNFA $\mathcal{A}' = (Q \cup \{q_0, p_0, p_1, q_+, q_-\}, \Sigma \cup \{r\}, \Delta')$. The transition relation Δ' includes all transitions in Δ, with the addition of $\Delta'(q_0, r) = I$, $\Delta'(q, r) = \{q_+\}$ if $q \in F$, and $\Delta'(q, r) = \{q_-\}$ if $q \in Q \setminus F$. We also define $\Delta'(p_0, r) = \{p_1\}$, $\Delta'(p_1, r) = \{q_-\}$. Finally, all letters in Σ induce self-loops for q_+, q_-, q_0, p_0, p_1. See Figure 1 for an illustration.

We claim that $\mathcal{A}$ does not accept a word in Σ^* if and only if there exists a word in $\mathcal{A}'$ which maps q_0 and p_0 to $\{q_-\}$, and all other states to the empty set. Indeed, if there exists a word w not accepted by $\mathcal{A}$, then the word rwr has the required action in $\mathcal{A}'$. Conversely, if w' is a word having the required action, then it must contain exactly two occurrences of r, and its factor between the first and the second occurrence of r maps I to a subset of $Q \setminus F$ and is thus not accepted by $\mathcal{A}$. Observe also that every word of rank one must have this action by construction. Furthermore, the described construction preserves the property that the NFA is an rpoNFA. It remains to use the fact that the universality problem is PSPACE-complete for rpoNFA acceptors and binary poNFA acceptors [26]. ◀

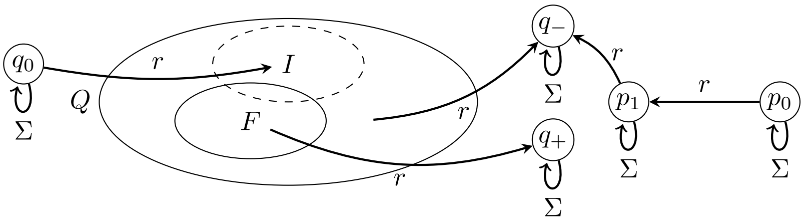


Figure 1 Illustration for the reduction in Proposition 6.

The following lemma provides a lower bound for the value discussed in Proposition 4. We remark that for constant m our lower and upper bounds are asymptotically the same, and if $m = n - 1$ the lower bound is exponential.

▶ **Lemma 7.** *Let n and m be positive natural numbers such that m divides $n - 2$. Then there exists an rpoNFA with n states and m letters such that for some states t and f the length of a shortest word w with $f \cdot w = \{f\}$ and $q \cdot w = \{t\}$ for every state $q \neq f$ is $\frac{n-2}{m}(\frac{n-2}{m} + 1)^{m-1}$.*

Proof. Let $n = ms + 2$. We construct an rpoNFA $\mathcal{A} = (Q, \Sigma, \Delta)$ with the required properties as follows. Let $Q^{(j)} = \{q_1^{(j)}, \ldots, q_s^{(j)}\}$ and $S = \cup_{1 \leq j \leq m} Q^{(j)}$. Take $Q = S \cup \{t, f\}$, where t and f are fresh states, and $\Sigma = \{a_1, a_2, \ldots, a_m\}$. The transition relation Δ is defined for the states in the sets $Q^{(1)}, \ldots, Q^{(m)}$ in such a way that while reading a word w with $S \cdot w = \{t\}$, the states in $Q^{(j)}$ that contain an image of a state from S imitate an m-ary counter counting to zero, with some additional traversal in the process. Formally, for $1 \leq j, k \leq m$, $1 \leq i \leq s$ we set

$$\Delta(q_i^{(k)}, a_j) = \begin{cases} \{f\} & \text{if } j < k; \\ \{q_{i+1}^{(k)}\} & \text{if } j = k \text{ and } i < s; \\ Q^{(k+1)} \cup \ldots \cup Q^{(m)} & \text{if } j = k < m \text{ and } i = s; \\ \{t\} & \text{if } j = k = m \text{ and } i = s; \\ \{q_i^{(k)}\} & \text{if } j > k. \end{cases}$$

See Figure 2 for an illustration.

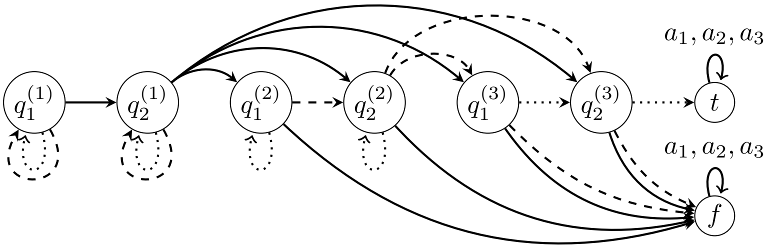


Figure 2 Illustration for the construction in Lemma 7. For unlabeled transitions, solid lines stand for a_1, dashed for a_2, dotted for a_3.

The only way to map a state to $\{t\}$ is to map it first to $\{q_s^{(m)}\}$ and then apply the letter a_m, otherwise the state is mapped to a set containing f. Observe that the shortest word such that $S \cdot w = \{t\}$ is unique. Indeed, at every moment of time there is a unique letter that does not act as the identity and does not map any state of S to f. To estimate the length of this word, we argue inductively. Mapping $Q^{(m)}$ to $\{t\}$ requires a word of length s. Mapping the set $Q^{(j)}$, $j < m$, to $\{t\}$ requires subsequently mapping each of its states to $\cup_{k>j} Q^{(k)}$, and then mapping this union to $\{t\}$ before we can proceed to $Q^{(j-1)}$. Hence we get the bound of $s(s+1)^{m-1}$. ◀

By applying the idea of the reduction in the proof of Proposition 6 to the automaton constructed in the proof of Lemma 7, one can get a slightly weaker lower bound of $\frac{n-4}{m-1}(\frac{n-4}{m-1} + 1)^{m-2} + 1$ for the length of a shortest word of rank one in rpoNFAs with n states and m letters, where $m-1$ divides $n-4$. Recall that a word of rank one is allowed to kill some states.

3.3 Partially ordered DFAs

The following theorem characterizes the complexity of membership and rank one for poDFAs.

▶ **Theorem 8.** *The membership problem is* NP*-complete for complete poDFAs, even over a binary alphabet. The rank one problem is* NP*-complete for poDFAs, even over a binary alphabet.*

The proof of NP-hardness is an adaptation of a construction from [37], and is omitted due to space constraints. Containment in NP follows from the following result.

▶ **Proposition 9.** *Let $\mathcal{A}$ be a poDFA with n states. Then for every word there exists a word of length at most $\frac{1}{2}(n+1)n$ with the same matrix.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta)$. Introduce a new state f, and define all undefined transitions of $\mathcal{A}$ to end there. Then for every state $q \in Q \setminus \{f\}$ we have $q \prec f$. Let w be a word such that there is no shorter word with the same matrix. For each prefix $w'a$ of w with $w' \in \Sigma^*, a \in \Sigma$, a must map one of the states in the image $Q \cdot w'$ to a larger state, otherwise this occurrence of a can be removed from w without changing its matrix. Hence the maximal length of w is at most $n + (n-1) + (n-2) + \ldots + 1 = \frac{1}{2}(n+1)n$. ◀

A lower bound on the value from Proposition 9 is provided in Lemma 17 below. Its adaptation provides the following lower bound on shortest words of rank one for poDFAs.

▶ **Lemma 10.** *For every natural numbers n and $m \geq 2$ such that m divides $n-1$, there exists a poDFA with $2n+1$ states and $m+1$ letters such that the length of a shortest word of rank one for it is $\frac{m-1}{2m}(n-1)^2 + (n-1)$.*

We remark that the rank one problem is trivial for poNFAs which do not have a word whose matrix is the zero matrix. Indeed, the minimum rank of a word in such poNFA is just the number of states such that each letter induces a self-loop for them. However, this obviously does not affect the complexity of membership, since adding a fresh state having self-loops by all letters guarantees that the matrix of each word is non-zero, while preserving the actions of all letters on the remaining states. For poDFAs, one can even obtain a complete poDFA by adding a sink state and sending all undefined transitions to it.

4 Total words of minimum rank and ergodic words

As shown above, the rank one problem is NP-complete for poDFAs and PSPACE-complete for rpoNFAs. In this section, we show that computing the minimum rank of a total word in a poNFA can be done in polynomial time. We then consider the case of poDFAs, show a tight quadratic bound on the length of shortest total words of minimum rank, and provide a fast algorithm to compute a total word of minimum rank within this length bound.

In [43], the second author proposed two algorithms for finding a total word of rank one in a poDFA if it exists. The first algorithm is greedy: take a letter mapping a currently active state to a larger one without taking undefined transitions. This algorithm does not seem to generalize to poNFAs, and is inherently sequential even for poDFAs. Moreover, its implementation proposed in [43] has time complexity $\mathcal{O}(mn^3)$. The second algorithm proposed in [43], of time complexity $\mathcal{O}(m^2n^2)$, uses a much less obvious approach. In this section we use a somewhat similar but more involved idea to deal with a more general case of poNFAs, and, in particular, show how to compute the minimum rank of a total word in linear time. For poDFAs, we show that a total word of minimum rank can be found in time $\mathcal{O}(mn^2)$, thus further improving on the results of [43]. We further refine the guarantee on the length of a found total word of minimum rank taking into account the size of the alphabet, and show that our upper bound it tight.

4.1 Total and ergodic words in poNFAs

▶ **Theorem 11.** *Let $\mathcal{A} = (Q, \Sigma, \Delta)$ be a poNFA with n states and m letters.*

(a) *The length of a shortest total word of minimum rank in $\mathcal{A}$ is at most $n(n+1)^m$.*

(b) *Finding the minimum rank of a total word in $\mathcal{A}$ can be done in $\mathcal{O}(m \log m + \log n)$ deterministic space.*
If $m = \mathcal{O}(\frac{\log n}{\log \log n})$, this problem is in L, and if $m = \Theta(n)$, it is P-complete.

(c) *The minimum rank of a total word in $\mathcal{A}$ can be computed in $\mathcal{O}(mn + |\Delta|)$ time, that is, in linear time in the size of the input.*

We first describe an algorithm that, given a poNFA $\mathcal{A} = (Q, \Sigma, \Delta)$ with n states and m letters, constructs an auxiliary sequence of sets $C_0 \supset C_1 \supset \ldots \supset C_k$ and a sequence of distinct letters $a_1, a_2, \ldots, a_k \in \Sigma$ for some $k \leq \min\{m, n-1\}$. Note that we use $\supset$ and $\subset$ to denote proper set inclusion. The value of k is determined by the algorithm and depends on the number of steps it makes. The intuition behind this algorithm (and the reason why it is called “stabilization algorithm”) are provided below in the proof of its correctness.

Stabilization algorithm

We initialize the set of currently considered states $C_0 = Q$ and the currently explored alphabet $\Sigma_0 = \emptyset$. At step i, $i \geq 1$, find a letter $a_i \in \Sigma \setminus \Sigma_{i-1}$ which is defined for every state in C_{i-1} and does not induce a self-loop for at least one of them. That is, for every state $q \in C_{i-1}$, $q \cdot a_i$ must be non-empty, and for at least one state $q \in C_{i-1}$ we must have $q \notin q \cdot a_i$. If no such letter a_i exists, stop and output $|C_{i-1}|$. Define C_i to be the subset of states in C_{i-1} for which a_i induces a self-loop, that is, $C_i = \{q \in C_{i-1} \mid q \in q \cdot a_i\}$. Take $\Sigma_i = \Sigma_{i-1} \cup \{a_i\}$. Go to the step $i + 1$.

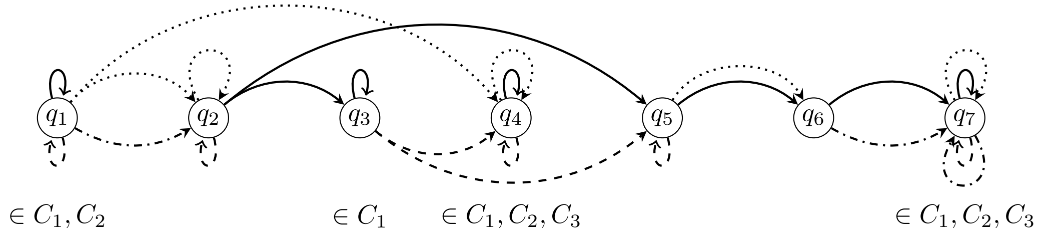


Figure 3 Running example for Section 4.1.

As a running example, we use the rpoNFA in Figure 3. The application of the algorithm gives $C_0 = Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$, $C_1 = \{q_1, q_3, q_4, q_7\}$, $C_2 = \{q_1, q_4, q_7\}$, $C_3 = \{q_4, q_7\}$. The letters chosen by the algorithm are as follows: a_1 is solid, a_2 is dashed, a_3 is dotted. The dashdotted letter is not used by the algorithm.

Correctness

We now analyze the output of the algorithm and show that it provides an upper bound on the minimum rank of a total word in $\mathcal{A}$. For rpoNFAs, this upper bound is tight. For general poNFAs, we then extend the algorithm with a second stage using similar ideas.

Assume that the algorithm makes $k + 1 \leq m + 1$ steps and then stops and returns the answer $|C_k|$. Consider the sequence of words defined as $w_0 = \epsilon$, $w_i = w_{i-1}(a_i w_{i-1})^n$ for $1 \leq i \leq k$. In our example, $w_1 = a_1^7$, $w_2 = a_1^7(a_2 a_1^7)^7$, $w_3 = a_1^7(a_2 a_1^7)^7(a_3 a_1^7(a_2 a_1^7)^7)^7$. Table 1 illustrates the images of states under w_i in our example.

The intuition is that with i increasing, the words w_i map every state to larger and larger states with respect to $\prec$, until a fixed point is reached. This can be informally seen as stabilization of the images of states from C_i under the word w_i.

Table 1 The images of states under w_i, $1 \leq i \leq 3$.

q	q_1	q_2	q_3	q_4	q_5	q_6	q_7
$q \cdot w_1$	q_1	q_3, q_7	q_3	q_4	q_7	q_7	q_7
$q \cdot w_2$	q_1	q_4, q_7	q_4, q_7	q_4	q_7	q_7	q_7
$q \cdot w_3$	q_4, q_7	q_4, q_7	q_4, q_7	q_4	q_7	q_7	q_7

▷ Claim 12. For each i, the word w_i is total.

Proof. We show by induction that the image of each state under w_i contains a state from C_i, starting with $i = 1$. By construction, a_1 is defined for every state $q \in C_0 = Q$. Since $\mathcal{A}$ is partially ordered, we get that after n applications of a_1, the image of every state must contain a state q such that $q \in q \cdot a_1$. Every such state is by definition in C_1.

Assume now that the image of every state under w_{i-1} contains a state from C_{i-1}. We thus only need to show that for each state in C_{i-1} its image under w_i contains a state from C_i. By definition, the image of every state from C_{i-1} under a_i is non-empty. Let $q \in C_{i-1}$ be a state. If $q \in C_i$, there is nothing to prove. Otherwise, $q \notin q \cdot a_i$. Thus we have that every state in $q \cdot a_i w_{i-1}$ is larger (with respect to $\prec$) than q. By the induction assumption, there is another state from C_{i-1} in $q \cdot a_i w_{i-1}$. Since $\mathcal{A}$ is partially ordered, by repeating this argument at most n times, we get a state from C_i in the image of q. ◁

▷ Claim 13. For every i, the rank of w_i is at most $|C_i|$.

Proof. For every state $q \in Q$, call *the i-trace of* q the set $q \cdot w_i$. To prove the claim, we show that for every $i \geq 1$ and every state $q \in Q$, the i-trace of q is a union of i-traces of some states from C_i. We prove that by induction on i. For $i = 1$, this is obvious from the construction of w_1 since $\mathcal{A}$ is partially ordered. Assume now that the statement holds true for $i - 1$. Then we need to show that for every $q \in Q$ and $p \in q \cdot w_{i-1}$ we have that $p \cdot (a_i w_{i-1})^n$ is a union of i-traces of some states from C_i. Observe that since $\mathcal{A}$ is partially ordered, for every $p' \in Q$ we have $p' \cdot (a_i w_{i-1})^n = p' \cdot (a_i w_{i-1})^{h(p')}$, where $h(p') = |\{s \in Q \mid p' \prec s \text{ or } p' = s\}|$. Hence, it is enough to prove that $p \cdot (a_i w_{i-1})^{h(p)}$ is a union of i-traces of some states from C_i.

We do that again by induction, now on the order $\prec$. For the largest state t with respect to $\prec$ this is obvious, since its i-trace for every i is $\{t\}$. Assume now that this is true for all states larger than p with respect to $\prec$. If $p \in C_i$, the statement is proved. Assume now that $p \notin C_i$. Then there exists $1 \leq j \leq i$ such that $p \notin p \cdot a_j$, and hence every state in $p \cdot a_i w_{i-1}$ is larger (with respect to $\prec$) than p. For every state p' such that $p \prec p'$, we have that $h(p') < h(p)$, hence we can use the assumption of the induction on $\prec$, which concludes the proof. $\lhd$

Hence we get that w_k is a total word of rank $|C_k|$. If $\mathcal{A}$ is an rpoNFA, the rank of every total word in $\mathcal{A}$ is at least $|C_k|$. Indeed, by the definition of the algorithm, each letter in Σ either induces a self-loop for every state of C_k, or is undefined for at least one state in C_k. Since $\mathcal{A}$ is an rpoNFA, this means that if the action of a word w is defined for every state of $\mathcal{A}$, then for every state $q \in C_k$ we must have $q \cdot w = \{q\}$. Since $\mathcal{A}$ is partially ordered, every set $q \cdot w$ can only contain q and states that are larger with respect to $\prec$ than q, hence the rank of w is at least $|C_k|$. Observe that the length of w_i is $(n+1)^m - 1$, which proves Theorem 11 (a) for rpoNFAs.

To deal with the general case of poNFAs instead of rpoNFAs, we continue our analysis of the stabilization algorithm, and extend it with a very similarly defined second part, which we call moving algorithm. The moving algorithm is executed after the stabilization algorithm. We start with some definitions. Recall that the result of the stabilization algorithm is the set C_k of states and the alphabet Σ_k.

Let $S \subseteq C_k$ be the set of states $q \in C_k$ such that for every letter $a \in \Sigma_k$ we have that $q \cdot a = \{q\}$. Let $\Sigma' \subseteq \Sigma \setminus \Sigma_k$ be the set of letters not from Σ_k which are defined for every state in S. Only letters from $\Sigma_k \cup \Sigma'$ can occur in a total word, since the first occurrence of any other letter will kill at least one state in S. After the stabilization algorithm provides its output, we run the moving algorithm defined as follows.

Moving algorithm

We initialize the set of currently considered states $C'_0 = C_k$ and the currently explored alphabet $\Sigma'_0 = \emptyset$. At step i, $i \geq 1$, find a letter $a'_i \in \Sigma' \setminus \Sigma'_{i-1}$ such that for at least one state $q \in C'_{i-1}$ we have $q \notin q \cdot a'_i$. If no such letter a'_i exists, stop and output $|C'_{i-1}|$. Define C'_i to be the subset of states in C'_{i-1} for which a'_i induces a self-loop, that is, $C'_i = \{q \in C'_{i-1} \mid q \in q \cdot a'_i\}$. Take $\Sigma'_i = \Sigma'_{i-1} \cup \{a'_i\}$. Go to the step $i+1$.

Correctness of the moving algorithm

Assume that the algorithm makes $k'+1$ steps, outputs $|C'_{k'}|$ and stops. Consider the words w'_i defined as follows. We take $w'_0 = (w_k)^n$. This is done to make sure that the image of every state in Q under w'_0 contains at least one state from S. For $1 \leq i \leq k'$, define $w'_i = w'_{i-1}(a'_i w'_{i-1})^n$.

First, we claim that each w'_i is total. This follows inductively from the fact that for every i, the image of every state under w'_i contains at least one state from S, and a'_i is defined for all states in S.

The fact that the rank of w'_i is at most $|C'_i|$ is proved by exactly the same argument as in the proof of Claim 13, since the construction of the words w'_i coincides with the construction of the words w_i. Observe that in the setting of Claim 13, a_i is defined for every state in C_i, but this fact is never used in the proof of this claim.

It remains to show that the rank of every total word in $\mathcal{A}$ is at least $|C'_{k'}|$. Indeed, by the definition of the moving algorithm, for each letter $a \in \Sigma$ which is defined for every state in S, we have that $q \in q \cdot a$ for every state $q \in C'_{k'}$. As argued above, only letters defined for each state in S can occur in a total word, which concludes the proof.

It is easy to see that the length of $w'_{k'}$ is at most $n(n+1)^m$. This concludes the proof of Theorem 11 (a) for general poNFAs.

To simplify the presentation, we assume that if the poNFA is not an rpoNFA, the moving algorithm is the second part of the stabilization algorithm. To comply with the notation of the stabilization algorithm, we define $C_{k+i} = C'_i$ and $\Sigma_{k+i} = \Sigma_k \cup \Sigma'_i$ for $1 \leq i \leq k'$. Observe that by construction, Σ_k and $\Sigma'_{k'}$ do not intersect, every letter in $\Sigma_{k+k'}$ induces a self-loop for every state in $C_{k+k'}$ and $w_{k+k'}$ is a total word of minimum rank. Thus, everywhere below we denote the value $k'+k$ simply by k.

Space complexity

We show how to implement a variant of the stabilization algorithm in $\mathcal{O}(m \log m + \log n)$ deterministic space. To do so, it is enough to note that the sets C_i do not have to be constructed explicitly, and each C_i can be reconstructed on the fly based only on Σ_i. Indeed, assume that the sequence $a_1, \ldots, a_i$ is already constructed and stored in the memory. Then to test if a given state q belongs to C_i, we need to check that all of $a_1, \ldots, a_i$ induce self-loops for it. Testing if a given letter a can be taken as a_i is then straightforward and only requires going through all states of $\mathcal{A}$ in an arbitrary order.

If $m = \mathcal{O}(\frac{\log n}{\log \log n})$, the bit length of the representation of one letter in Σ is $\mathcal{O}(\log \log n)$. Thus storing a sequence $a_1, \ldots, a_d$, $d \leq m$, of letters in Σ requires $\mathcal{O}(\frac{\log n}{\log \log n} \cdot \log \log n) = \mathcal{O}(\log n)$ bits, which provides a deterministic algorithm running in logarithmic space. The described algorithm can obviously also be implemented in polynomial time, and P-hardness is stated in the following lemma. This concludes the proof of Theorem 11 (b).

▶ **Lemma 14.** *Deciding if a poDFA over an alphabet of linear size in the number of its states admits a total word of rank one is* P*-hard under an* AC^0 *reduction.*

Proof. We reduce from the monotone circuit value problem. A *monotone Boolean circuit* is an acyclic digraph $C = (V, E)$ with a labeling function $L : V \to \{\wedge, \vee, \Box\}$. Intuitively, the word "monotone" refers to the fact that negation gates are not allowed. Every vertex labeled by $\{\wedge, \vee\}$ is called *an inner gate* and has indegree two. Each vertex labeled by $\Box$ is called *an input gate* and has outdegree one and indegree zero. Additionally, there is a designated vertex of outdegree zero, which is called *the output gate.* A vertex v with an outgoing edge to a vertex u is called the child of u.

Let k be the number of input gates in C. Then the value of C on the input $x_1, \ldots, x_k$, $x_i \in \{0,1\}$, is defined recursively by computing the value of a gate by applying the corresponding Boolean operation to its children. Given a monotone Boolean circuit C with k input gates and the values $x_1, \ldots, x_k$, $x_i \in \{0,1\}$, of these input gates, the monotone circuit value problem asks if the value of the output gate of C is 1. This problem is P-complete, even if the circuit in the input is topologically ordered [19].

Given $C = (V, E)$ with a labeling function L, and the input values $x_1, \ldots, x_k$, define a poDFA $\mathcal{A} = (Q, \Sigma, \delta)$ as follows. Define $Q = V \cup \{f\}$, where f is a fresh state.

We now define the letters in Σ and their action on the states to force the following behavior of $\mathcal{A}$. We start with the whole set of states active, and then make a state non-active if the corresponding gate can be evaluated to one at the current step. First, we can map every state corresponding to an input gate with value one to f. This is done by a separate letter for each gate, and such a letter leaves every other state in its place. Then we inductively proceed with $\wedge$- and $\vee$-gates, whose corresponding states can be sent to f if and only if all their children in C have value 1 (respectively, at least on child has value 1). This is done by simple gadgets guaranteeing that all the children of a gate (respectively, at least one child of a gate) are non-active by leaving certain transitions for them undefined, see Figure 4 for an illustration of these gadgets. If by some sequence of such letter applications the state corresponding to the output gate can be made non-active, and hence the value of C is 1, then another letter mapping all other states to f can be taken. This letter cannot be taken initially, since it is undefined for the state corresponding to the output gate.

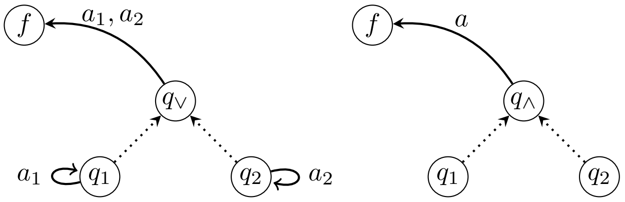


Figure 4 The gadgets for an $\vee$-gate $q_\vee$ with children q_1, q_2 (left) and an $\wedge$-gate $q_\wedge$ with children q_1, q_2 (right). The solid arrows denote transitions of the DFA, and the dotted arrows denote child-parent relations in the circuit.

We now formally define the remaining part of $\mathcal{A}$ to guarantee the described behavior.

- For each input gate v whose value is 1, add a fresh letter mapping v to f and acting as the identity for all states in Q except v.
- For each $\vee$-gate v, add two fresh letters a_1^v, a_2^v acting as follows. Both a_1^v and a_2^v maps v to f. Letter a_1^v induces a self-loop for the first child of v and is undefined for its second child. Symmetrically, a_2^v is undefined for the first child and maps the second child to itself. For all remaining states both a_1^v and a_2^v induce self-loops.
- Similarly, for each $\wedge$-gate v, add a fresh letter a^v to Σ such that a^v maps v to f, is undefined for both children of v and induces self-loops for all other states.
- Finally, add a fresh letter r which is undefined for the state corresponding to the output gate and maps all other states to f.

Applying a total word to $\mathcal{A}$ now resembles sequential evaluation of C: starting from the states corresponding to the input gates, we can make the states corresponding to gates with value one non-active by mapping them to f, and then proceed inductively to their parents. By definition of the action of r, $\mathcal{A}$ has a total word of rank one if and only if the state corresponding to the output gate eventually becomes non-active, which is possible if and only if the value of the output gate in C is one. ◀

Time complexity

To implement the stabilization algorithm with low time complexity, we use a data structure which we simply refer to as a list of states. We implement it as an array of length n that indicates for each state if it appears in the list, and also refers to the previous and next states in the array with respect to $\prec$, if they exist. Hence, testing non-emptiness of such a list can be done in constant time, going through its elements takes time linear in the size of the list (and not in n), and removing an element takes constant time.

We first show how to compute the minimum rank of a total word in time complexity $\mathcal{O}(mn)$. For each letter $a \in \Sigma$ we maintain a list D_a. After step i, this list contains the states in C_i for which a is undefined. We also maintain a list D of letters $a \in \Sigma \setminus \Sigma_i$ for which D_a is currently empty and there is a state $q \in C_i$ which is mapped by a to a different state.

The algorithm makes at most m steps. At step i, we take an arbitrary letter from D as a_i. We then update the set of currently considered states to C_i, which takes $\mathcal{O}(n)$ time, and remove the entry corresponding to each state in $C_{i-1} \setminus C_i$ from all the lists D_a, $a \in \Sigma \setminus \Sigma_i$. Observe that the entry for each state is removed at most once during the execution of the algorithm. Hence, the overall time complexity is $\mathcal{O}(mn + |\Delta|)$, which proves Theorem 11 (c).

Ergodicity

Recall that a word is ergodic if and only if its matrix has a column consisting of all ones. Clearly, in the case of poNFAs, this can only be the last column. We now apply the obtained results to ergodic words for a poNFA $\mathcal{A}$. Let Σ' be the set of letters in Σ which are defined for every state in C_k obtained in the stabilization algorithm. Let $C_k = \{q_1, q_2, \ldots, q_r\}$ such that $q_1 \prec q_2 \prec \ldots \prec q_r$.

Observe that there exists an ergodic word for $\mathcal{A}$ if and only if for every state q_i, $1 \le i \le r-1$, there exists a letter $a'_i \in \Sigma'$ such that there is a state $p \in Q$, $q \prec p$, with $p \in q \cdot a'_i$. If such a letter does not exist for some state in $C_k \setminus \{q_r\}$, an ergodic word for $\mathcal{A}$ obviously does not exist. For the opposite direction, observe that for every state in Q its image under $a'_1 w_k$ contains a state from $C_k \setminus \{q_1\}$. Moreover, $q_r \cdot a'_1 w_k = \{q_r\}$, since q_r must be the largest state in Q with respect to $\prec$. By inductively repeating this argument $r-1$ times in total, we get that for every state in Q its image under the concatenation of the words $(a'_i w_k)$ for $1 \le i \le r-1$ must contain q_k, and hence this word is ergodic. Since $r \le n$, its length is at most $(n-1)n(n+1)^m$. We thus obtain the following theorem.

▶ **Theorem 15.** *Let $\mathcal{A}$ be a poNFA with n states and m letters.*

(a) *The length of its shortest ergodic word is at most $(n-1)n(n+1)^m$.*

(b) *Checking if $\mathcal{A}$ is ergodic can be done in $\mathcal{O}(m \log m + \log n)$ deterministic space. If $m = \mathcal{O}(\frac{\log n}{\log \log n})$, this problem is in L, and if $m = \Theta(n)$, it is P-complete.*

(c) *Checking if $\mathcal{A}$ is ergodic can be done in time $\mathcal{O}(mn + |\Delta|)$.*

4.2 Partially ordered DFAs

For poDFAs, a total word has rank one if and only if it is ergodic. Thus the following theorem, which is the main result of this subsection, applies to ergodic words if one takes $r = 1$.

▶ **Theorem 16.** *Let $\mathcal{A}$ be a poDFA with n states and m letters.*

(a) *The length of a shortest total word of minimum rank r in $\mathcal{A}$ is at most $\frac{k-1}{2k}(n-r)^2 + (n-r)$, where $k = \min\{n-r, m\}$. If m divides $n-r$, the bound is tight.*

(b) *Finding such a word of at most this length can be done in time $\mathcal{O}(mn^2)$.*

We remark on the values that the bound $\frac{k-1}{2k}(n-r)^2 + (n-r)$ takes depending on the size of the alphabet, assuming that $r = 1$. If $m = 2$, the lower bound is $\frac{1}{4}(n-1)(n+3) = \frac{1}{4}n^2 + \mathcal{O}(n)$, while for $m = n-1$ it becomes $\frac{1}{2}n(n-1)$. As we will see later, having more than $n-1$ letters in the alphabet cannot increase the bound.

Proof. We start with proving (a). We continue the analysis of the stabilization algorithm for rpoNFAs from Section 4.1. In the proof of its correctness, we constructed a very long word w_k. However, intuitively, during the application of this word to a poDFA, most letters do not change the image of any state at the moment of their application, and each such letter can thus be removed from the word. For $1 \le i \le k$, denote $|C_i| = c_i$

Put a token onto each state of $\mathcal{A}$ and move them according to the transitions taken during the application of w_k letter by letter. We are going to track the paths of these tokens. First, we claim that for each state, if a token appears on it and then leaves it, it leaves it every time along the transition labeled by the same letter. Indeed, by construction of w_k, each state $q \in Q \setminus C_k$ belongs to a set $C_{i-1} \setminus C_i$ for exactly one value of i, and thus we can show by induction that a_i is the letter labeling such a transition from q. Given a state $q \in Q$, we call such a transition *the moving transition of* q. The number of moving transitions labeled by a_i is $|C_{i-1} \setminus C_i| = c_{i-1} - c_i$. The total number of moving transitions labeled by letters in Σ_i is thus $(c_0 - c_1) + (c_1 - c_2) + \ldots + (c_{i-1} - c_i) = c_0 - c_i$.

Now, consider what happens when w_{i-1} is already applied to $\mathcal{A}$, and we start applying $(a_i w_{i-1})^n$. Observe that at this moment the set of states containing a token is precisely the set C_{i-1}, which can be proved by induction. By the application of $(a_i w_{i-1})^n$, each token on a state in $C_{i-1} \setminus C_i$ is moved to a state in C_i. The number of letters in $(a_i w_{i-1})^n$ whose application moves at least one token is thus upper bounded by the number of moved tokens, which is $|C_{i-1} \setminus C_i|$, multiplied by the number of moving transitions labeled by letters in Σ_{i-1}, plus at most one application of a_i per state in $C_{i-1} \setminus C_i$. Hence we get that the number of useful letters in $(a_i w_{i-1})^n$ is at most $(c_{i-1} - c_i)(c_0 - c_{i-1}) + (c_{i-1} - c_i) = (c_{i-1} - c_i)(c_0 - c_{i-1} + 1)$.

By construction, $c_0 = n$ and $c_k = r$. The overall length of the word consisting only of letters in w_k moving at least one token is upper bounded by

$$\sum_{1 \le i \le k} (c_{i-1} - c_i)(c_0 - c_{i-1} + 1) = (c_0 + 1)(c_0 - c_k) - (\sum_{1 \le i \le k} c_{i-1}^2 - \sum_{1 \le i \le k} c_{i-1} c_i)$$
$$= (c_0 + 1)(c_0 - c_k) - (\frac{1}{2}(c_0^2 - c_k^2) + \frac{1}{2} \sum_{1 \le i \le k} (c_{i-1} - c_i)^2)$$
$$\le (c_0 + 1)(c_0 - c_k) - \frac{1}{2}(c_0^2 - c_k^2) - \frac{1}{2k}(c_0 - c_k)^2 = \frac{k-1}{2k}(n - r)^2 + (n - r).$$

Hence the upper bound is proved. The fact that if m divides $n - r$ the bound is tight follows from the following lemma, concluding the proof of (a).

▶ **Lemma 17.** *For every positive natural numbers n, m and r such that m divides $n - 1$, there exists a poDFA $\mathcal{A}$ with n states and m letters such that the length of its shortest total word of minimum rank r is $\frac{m-1}{2m}(n - r)^2 + (n - r)$.*

Proof. We start with $r = 1$. We construct a poDFA $\mathcal{A} = (Q, \Sigma, \delta)$ as follows. Let $n = ms + 1$ and $Q^{(j)} = \{q_1^{(j)}, \ldots, q_s^{(j)}\}$. Take $Q = \cup_{1 \le j \le m} Q^{(j)} \cup \{q_1^{(m+1)}\}$, and $\Sigma = \{a_1, a_2, \ldots, a_m\}$. For each $1 \le j \le m$, define $\delta(q_i^{(j)}, a_j) = q_{i+1}^{(j)}$ for $1 \le i < s$, $\delta(q_s^{(j)}, a_j) = q_1^{(j+1)}$ and $\delta(q_i^{(j)}, a_\ell) = q_i^{(j)}$ if $j < \ell$ and $1 \le i \le s$. Set each letter in Σ to induce a self-loop for $q_1^{(m+1)}$. Leave all remaining transitions undefined. See Figure 5 for an example.

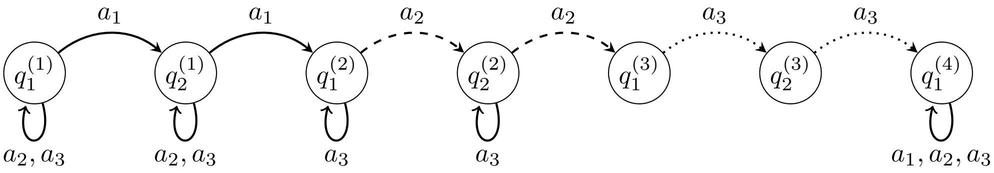


Figure 5 Example of the construction in the proof of Lemma 17 for $m = 3$, $s = 2$ and $r = 1$.

Observe that there exists a unique shortest total word of rank one for $\mathcal{A}$. Indeed, at every moment, only one transition that does not act as the identity can be taken. Namely, if no state in $Q^{(j)}$ is active and some state in $Q^{(j-1)}$ is active, we can take a_j, and as soon as one state in $Q^{(j)}$ becomes active, it has to be sent all the way to $a_1^{(m+1)}$ before any other state can be sent to another state. Hence, the length of a shortest total word of rank one for $\mathcal{A}$ is $s+(s^2+s)+(2s^2+s)+\ldots+((m-1)s^2+s) = s^2(1+2+\ldots+(m-1))+sm = s^2 \cdot \frac{(m-1)m}{2}+ms$. Since $s = \frac{n-1}{m}$, this value is equal to $\frac{m-1}{2m}(n-1)^2 + (n-1)$.

Finally, observe that adding $r-1$ states such that every letter induces a self-loop for each of them provides a required construction for the case of rank r. ◀

We continue with proving item (b) of Theorem 16. To construct a total word of minimum rank within the required length bound, recall the definition of the moving transitions from the length analysis above. For each state $q \in Q$, find the letter labeling its moving transition. Now we follow the tokens. At step i, all tokens on states in C_i are not moved, so we only need to compute a word that maps all tokens on the states in $C_{i-1} \setminus C_i$ to states in C_i. To do so, we need to decide which letters from $(a_i w_i)^n$ move at least one token. Clearly, if a_i does not move anything, we go to the next step. Otherwise, we apply a_i and recursively perform the argument from step $i-1$ for the moved tokens. The total number of moments of time when we do not move anything and thus go to the next step is upper bounded by $\mathcal{O}(mn^2)$ by the construction of w_k, and the number of moves of tokens is upper bounded by $\frac{m-1}{2m}(n-r)^2 + (n-r) = \mathcal{O}(mn^2)$. This concludes the proof of (b). ◀

References

1 László Babai, Robert Beals, Jin-yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, pages 498–507, USA, 1996. Society for Industrial and Applied Mathematics.

2 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

3 R. B. Bapat and T. E. S. Raghavan. *Nonnegative Matrices and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1997.

4 Marie-Pierre Béal, Eugen Czeizler, Jarkko Kari, and Dominique Perrin. Unambiguous automata. *Mathematics in Computer Science*, 1:625–638, 2008. doi:10.1007/s11786-007-0027-1.

5 Marie-Pierre Béal and Dominique Perrin. Synchronised automata. In Valérie Berthé and Michel Rigo, editors, *Combinatorics, Words and Symbolic Dynamics*, Encyclopedia of Mathematics and its Applications, pages 213–240. Cambridge University Press, 2016. doi:10.1017/CBO9781139924733.008.

6 M. Beaudry, P. McKenzie, and D. Thérien. The membership problem in aperiodic transformation monoids. *Journal of the ACM*, 39(3):599–616, 1992.

7 Martin Beaudry. Membership testing in commutative transformation semigroups. *Information and Computation*, 79(1):84–93, 1988. doi:10.1016/0890-5401(88)90018-1.

8 Martin Beaudry. Membership testing in transformation monoids. *PhD thesis, McGill University, Montreal, Quebec*, 1988.

9 Martin Beaudry. Membership testing in threshold one transformation monoids. *Information and Computation.*, 113(1):1–25, 1994. doi:10.1006/INCO.1994.1062.

10 Mikhail V. Berlinkov. On two algorithmic problems about synchronizing automata - (short paper). In Arseny M. Shur and Mikhail V. Volkov, editors, *Developments in Language Theory - 18th International Conference, DLT 2014, Ekaterinburg, Russia, August 26-29, 2014. Proceedings*, volume 8633 of *Lecture Notes in Computer Science*, pages 61–67. Springer, 2014. doi:10.1007/978-3-319-09698-8_6.

11 Mikhail V. Berlinkov, Robert Ferens, Andrew Ryzhikov, and Marek Szykuła. Synchronizing Strongly Connected Partial DFAs. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2021.12.

12 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and automata*, volume 129 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2010.

13 Michael Blondin, Andreas Krebs, and Pierre McKenzie. The complexity of intersecting finite automata having few final states. *computational complexity*, 25(4):775–814, 2016. doi:10.1007/s00037-014-0089-9.

14 Arturo Carpi. On synchronizing unambiguous automata. *Theoretical Computer Science*, 60:285–296, 1988. doi:10.1016/0304-3975(88)90114-4.

15 Pierre-Yves Chevalier, Vladimir V. Gusev, Raphaël M. Jungers, and Julien M. Hendrickx. Sets of stochastic matrices with converging products: Bounds and complexity. *CoRR*, abs/1712.02614, 2017. arXiv:1712.02614.

16 Pierre-Yves Chevalier, Julien M. Hendrickx, and Raphaël M. Jungers. Reachability of consensus and synchronizing automata. In *54th IEEE Conference on Decision and Control, CDC 2015, Osaka, Japan, December 15-18, 2015*, pages 4139–4144. IEEE, 2015. doi:10.1109/CDC.2015.7402864.

17 V. Froidure. Ranks of binary relations. *Semigroup Forum*, 54:381–401, 1997. doi:10.1007/BF02676619.

18 Pavel Goralčík and Václav Koubek. Rank problems for composite transformations. *International Journal of Algebra and Computation*, 05(03):309–316, 1995. doi:10.1142/S0218196795000185.

19 Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

20 Balázs Imreh and Magnus Steinby. Directable nondeterministic automata. *Acta Cybernetica*, 14(1):105–115, 1999.

21 Masami Ito and Kayoko Shikishima-Tsuji. Shortest directing words of nondeterministic directable automata. *Discrete Mathematics*, 308(21):4900–4905, 2008. doi:10.1016/J.DISC.2007.09.010.

22 R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM*, 33(4):808–821, 1986. doi:10.1145/6490.6496.

23 Jarkko Kari and Mikhail V. Volkov. Černý's conjecture and the road colouring problem. In *Handbook of Automata Theory*, pages 525–565. EMS Press, 2021. doi:10.4171/AUTOMATA-1/15.

24 Stefan Kiefer and Corto N. Mascle. On nonnegative integer matrices and short killing words. *SIAM Journal on Discrete Mathematics*, 35(2):1252–1267, 2021. doi:10.1137/19M1250893.

25 Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.

26 Markus Krötzsch, Tomáš Masopust, and Michaël Thomazo. Complexity of universality and related problems for partially ordered NFAs. *Information and Computation*, 255:177–192, 2017. doi:10.1016/J.IC.2017.06.004.

27 Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding*. Cambridge University Press, 2021.

28 Alexei Lisitsa and Igor Potapov. Membership and reachability problems for row-monomial transformations. In Jiří Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004*, pages 623–634, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

29 Pavel Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory Comput. Syst.*, 54(2):293–304, 2014. doi:10.1007/S00224-013-9516-6.

30 Tomáš Masopust and Markus Krötzsch. Partially Ordered Automata and Piecewise Testability. *Logical Methods in Computer Science*, 17(2):14:1–14:36, 2021. doi:10.23638/LMCS-17(2:14)2021.

31 Tomáš Masopust and Michaël Thomazo. On boolean combinations forming piecewise testable languages. *Theor. Comput. Sci.*, 682:165–179, 2017. doi:10.1016/J.TCS.2017.01.017.

32 Mike Paterson. Unsolvability in 3 × 3 matrices. *Studies in Applied Mathematics*, 49:105–107, 1970.

33 Jean-Eric Pin and Howard Straubing. Monoids of upper triangular boolean matrices. In *Semigroups. Structure and Universal AIgebraic Problems*, volume 39, pages 259–272, 1981.

34 Igor Potapov and Pavel Semukhin. Decidability of the membership problem for 2 × 2 integer matrices. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 170–186. SIAM, 2017. `doi:10.1137/1.9781611974782.12`.

35 Vladimir Yu. Protasov. Analytic methods for reachability problems. *J. Comput. Syst. Sci.*, 120:1–13, 2021. `doi:10.1016/J.JCSS.2021.02.007`.

36 Andrew Ryzhikov. Mortality and synchronization of unambiguous finite automata. In Robert Mercas and Daniel Reidenbach, editors, *Combinatorics on Words - 12th International Conference, WORDS 2019, Loughborough, UK, September 9-13, 2019, Proceedings*, volume 11682 of *Lecture Notes in Computer Science*, pages 299–311. Springer, 2019. `doi:10.1007/978-3-030-28796-2_24`.

37 Andrew Ryzhikov. Synchronization problems in automata without non-trivial cycles. *Theoretical Computer Science*, 787:77–88, 2019.

38 Eugene Seneta. *Non-negative matrices and Markov chains.* Springer Science & Business Media, 2006.

39 Michael Sipser. *Introduction to the Theory of Computation.* Course Technology, Boston, MA, third edition, 2013.

40 Seinosuka Toda. On the complexity of topological sorting. *Information Processing Letters*, 35(5):229–233, 1990. `doi:10.1016/0020-0190(90)90050-8`.

41 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008.

42 Mikhail V. Volkov. Synchronization of finite automata. *Russian Mathematical Surveys*, 77(5):819–891, 2022. `doi:10.4213/rm10005e`.

43 Petra Wolf. Synchronization under dynamic constraints. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, 2020.

44 Yaokun Wu and Yinfeng Zhu. Primitivity and Hurwitz primitivity of nonnegative matrix tuples: A unified approach. *SIAM Journal on Matrix Analysis and Applications*, 44(1):196–211, 2023. `doi:10.1137/22M1471535`.

An Algorithmic Meta Theorem for Homomorphism Indistinguishability

Tim Seppelt
RWTH Aachen University, Germany

Abstract

Two graphs G and H are *homomorphism indistinguishable* over a family of graphs $\mathcal{F}$ if for all graphs $F \in \mathcal{F}$ the number of homomorphisms from F to G is equal to the number of homomorphism from F to H. Many natural equivalence relations comparing graphs such as (quantum) isomorphism, cospectrality, and logical equivalences can be characterised as homomorphism indistinguishability relations over various graph classes.

The wealth of such results motivates a more fundamental study of homomorphism indistinguishability. From a computational perspective, the central object of interest is the decision problem $\textsc{HomInd}(\mathcal{F})$ which asks to determine whether two input graphs G and H are homomorphism indistinguishable over a fixed graph class $\mathcal{F}$. The problem $\textsc{HomInd}(\mathcal{F})$ is known to be decidable only for few graph classes $\mathcal{F}$. Due to a conjecture by Roberson (2022) and results by Seppelt (MFCS 2023), homomorphism indistinguishability relations over minor-closed graph classes are of special interest. We show that $\textsc{HomInd}(\mathcal{F})$ admits a randomised polynomial-time algorithm for every minor-closed graph class $\mathcal{F}$ of bounded treewidth.

This result extends to a version of HomInd where the graph class $\mathcal{F}$ is specified by a sentence in counting monadic second-order logic and a bound k on the treewidth, which are given as input. For fixed k, this problem is randomised fixed-parameter tractable. If k is part of the input, then it is coNP- and coW[1]-hard. Addressing a problem posed by Berkholz (2012), we show coNP-hardness by establishing that deciding indistinguishability under the k-dimensional Weisfeiler–Leman algorithm is coNP-hard when k is part of the input.

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

Keywords and phrases homomorphism indistinguishability, graph homomorphism, graph minor, recognisability, randomised algorithm, Courcelle's Theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.82

Related Version *Full Version*: `https://arxiv.org/abs/2402.08989`

Funding German Research Foundation (DFG) within Research Training Group 2236/2 (UnRAVeL), European Union (ERC, SymSim, 101054974). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Acknowledgements I would like to thank Martin Grohe and Louis Härtel for fruitful discussions and their support. In particular, I thank Martin Grohe for making me aware of Chinese remaindering.

1 Introduction

In 1967, Lovász [32] proved that two graphs G and H are isomorphic if and only if they are *homomorphism indistinguishable* over all graphs, i.e. they admit the same number of homomorphisms from every graph F. Subsequently, many graph isomorphism relaxations have been characterised as homomorphism indistinguishability relations. For example, two graphs are quantum isomorphic if and only if they are homomorphism indistinguishable over all planar graphs [34]. Moreover, two graphs satisfy the same sentences in k-variable first-order logic with counting quantifiers if and only if they are homomorphism indistinguishable over all graphs of treewidth less than k [20, 19]. A substantial list of similar results characterises notions from quantum information [34, 5], finite model theory [20, 24, 22], convex optimisation [19, 28, 38], algebraic graph theory [19, 28], machine learning [35, 41, 25], and category theory [18, 1] as homomorphism indistinguishability relations.

The wealth of such examples motivates a more principled study of homomorphism indistinguishability [3, 37, 40]. Notably, all graph classes featured in the results listed above are minor-closed and this is not a mere coincidence [40, Theorem 1]. Therefore, homomorphism indistinguishability relations of minor-closed graph classes are of central interest in light of the emerging theory of homomorphism indistinguishability. In [37], Roberson conjectured that *any two distinct graph classes which are closed under disjoint unions and taking minors have distinct homomorphism indistinguishability relations.* From a computational perspective, the central question on homomorphism indistinguishability concerns the complexity and computability of the following decision problem for a fixed graph class $\mathcal{F}$ [37, Question 9]:

HomInd($\mathcal{F}$)
Input Graphs G and H.
Question Are G and H homomorphism indistinguishable over $\mathcal{F}$?

The graphs G and H may be arbitrary graphs and do not necessarily have to be in $\mathcal{F}$. Typically, the graph class $\mathcal{F}$ is infinite. Thus, the trivial approach to HomInd($\mathcal{F}$) of checking whether G and H have the same number of homomorphisms from every $F \in \mathcal{F}$ does not even render HomInd($\mathcal{F}$) decidable. Beyond this observation, the understanding of the problems HomInd($\mathcal{F}$) is limited to a short list of examples of graph classes $\mathcal{F}$: For the class $\mathcal{G}$ of all graphs, HomInd($\mathcal{G}$) is graph isomorphism [32], a problem representing a long standing complexity-theoretic challenge and currently only known to be in quasi-polynomial time [6]. For the class $\mathcal{P}$ of all planar graphs, HomInd($\mathcal{P}$) is quantum isomorphism and undecidable [34]. Finally, for the class $\mathcal{TW}_k$ of all graphs of treewidth at most k, HomInd($\mathcal{TW}_k$) can be decided in polynomial time with the well-known k-dimensional Weisfeiler–Leman algorithm [20, 19].

These results illustrate that the complexity of HomInd($\mathcal{F}$) is not monotone in the sense that if $\mathcal{F}_1 \subseteq \mathcal{F}_2$, then HomInd($\mathcal{F}_1$) is at most as hard as HomInd($\mathcal{F}_2$). For example, despite that $\mathcal{TW}_2 \subseteq \mathcal{P} \subseteq \mathcal{G}$, deciding homomorphism indistinguishability over $\mathcal{TW}_2$, $\mathcal{P}$, and $\mathcal{G}$ is polynomial-time, undecidable, and quasi-polynomial-time, respectively. Furthermore, although HomInd($\mathcal{TW}_k$) is in polynomial-time for every k, there are infinitely many minor-closed graph classes $\mathcal{F}$ of bounded treewidth, e.g. the classes of k-outerplanar graphs, for which HomInd($\mathcal{F}$) could yet be undecidable. Our main result shows that this is not the case: HomInd($\mathcal{F}$) is in randomised polynomial time for every minor-closed graph class $\mathcal{F}$ of bounded treewidth.

▶ **Theorem 1.** *Let $k \geq 1$. If $\mathcal{F}$ is a k-recognisable class of graphs of treewidth at most $k-1$, then* HomInd($\mathcal{F}$) *is in* coRP*.*

Spelled out, Theorem 1 asserts that there exists a randomised algorithm for HomInd($\mathcal{F}$) which always runs in polynomial time, accepts all YES-instances and incorrectly accepts NO-instances with probability less than one half. Recognisability is a fairly general property that arises in the context of Courcelle's theorem [14], cf. Definition 10. Courcelle showed that every graph class definable in counting monadic second-order logic CMSO_2 is recognisable. This subsumes graph classes defined by finitely many forbidden (induced) subgraphs and minors, and by the Robertson–Seymour Theorem, all minor-closed graph classes.

Thereby, Theorem 1 applies to e.g. the class of graphs of bounded branchwidth, k-outerplanar graphs, and the class of trees of bounded degree. As a concrete application, we resolve an open question from [38] by showing in Theorem 22 that the exact feasibility of the Lasserre semidefinite programming hierarchy for graph isomorphism can be decided in randomised polynomial-time.

The proof of Theorem 1 combines Courcelle's graph algebras [15] with the homomorphism tensors from [34, 28]. Graph algebras comprise labelled graphs and operations on them such as series and parallel composition. Homomorphism tensors keep track of homomorphism counts of labelled graphs. We show that recognisability and bounded treewidth guarantee that homomorphism tensors yield finite-dimensional representations of suitable graph algebras which certify homomorphism indistinguishability and are efficiently computable. The algorithm in Theorem 1 is randomised as it employs arithmetic modulo random primes to deal with integers which would otherwise grow to exponential size. For graph classes of bounded pathwidth, this issue can be avoided:

▶ **Theorem 2.** *Let $k \geq 1$. If $\mathcal{F}$ is a k-recognisable class of graphs of pathwidth at most $k-1$, then* HomInd$(\mathcal{F})$ *is in polynomial time.*

The connection to Courcelle's theorem motivates considering the parametrised problem HomInd. Here, the CMSO_2-sentence φ allows the graph class to be specified as part of the input. Using results by Courcelle [14], we generalise Theorem 1 in Theorem 3.

HomInd
Input Graphs G and H, a CMSO_2-sentence φ, an integer k.
Parameter $|\varphi| + k$.
Question Are G and H homomorphism indistinguishable over the graph class $\mathcal{F}_{\varphi,k}$ of graphs of treewidth at most $k-1$ satisfying φ?

▶ **Theorem 3.** *There exists a computable function $f\colon \mathbb{N} \to \mathbb{N}$ and a randomised algorithm for* HomInd *of runtime $f(|\varphi| + k)n^{O(k)}$ for $n := \max\{|V(G)|, |V(H)|\}$ which accepts all YES-instances and accepts NO-instances with probability less than one half.*

Equipped with the parametrised perspective offered by HomInd, we finally consider lower bounds on the complexity of this problem. Firstly, we show that it is coW[1]-hard and that the runtime in Theorem 3 is optimal under the Exponential Time Hypothesis (ETH).

▶ **Theorem 4.** HomInd *is* coW[1]*-hard under fpt-reductions. Unless* ETH *fails, there is no algorithm for* HomInd *that runs in time $f(|\varphi| + k)n^{o(|\varphi|+k)}$ for any computable function $f\colon \mathbb{N} \to \mathbb{N}$.*

Secondly, we show that, when disregarding the parametrisation, HomInd is coNP-hard. We do so by showing coNP-hardness of deciding indistinguishability under the k-dimensional Weisfeiler–Leman (WL) algorithm (recently, the same result was independently obtained by [31]). WL is an important heuristic in graph isomorphism and tightly related to notions in finite model theory [13] and graph neural networks [41, 35, 25]. The fastest known algorithm for WL runs in time $O(k^2 n^{k+1} \log n)$ [29], which is exponential when regarding k as part of the input. It was shown that when k is fixed, then the problem is PTIME-complete [23]. Establishing lower bounds on the complexity of WL is a challenging problem [7, 8, 26]. Theorem 5 is a first step towards resolving a question posed by Berkholz [7]: Is the decision problem in Theorem 5 EXPTIME-complete?

▶ **Theorem 5.** *The problem of deciding given graphs G and H and an integer $k \in \mathbb{N}$ whether G and H are k-*WL *indistinguishable is* coNP*-hard under polynomial-time many-one reductions.*

2 Preliminaries

All graphs in this work are finite, undirected, and without multiple edges and loops. For a graph G, we write $V(G)$ for its vertex set and $E(G)$ for its edge set. A *homomorphism* $h\colon F \to G$ from a graph F to a graph G is a map $V(F) \to V(G)$ such that $h(u)h(v) \in E(G)$ for all $uv \in E(F)$. Write $\hom(F, G)$ from the number of homomorphisms from F to G.

Two graphs G and H are *homomorphism indistinguishable* over a class of graph $\mathcal{F}$, in symbols $G \equiv_{\mathcal{F}} H$, if $\hom(F, G) = \hom(F, H)$ for all $F \in \mathcal{F}$. For an integer $n \geq 2$, G and H are *homomorphism indistinguishable over $\mathcal{F}$ modulo n*, in symbols $G \equiv_{\mathcal{F}}^{n} H$, if $\hom(F, G) \equiv \hom(F, H) \mod n$ for all $F \in \mathcal{F}$, cf. [21, 30].

2.1 (Bi)labelled Graphs and Homomorphism Tensors

Let $k, \ell \geq 1$. A *distinctly k-labelled graph* is a tuple $\boldsymbol{F} = (F, \boldsymbol{u})$ where F is a graph and $\boldsymbol{u} \in V(F)^k$ is such that $u_i \neq u_j$ for all $1 \leq i < j \leq k$. We say $u_i \in V(F)$, the i-th entry of $\boldsymbol{u}$, *carries the i-th label.* Write $\mathcal{D}(k)$ for the class of distinctly k-labelled graphs.

A *distinctly (k, ℓ)-bilabelled graph* is a tuple $\boldsymbol{F} = (F, \boldsymbol{u}, \boldsymbol{v})$ where F is a graph and $\boldsymbol{u} \in V(F)^k$ and $\boldsymbol{v} \in V(F)^\ell$ are such that $u_i \neq u_j$ for all $1 \leq i < j \leq k$ and $v_i \neq v_j$ for all $1 \leq i < j \leq \ell$. Note that $\boldsymbol{u}$ and $\boldsymbol{v}$ might share entries. We say $u_i \in V(F)$ and $v_i \in V(F)$ *carry the i-th in-label* and *out-label*, respectively. Weite $\mathcal{D}(k, \ell)$ for the class of distinctly (k, ℓ)-bilabelled graphs.

For a graph G, and $\boldsymbol{F} = (F, \boldsymbol{u}) \in \mathcal{D}(k)$ define the *homomorphism tensor* $\boldsymbol{F}_G \in \mathbb{N}^{V(G)^k}$ of $\boldsymbol{F}$ w.r.t. G whose $\boldsymbol{v}$-th entry is equal to the number of homomorphisms $h\colon F \to G$ such that $h(u_i) = v_i$ for all $i \in [k]$. Analogously, for $\boldsymbol{F} \in \mathcal{D}(k, \ell)$, define $\boldsymbol{F}_G \in \mathbb{N}^{V(G)^k \times V(G)^\ell}$. As the entries of homomorphism tensors are integral, we can view them as vectors in vector spaces over $\mathbb{R}$ as in Section 3 or over finite fields $\mathbb{F}_p$ as in Section 4.

As observed in [34, 28], (bi)labelled graphs and their homomorphism tensors are intriguing due to the following correspondences between combinatorial operations on the former and algebraic operations on the latter:

Dropping labels corresponds to sum-of-entries (soe). For $\boldsymbol{F} = (F, \boldsymbol{u}) \in \mathcal{D}(k)$, define the underlying unlabelled graph $\operatorname{soe}(\boldsymbol{F}) := F$ of $\boldsymbol{F}$. Then for all graphs G, $\hom(\operatorname{soe} \boldsymbol{F}, G) = \sum_{\boldsymbol{v} \in V(G)^k} \boldsymbol{F}_G(\boldsymbol{v}) =: \operatorname{soe}(\boldsymbol{F}_G)$.

Gluing corresponds to Schur products. For $\boldsymbol{F} = (F, \boldsymbol{u})$ and $\boldsymbol{F}' = (F', \boldsymbol{u}')$ in $\mathcal{D}(k)$, define $\boldsymbol{F} \odot \boldsymbol{F}' \in \mathcal{D}(k)$ as the k-labelled graph obtained by taking the disjoint union of F and F' and placing the i-th label at the vertex obtained by merging u_i with u'_i for all $i \in [k]$. Then for every graph G and $\boldsymbol{v} \in V(G)^k$, $(\boldsymbol{F} \odot \boldsymbol{F}')_G(\boldsymbol{v}) = \boldsymbol{F}_G(\boldsymbol{v})\boldsymbol{F}'_G(\boldsymbol{v}) =: (\boldsymbol{F}_G \odot \boldsymbol{F}'_G)(\boldsymbol{v})$. One may similarly define the gluing product of two (k, ℓ)-bilabelled graphs.

Series composition corresponds to matrix products. For bilabelled graphs $\boldsymbol{K} = (K, \boldsymbol{u}, \boldsymbol{v})$ and $\boldsymbol{K}' = (K', \boldsymbol{u}', \boldsymbol{v}')$ in $\mathcal{D}(k, k)$, define $\boldsymbol{K} \cdot \boldsymbol{K}' \in \mathcal{D}(k, k)$ as the bilabelled graph obtained by taking the disjoint union of K and K', merging the vertices v_i and u'_i for $i \in [k]$, and placing the i-th in-label (out-label) on u_i (on v'_i) for $i \in [k]$. Then for all graphs G and $\boldsymbol{x}, \boldsymbol{z} \in V(G)^k$, $(\boldsymbol{K} \cdot \boldsymbol{K}')_G(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\boldsymbol{y} \in V(G)^k} \boldsymbol{K}_G(\boldsymbol{x}, \boldsymbol{y})\boldsymbol{K}'_G(\boldsymbol{y}, \boldsymbol{z}) =: (\boldsymbol{K}_G \cdot \boldsymbol{K}'_G)(\boldsymbol{x}, \boldsymbol{z})$. One may similarly compose a graph in $\mathcal{D}(k, k)$ with a graph in $\mathcal{D}(k)$ obtaining one in $\mathcal{D}(k)$. This operation corresponds to the matrix-vector product.

2.2 Labelled Graphs of Bounded Treewidth

Labelled graphs of bounded treewidth represent the technical foundation of most proofs in this article. Several versions of such have been used in previous works [28, 15, 22].

Let F be a graph. A *tree decomposition* of F is a pair (T, β) where T is a tree and $\beta \colon V(T) \to 2^{V(F)}$ is a map such that

1. the union of the $\beta(t)$ for $t \in V(T)$ is equal to $V(F)$,
2. for every edge $uv \in E(F)$ there exists $t \in V(T)$ such that $\{u, v\} \subseteq \beta(t)$,
3. for every vertex $u \in V(F)$ the set of vertices $t \in V(T)$ such that $u \in \beta(t)$ is connected in T.

The *width* of (T, β) is the maximum over all $|\beta(t)| - 1$ for $t \in V(T)$. The *treewidth* $\operatorname{tw} F$ of F is the minimum width of a tree decomposition of F. A *path decomposition* is a tree decomposition (T, β) where T is a path. The *pathwidth* $\operatorname{pw} F$ of F is the minimum width of a path decomposition of F.

Building on [9, Lemma 8], we show in the following Lemma 6 that every tree decomposition can be rearranged such that the depth of the decomposition tree gives a bound on the number of vertices in the decomposed graph.

▶ **Lemma 6.** *Let $k \geq 1$ and F be a graph such that $\operatorname{tw} F \leq k - 1$ and $|V(F)| \geq k$. Then F admits tree decomposition (T, β) such that*

1. *$|\beta(t)| = k$ for all $t \in V(T)$,*
2. *$|\beta(s) \cap \beta(t)| = k - 1$ for all $st \in E(T)$,*
3. *there exists a vertex $r \in V(T)$ such that the out-degree of every vertex in the rooted tree (T, r) is at most k.*

Proof. By [9, Lemma 8], there exists a tree decomposition (T, β) of F satisfying the first two assertions. Pick a root $r \in V(T)$ arbitrarily. To ensure that the last property holds, the tree decomposition is modified recursively as follows:

By merging vertices, it can be ensured that no two children of r carry the same bag, i.e. that there exist no two children $s_1 \neq s_2$ of r such that $\beta(s_1) = \beta(s_2)$.

For every $v \in \beta(r)$, let $C(v)$ denote the set of all children t of r such that $\beta(r) \setminus \beta(t) = \{v\}$, i.e. $C(v)$ is the set of all children of r whose bags do not contain the vertex v. The collection $C(v)$ for $v \in \beta(r)$ is a partition of the children of r in at most k parts. Note that for two distinct children $t_1 \neq t_2$ in the same part $C(v)$ it holds that $|\beta(t_1) \cap \beta(t_2)| = k - 1$. Rewire the children of r as follows: For every $v \in \beta(r)$ with $C(v) \neq \emptyset$, pick $t \in C(v)$, make t a child of r and all other elements of $C(v)$ children of t. The vertex r now has at most k children and the new tree decomposition still satisfies the first two assertions. Proceed by processing the children of r. ◀

Inspired by Lemma 6, we consider the following family of distinctly labelled graphs. The *depth* of a rooted tree (T, r) is the maximal number of vertices on any path from r to a leaf.

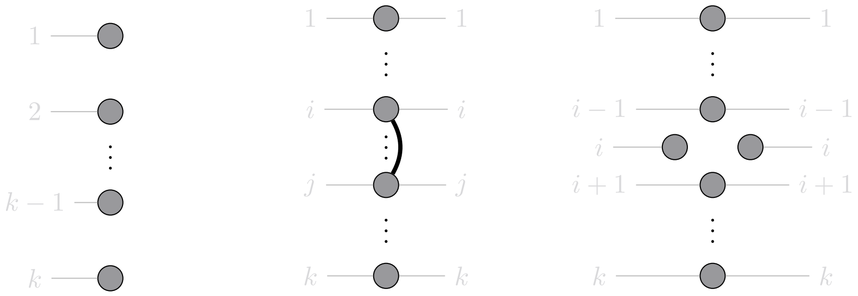
▶ **Definition 7.** *Let $k, d \geq 1$. Define $\mathcal{TW}_d(k)$ as the set of all $\boldsymbol{F} = (F, \boldsymbol{u}) \in \mathcal{D}(k)$ such that F admits a tree decomposition (T, β) of width $\leq k - 1$ satisfying the assertions of Lemma 6 with some $r \in V(T)$ such that $\beta(r) = \{u_1, \dots, u_k\}$ and (T, r) is of depth at most d. Let $\mathcal{TW}(k) \coloneqq \bigcup_{d \geq 1} \mathcal{TW}_d(k)$.*

Every graph in $\mathcal{TW}_d(k)$ has at least k vertices. Conversely, Definition 7 permits the following upper bound on the size of the graphs in $\mathcal{TW}_d(k)$ in terms of d and k.

▶ **Lemma 8.** *Let $k, d \geq 1$. Every $\boldsymbol{F} \in \mathcal{TW}_d(k)$ has at most $\max\{k^d, d\}$ vertices.*

Proof. Let (T, β) and $r \in V(T)$ be as in Definition 7. If $k = 1$, then every vertex in (T, r) has out-degree 1 and F at most d vertices.

Suppose that $k \geq 2$. The proof is by induction on the depth d of the rooted tree (T, r). If $d = 1$, then T contains only a single vertex and $\boldsymbol{F}$ has at most k vertices.



(a) $\boldsymbol{1} \in \mathcal{TW}(k)$. **(b)** $\boldsymbol{A}^{ij} \in \mathcal{D}(k,k)$. **(c)** $\boldsymbol{J}^i \in \mathcal{D}(k,k)$.

Figure 1 The (bi)labelled generators of $\mathcal{TW}(k)$ in wire notion of [34]. A vertex carries in-label (out-label) i if it is connected to the index i on the left (right) by a wire. Actual edges and vertices of the graph are depicted in black.

For the inductive step, let $\boldsymbol{F}$ be of depth d. If r has only a single neighbour s, then $S \coloneqq T - r$ is such that (S, s) is of depth $d-1$. By the inductive hypothesis, $\left|\bigcup_{s\in V(S)} \beta(s)\right| \leq k^d$. Furthermore, $\left|\bigcup_{t\in V(T)} \beta(t) \setminus \bigcup_{s\in V(S)} \beta(s)\right| = 1$. Hence, $\boldsymbol{F}$ has at most $k^{d-1} + 1 \leq k^d$ many vertices.

If r has multiple neighbours, observe that due to Lemma 6 every vertex in $\beta(r)$ is also in $\beta(s)$ for some neighbour s of r. Hence, the number of vertices in $\boldsymbol{F}$ is bounded by the number of vertices covered by the subtrees of $T - r$ rooted in s. Thus, $\boldsymbol{F}$ has at most $k^{d-1} \cdot k \leq k^d$ many vertices. ◀

Clearly, if $\boldsymbol{F} \in \mathcal{TW}(k)$, then $\operatorname{tw}(\operatorname{soe} \boldsymbol{F}) \leq k-1$. Conversely, by Lemma 6, for every F with $\operatorname{tw} F \leq k-1$ and $|V(F)| \geq k$, there exists $\boldsymbol{u} \in V(F)^k$ such that $(F, \boldsymbol{u}) \in \mathcal{TW}(k)$. Thus the underlying unlabelled graphs of the labelled graphs in $\mathcal{TW}(k)$ are exactly the graphs of treewidth $\leq k-1$ on $\geq k$ vertices.

The family $\mathcal{TW}(k)$ is generated by certain small building blocks under series composition and gluing as follows: Let $\boldsymbol{1} \in \mathcal{TW}_1(k)$ be the distinctly k-labelled graph on k vertices without any edges. For $i \in [k]$, let $\boldsymbol{J}^i = (J^i, (1, \dots, k), (1, \dots, i-1, \widehat{i}, i+1, \dots, k))$ the distinctly (k,k)-bilabelled graph with $V(J^i) \coloneqq [k] \cup \{\widehat{i}\}$ and $E(J^i) \coloneqq \emptyset$. Writing $\binom{[k]}{2}$ for the set of pairs of distinct elements in $[k]$, let $\boldsymbol{A}^{ij} = (A^{ij}, (1, \dots, k), (1, \dots, k))$ for $ij \in \binom{[k]}{2}$ be the distinctly (k,k)-bilabelled graph with $V(A^{ij}) \coloneqq [k]$ and $E(A^{ij}) \coloneqq \{ij\}$. These graphs are depicted in Figure 1. Let $\mathcal{B}(k) \coloneqq \{\boldsymbol{J}^i \mid i \in [k]\} \cup \{\boldsymbol{A}^{ij} \mid ij \in \binom{[k]}{2}\}$.

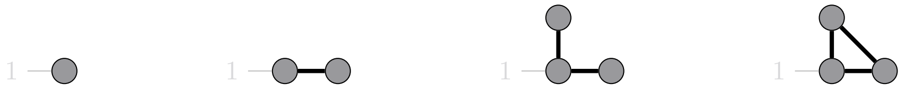
It can be readily verified that if $\boldsymbol{F} \in \mathcal{TW}(k)$ and $\boldsymbol{B} \in \mathcal{B}(k)$, then $\boldsymbol{B} \cdot \boldsymbol{F} \in \mathcal{TW}(k)$. Furthermore, if $\boldsymbol{F}, \boldsymbol{F}' \in \mathcal{TW}(k)$, then $\boldsymbol{F} \odot \boldsymbol{F}' \in \mathcal{TW}(k)$. Conversely, the elements of $\mathcal{B}(k)$ generate $\mathcal{TW}(k)$ in the following sense:

▶ **Lemma 9.** *Let $k \geq 1$. For every $\boldsymbol{F} \in \mathcal{TW}(k)$, one of the following holds:*

1. $\boldsymbol{F} = \boldsymbol{1}$,
2. $\boldsymbol{F} = \prod_{ij \in A} \boldsymbol{A}^{ij} \cdot \boldsymbol{F}'$ *for some* $A \subseteq \binom{[k]}{2}$ *and* $\boldsymbol{F}' \in \mathcal{TW}(k)$ *with less edges than* $\boldsymbol{F}$,
3. $\boldsymbol{F} = \boldsymbol{J}^i \cdot \boldsymbol{F}'$ *for some* $i \in [k]$ *and* $\boldsymbol{F}' \in \mathcal{TW}(k)$ *with less vertices than* $\boldsymbol{F}$,
4. $\boldsymbol{F} = \boldsymbol{F}_1 \odot \boldsymbol{F}_2$ *for* $\boldsymbol{F}_1, \boldsymbol{F}_2 \in \mathcal{TW}(k)$ *on less vertices than* $\boldsymbol{F}$.

2.3 Recognisability and CMSO$_2$

Recognisability is a property of a class of unlabelled graphs which is shared by most named graph classes. It is the assumption of Theorem 1 which rules out esoteric graph classes as constructed in [12, Theorem 1]. We consider the following definition of recognisability.



(a) $\mathbf{1}$. **(b)** $\boldsymbol{P}$. **(c)** $\boldsymbol{Q}$. **(d)** $\boldsymbol{C}$.

Figure 2 Representatives for $\sim^1_{\mathcal{W}}$ and $\mathcal{W}$ the class of paths from Example 11.

▶ **Definition 10** ([11]). *Let $k \geq 1$. For class of unlabelled graphs $\mathcal{F}$, define the equivalence relation $\sim^k_{\mathcal{F}}$ on the class of distinctly k-labelled graphs $\mathcal{D}(k)$ by letting $\boldsymbol{F}_1 \sim^k_{\mathcal{F}} \boldsymbol{F}_2$ if and only if for all $\boldsymbol{K} \in \mathcal{D}(k)$ it holds that*

$$\operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F}_1) \in \mathcal{F} \iff \operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F}_2) \in \mathcal{F}.$$

The class $\mathcal{F}$ is k-recognisable *if $\sim^k_{\mathcal{F}}$ has finitely many equivalence classes. The number of classes of $\sim^k_{\mathcal{F}}$ is the* k-recognisability index *of $\mathcal{F}$.*

To parse Definition 10, first recall that $\boldsymbol{K} \odot \boldsymbol{F}_1$ is the k-labelled graph obtained by gluing $\boldsymbol{K}$ and $\boldsymbol{F}_1$ together at their labelled vertices. The soe-operator drops the labels yielding unlabelled graphs. Intuitively, $\boldsymbol{F}_1 \sim^k_{\mathcal{F}} \boldsymbol{F}_2$ iff both or neither of their underlying unlabelled graphs are in $\mathcal{F}$ and the positions of the labels in $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ is equivalent with respect to membership in $\mathcal{F}$. This intuition is made more concrete in the following example:

▶ **Example 11.** The class $\mathcal{W}$ of all paths is 1-recognisable. Its 1-recognisability index is 4. The equivalence classes are described by the representatives in Figure 2.

Proof. To show that the labelled graphs in Figure 2 cover all equivalence classes, let $\boldsymbol{F} = (F, u)$ be arbitrary. If F is not a path, then $\boldsymbol{F} \sim^1_{\mathcal{W}} \boldsymbol{C}$. Indeed, for every $\boldsymbol{K} \in \mathcal{D}(1)$, F is a subgraph of $\operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F})$. Hence, regardless of $\boldsymbol{K}$, both $\operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F})$ and $\operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{C})$ are not paths. If F is a path,, then $\boldsymbol{F}$ and $\mathbf{1}$, $\boldsymbol{P}$, or $\boldsymbol{Q}$ are equivalent depending on whether the degree of u is 0, 1, or 2.

To show that the representatives in Figure 2 are in distinct classes, observe for example that $\operatorname{soe}(\boldsymbol{P} \odot \boldsymbol{P}) \in \mathcal{W}$ while $\operatorname{soe}(\boldsymbol{P} \odot \boldsymbol{Q}) \notin \mathcal{W}$, thus $\boldsymbol{P} \not\sim^1_{\mathcal{W}} \boldsymbol{Q}$. Similarly, $\operatorname{soe}(\mathbf{1} \odot \boldsymbol{Q}) \in \mathcal{W}$ whereas $\operatorname{soe}(\boldsymbol{P} \odot \boldsymbol{Q}) \notin \mathcal{W}$, thus $\mathbf{1} \not\sim^1_{\mathcal{W}} \boldsymbol{P}$. ◀

A more involved example is the following. Analogously, one may argue that every class defined by forbidden minors is recognisable.

▶ **Example 12.** Let $\mathcal{F}$ be the family of H-subgraph-free graphs for some graph H. Then $\mathcal{F}$ is k-recognisable for every $k \geq 1$.

Proof. Suppose H has m vertices. For a distinctly k-labelled graph $\boldsymbol{F} = (F, \boldsymbol{u})$, consider the set $\mathcal{H}(\boldsymbol{F})$ of (isomorphism types of) distinctly k-labelled graphs $\boldsymbol{F}' = (F', \boldsymbol{u})$ where F' is a subgraph of F such that $V(F') \supseteq \{u_1, \dots, u_k\}$ has at most $k + m$ vertices. Clearly, there are only finitely many possible sets $\mathcal{H}(\boldsymbol{F})$. Furthermore, if $\mathcal{H}(\boldsymbol{F}_1) = \mathcal{H}(\boldsymbol{F}_2)$, then $\boldsymbol{F}_1 \sim^k_{\mathcal{F}} \boldsymbol{F}_2$. Indeed, if $\boldsymbol{K} \in \mathcal{D}(k)$ is such that $\operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F}_1)$ contains H as a subgraph, then so does $\operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F}_2)$ since $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ contain the same subgraphs on $k + m$ vertices containing their labelled vertices. ◀

Courcelle [14] proved that every CMSO_2-definable graph class is *recognisable*, i.e. it is k-recognisable for every $k \in \mathbb{N}$. Conversely, Bojańczyk and Pilipczuk [11] proved that if a recognisable class $\mathcal{F}$ has bounded treewidth, then it is CMSO_2-definable. Furthermore, they conjecture that k-recognisability is a sufficient condition for a graph class of treewidth at most $k - 1$ to be CMSO_2-definable.

Here, *counting monadic second-order logic* CMSO_2 is the extension of first-order logic by (1) variables that range over sets of vertices or edges, or over edges, (2) atomic formulas $\operatorname{inc}(x, y)$ which evaluate to true if x is assigned a vertex v and y is assigned an edge e such that v is incident with e, and (3) atomic formulas $\operatorname{card}_{p,q}(X)$ for integers $p, q \in \mathbb{N}$ and set variables X expressing that $|X| \equiv p \mod q$. See [15] for further details.

3 Decidability

As a first step, we show that the problem HomInd($\mathcal{F}$) is decidable for every k-recognisable graph class $\mathcal{F}$ of treewidth at most $k-1$. We do so by establishing a bound on the maximum size of a graph $F \in \mathcal{F}$ for which $\hom(F, G) = \hom(F, H)$ needs to be checked in order to conclude whether $G \equiv_{\mathcal{F}} H$. For a graph class $\mathcal{F}$ and $\ell \in \mathbb{N}$, define the class $\mathcal{F}_{\leq \ell} \coloneqq \{F \in \mathcal{F} \mid |V(F)| \leq \ell\}$.

▶ **Theorem 13.** *Let $k \geq 1$. Let $\mathcal{F}$ be a graph class of treewidth $\leq k-1$ with k-recognisability index C. For graphs G and H on at most n vertices, with $f_{k,C}(n) \coloneqq \max\{k^{2Cn^k}, 2Cn^k\}$,*

$$G \equiv_{\mathcal{F}} H \iff G \equiv_{\mathcal{F}_{\leq f_{k,C}(n)}} H.$$

Fix throughout a graph class $\mathcal{F}$ as in Theorem 13. In reminiscence of Courcelle's theorem, we let Q denote the set of equivalence classes of $\sim^k_{\mathcal{F}}$, as defined in Definition 10, and call them *states*. A state $q \in Q$ is *accepting* if for an (and equivalently, every) $\boldsymbol{F}$ in q it holds that $\operatorname{soe}(\boldsymbol{F}) \in \mathcal{F}$. Write $A \subseteq Q$ for the set of all accepting states.

To every state $q \in Q$, we associate a finite-dimensional vector space spanned by the homomorphism tensors of the k-labelled graphs $\boldsymbol{F}$ that belong to state q. We show that these vector spaces certify homomorphism indistinguishability. Using a dimensionality argument, we show that these vector spaces are spanned by homomorphism tensors of graphs whose size is bounded by the function f from Theorem 13. To that end, we decompose the labelled graphs $\boldsymbol{F} \in \mathcal{TW}(k)$ using the operations considered in Lemma 9.

Formally, we associate to a state $q \in Q$ and an integer $d \geq 1$ the vector space[1]

$$S_d(q) \coloneqq \langle \{\boldsymbol{F}_G \oplus \boldsymbol{F}_H \mid \boldsymbol{F} \in \mathcal{TW}_d(k) \text{ in state } q\} \rangle \subseteq \mathbb{R}^{V(G)^k \cup V(H)^k}.$$

Here, $\boldsymbol{F}$ is a k-labelled graph of bounded treewidth in the state q. The vector $\boldsymbol{F}_G \oplus \boldsymbol{F}_H \coloneqq \binom{\boldsymbol{F}_G}{\boldsymbol{F}_H} \in \mathbb{R}^{V(G)^k \cup V(H)^k}$ is obtained by stacking the homomorphisms vectors of $\boldsymbol{F}$ w.r.t. G and H. Since $\mathcal{TW}_d(k) \subseteq \mathcal{TW}_{d+1}(q)$, the space $S_d(q)$ is a subspace of $S_{d+1}(q)$ for every $d \geq 1$. Ultimately, we are interested in $S(q) \coloneqq \bigcup_{d \geq 1} S_d(q)$, i.e. the vector space spanned by the homomorphism vectors of all labelled graphs of treewidth $\leq k-1$ in state q.

By the following Lemma 14, the vectors in $S(q)$ for $q \in A$ can be used to infer whether G and H are homomorphism indistinguishable over $\mathcal{F}$. For a labelled graph $\boldsymbol{F} \in \mathcal{TW}(k)$, the number $\boldsymbol{1}_G^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H)$ is equal to the number of homomorphisms from the underlying unlabelled graph of $\boldsymbol{F}$ to G. This observation readily yields the backward implication in Lemma 14. For the forward implication, observe that the space $S(q)$ is spanned by homomorphism tensors $\boldsymbol{F}_G \oplus \boldsymbol{F}_H$, which satisfy the assertion by assumption.

▶ **Lemma 14.** *Two graph G and H are homomorphism indistinguishable over $\mathcal{F}_{\geq k} \coloneqq \{F \in \mathcal{F} \mid |V(F)| \geq k\}$ if and only if $\boldsymbol{1}_G^T v = \boldsymbol{1}_H^T v$ for every $q \in A$ and every $v \in S(q)$.*

[1] Wlog we may suppose that $V(G)$ and $V(H)$ are disjoint.

Proof. For the forward direction, note that $S(q)$ is spanned by the $\boldsymbol{F}_G \oplus \boldsymbol{F}_H$ where $\boldsymbol{F} \in \mathcal{TW}(k)$ is in state q. Let $\boldsymbol{F}$ in $q \in A$ be arbitrary. Then $\operatorname{soe}(\boldsymbol{F}) =: F \in \mathcal{F}_{\geq k}$ and it holds that $\mathbf{1}_G^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H) = \operatorname{soe}(\boldsymbol{F}_G) = \hom(F, G) = \hom(F, H) = \mathbf{1}_H^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H)$. Conversely, let $F \in \mathcal{F}_{\geq k}$ be arbitrary. Since $\operatorname{tw} F \leq k-1$, by Lemma 6, there exists $\boldsymbol{u} \in V(F)^k$ such that $\boldsymbol{F} := (F, \boldsymbol{u}) \in \mathcal{TW}(k)$. Furthermore, $\boldsymbol{F}$ belongs to some accepting $q \in Q$. Thus, $\boldsymbol{F}_G \oplus \boldsymbol{F}_H \in S(q)$, and hence $\hom(F, G) = \mathbf{1}_G^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H) = \mathbf{1}_H^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H) = \hom(F, H)$. ◀

By Lemma 8, the space $S_d(q)$ for $d \geq 1$ is spanned by homomorphism tensors of graphs of size $\max\{k^d, d\}$. Thus, Theorem 13 follows once we establish that $S_{d'}(q) = S(q)$ for all $q \in Q$ and $d' := 2Cn^k$. This d' arises as an upper bound on the dimension of the space $\bigoplus_{q \in Q} S(q)$. The spaces $\bigoplus_{q \in Q} S_d(q)$ for $d \geq 1$ form a chain of nested subspaces in $\bigoplus_{q \in Q} S(q)$. The following Lemma 15 shows that once this chain becomes stationary, then the maximal subspace is reached.

▶ **Lemma 15.** *If $S_d(q) = S_{d+1}(q)$ for $d \geq 1$ and all $q \in Q$, then $S_d(q) = S(q)$ for all $q \in Q$. In particular, $S_{2Cn^k}(q) = S(q)$ for all $q \in Q$.*

The proof of Lemma 15 relies on the properties of the relation $\sim_{\mathcal{F}}^k$. In particular, it uses the fact that series composition and gluing, the operations under which $\mathcal{TW}(k)$ is generated by Lemma 9, preserve the relation $\sim_{\mathcal{F}}^k$.

▶ **Lemma 16.** *For $\boldsymbol{F}, \boldsymbol{F}', \boldsymbol{F}_1, \boldsymbol{F}_2, \boldsymbol{F}_1', \boldsymbol{F}_2' \in \mathcal{D}(k)$, $\boldsymbol{L} \in \mathcal{D}(k, k)$,*

1. *if $\boldsymbol{F}_1 \sim_{\mathcal{F}}^k \boldsymbol{F}_1'$ and $\boldsymbol{F}_2 \sim_{\mathcal{F}}^k \boldsymbol{F}_2'$, then $\boldsymbol{F}_1 \odot \boldsymbol{F}_2 \sim_{\mathcal{F}}^k \boldsymbol{F}_1' \odot \boldsymbol{F}_2'$,*
2. *if $\boldsymbol{F} \sim_{\mathcal{F}}^k \boldsymbol{F}'$, then $\boldsymbol{L} \cdot \boldsymbol{F} \sim_{\mathcal{F}}^k \boldsymbol{L} \cdot \boldsymbol{F}'$.*

Proof. Let $\boldsymbol{K} = (K, \boldsymbol{u}) \in \mathcal{D}(k)$ be arbitrary. Then $\operatorname{soe}((\boldsymbol{K} \odot \boldsymbol{F}_1) \odot \boldsymbol{F}_2) \in \mathcal{F} \Leftrightarrow \operatorname{soe}((\boldsymbol{K} \odot \boldsymbol{F}_1) \odot \boldsymbol{F}_2') \in \mathcal{F} \Leftrightarrow \operatorname{soe}(\boldsymbol{K} \odot \boldsymbol{F}_1' \odot \boldsymbol{F}_2') \in \mathcal{F}$.

For a (k, k)-bilabelled graph $\boldsymbol{L} = (L, \boldsymbol{u}, \boldsymbol{v})$, write $\boldsymbol{L}^* := (L, \boldsymbol{v}, \boldsymbol{u})$ for the (k, k)-bilabelled graph obtained by swapping the in-labels and out-labels. Then $\operatorname{soe}(\boldsymbol{K} \odot (\boldsymbol{L} \cdot \boldsymbol{F})) = \operatorname{soe}((\boldsymbol{L}^* \cdot \boldsymbol{K}) \odot \boldsymbol{F})$. Thus the second claim follows from the first. ◀

The algebraic operations on homomorphism tensors corresponding to series composition and gluing are the matrix-vector product and Schur product. Crucially, these operations are linear and bilinear, respectively. This allows Lemma 15 to be proven by structural induction along Lemma 9.

Proof of Lemma 15. We argue that $S_d(q) \supseteq S_{d+i}(q)$ for all $i \geq 1$ by induction on i. The base case holds by assumption. The space $S_{d+i+1}(q)$ is spanned by the vectors $\boldsymbol{F}_G \oplus \boldsymbol{F}_H$ where $\boldsymbol{F} \in \mathcal{TW}_{d+i+1}(k)$ is in q. For such $\boldsymbol{F}$, by Lemma 9, there exist $A \subseteq \binom{[k]}{2}$, $L \subseteq [k]$, and $\boldsymbol{F}^\ell \in \mathcal{TW}_{d+i}(k)$ for $\ell \in L$ such that

$$\boldsymbol{F} = \prod_{ij \in A} \boldsymbol{A}^{ij} \cdot \bigodot_{\ell \in L} \boldsymbol{J}^\ell \boldsymbol{F}^\ell.$$

Let q_ℓ denote the state of $\boldsymbol{F}^\ell$. By assumption, there exist $\boldsymbol{K}^{\ell m} \in \mathcal{TW}_d(k)$ in state q_ℓ and $\alpha_m \in \mathbb{R}$ such that $\boldsymbol{F}_G^\ell \oplus \boldsymbol{F}_H^\ell = \sum \alpha_m \boldsymbol{K}_G^{\ell m} \oplus \boldsymbol{K}_H^{\ell m}$. By Lemma 16,

$$\boldsymbol{F} \sim_{\mathcal{F}}^k \prod_{ij \in A} \boldsymbol{A}^{ij} \cdot \bigodot_{\ell \in L} \boldsymbol{J}^\ell \boldsymbol{K}^{\ell m}$$

for all m. Thus, $\boldsymbol{F}_G \oplus \boldsymbol{F}_H$ can be written as linear combination of vectors in $S_{d+i}(q) \subseteq S_d(q)$, by induction. For the final claim, consider the chain of nested subspaces

$$\bigoplus_{q \in Q} S_1(q) \subseteq \bigoplus_{q \in Q} S_2(q) \subseteq \cdots \subseteq \bigoplus_{q \in Q} S(q).$$

By what was just shown, for every $d \geq 1$, either $\bigoplus_{q \in Q} S_d(q)$ is a proper subspace of $\bigoplus_{q \in Q} S_{d+1}(q)$ or $\bigoplus_{q \in Q} S_d(q) = \bigoplus_{q \in Q} S(q)$. Since the dimension of $\bigoplus_{q \in Q} S(q)$ is at most $2Cn^k$, the chain becomes stationary after at most $2Cn^k$ steps. ◀

This concludes the preparations for the proof of Theorem 13.

Proof of Theorem 13. It suffices to prove the backward implication. Since $k \leq k^{2Cn^k}$, it suffices to show that $G \equiv_{\mathcal{F}_{\geq k}} H$ by verifying the condition in Lemma 14. By Lemma 15, $S_d(q) = S(q)$ for $d := 2Cn^k$ and all $q \in Q$. Hence, $S(q)$ is spanned by the $\boldsymbol{F}_G \oplus \boldsymbol{F}_H$ where $\boldsymbol{F} \in \mathcal{TW}_d(k)$ is in state q. By Lemma 8, these graphs have at most $\max\{k^d, d\} = \max\{k^{2Cn^k}, 2Cn^k\}$ vertices. Thus, $\mathbf{1}_G^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H) = \hom(\operatorname{soe} \boldsymbol{F}, G) = \hom(\operatorname{soe} \boldsymbol{F}, H) = \mathbf{1}_H^T(\boldsymbol{F}_G \oplus \boldsymbol{F}_H)$, as desired. ◀

Finally, we adapt the techniques developed so far to prove the following analogue of Theorem 13 for graph classes of bounded pathwidth. In contrast to Theorem 13, the function in Theorem 17 bounding the size of the graphs which need to be considered is polynomial. The proof is deferred to the full version.

▶ **Theorem 17.** *Let $k \geq 1$. Let $\mathcal{F}$ be a graph class of pathwidth $\leq k-1$ with k-recognisability index C. For graphs G and H on at most n vertices, with $f_{k,C}(n) := 2Cn^k + k - 1$,*

$$G \equiv_{\mathcal{F}} H \iff G \equiv_{\mathcal{F}_{\leq f_{k,C}(n)}} H.$$

4 Modular Homomorphism Indistinguishability in Polynomial Time

The insight that yielded Theorem 13 is that the chain of vector spaces $S_1(q) \subseteq \dots \subseteq S_d(q) \subseteq S_{d+1}(q) \subseteq \dots$ reaches a fixed point after polynomially many steps. In this section, we strengthen this result by showing that bases $B(q)$ for the spaces $S(q)$ can be computed efficiently. A technical difficulty arising here is that the numbers produced in the process can be of doubly exponential magnitude. In order to overcome this problem, we first consider homomorphism indistinguishability modulo primes. See [21, 30], for background on modular homomorphism indistinguishability.

ModHomInd($\mathcal{F}$)
Input Graphs G and H, a prime p in binary.
Question Are G and H homomorphism indistinguishable over $\mathcal{F}$ modulo p?

▶ **Theorem 18.** *Let $k \geq 1$. If $\mathcal{F}$ is a k-recognisable graph class of treewidth $\leq k-1$, then* ModHomInd($\mathcal{F}$) *is in polynomial time.*

The algorithm yielding Theorem 18 is formally stated as Algorithm 1. The idea is to iteratively compute bases $B(q)$ for the spaces

$$S(q) := \langle \{\boldsymbol{F}_G \oplus \boldsymbol{F}_H \mid \boldsymbol{F} \in \mathcal{TW}(k) \text{ in state } q\} \rangle \subseteq \mathbb{F}_p^{V(G)^k \cup V(H)^k}.$$

Initially, all $B(q)$ are empty. Only $B(q_0)$ where q_0 is the state of $\mathbf{1} \in \mathcal{TW}(k)$ from Figure 1a contains the homomorphism vector $\mathbf{1}_G \oplus \mathbf{1}_H$. Subsequently, the operations from Lemma 9 are applied to compute new homomorphism vectors. For every new vector belonging to state q, it is checked whether it is a linear combination of the already computed basis vectors in $B(q)$. If not, it is added to $B(q)$. Analogous to Lemma 15, this process reaches a fixed point after a polynomial number of iterations. At this point, the computed $B(q)$ are bases for the $S(q)$. Finally, Lemma 14 can be invoked to conclude whether the input graphs are homomorphism indistinguishable over $\mathcal{F}$ modulo p.

Algorithm 1 is supplied with a hard-coded description of the graph class $\mathcal{F}$. To that end, consider the following objects. Write $\pi \colon \mathcal{TW}(k) \to Q$ for the map that associates an $\boldsymbol{F} \in \mathcal{TW}(k)$ to its state $q \in Q$. Write q_0 for the state of $\mathbf{1} \in \mathcal{TW}(k)$. Furthermore, write $g \colon Q \times Q \to Q$ and $b_{\boldsymbol{B}} \colon Q \to Q$ for every $\boldsymbol{B} \in \mathcal{B}(k)$ such that

$$g(\pi(\boldsymbol{F}), \pi(\boldsymbol{F}')) = \pi(\boldsymbol{F} \odot \boldsymbol{F}'), \tag{1}$$

$$b_{\boldsymbol{B}}(\pi(\boldsymbol{F})) = \pi(\boldsymbol{B} \cdot \boldsymbol{F}). \tag{2}$$

for every $\boldsymbol{F}, \boldsymbol{F}' \in \mathcal{TW}(k)$ and $\boldsymbol{B} \in \mathcal{B}(k)$. Note that Q, A, g, q_0 and the $b_{\boldsymbol{B}}$, $\boldsymbol{B} \in \mathcal{B}(k)$, are finite objects, which can be hard-coded. The map π does not need to be computable and is only needed for analysing the algorithm.

Algorithm 1 ModHomInd($\mathcal{F}$) for k-recognisable $\mathcal{F}$ of treewidth $\leq k-1$.

```
   Input: graphs $G$ and $H$, a prime $p$ in binary.
   Data: $k$, $Q$, $A$, $q_0$, $g$, $b_{\boldsymbol{B}}$ for $\boldsymbol{B} \in \mathcal{B}(k)$.
   Output: whether $G \equiv^p_{\mathcal{F}} H$.
 1 With brute force check whether $G$ and $H$ are homomorphism indistinguishable over
    the finite graph class $\mathcal{F}_{\leq k}$ modulo $p$ and reject if not;
 2 $B(q_0) \leftarrow \{\mathbf{1}_G \oplus \mathbf{1}_H\} \subseteq \mathbb{F}_p^{V(G)^k \cup V(H)^k}$;
 3 $B(q) \leftarrow \emptyset \subseteq \mathbb{F}_p^{V(G)^k \cup V(H)^k}$ for all $q \neq q_0$;
 4 repeat
 5     foreach $\boldsymbol{B} \in \mathcal{B}(k)$, $q \in Q$, $v \in B(q)$ do
 6         $w \leftarrow (\boldsymbol{B}_G \oplus \boldsymbol{B}_H)v := \begin{pmatrix} \boldsymbol{B}_G & 0 \\ 0 & \boldsymbol{B}_H \end{pmatrix} v$;
 7         if $w \notin \langle B(b_{\boldsymbol{B}}(q)) \rangle$ then
 8             add $w$ to $B(b_{\boldsymbol{B}}(q))$;
 9     foreach $q_1, q_2 \in Q$, $v_1 \in B(q_1)$, $v_2 \in B(q_2)$ do
10         $w \leftarrow v_1 \odot v_2$;
11         if $w \notin \langle B(g(q_1, q_2)) \rangle$ then
12             add $w$ to $B(g(q_1, q_2))$;
13 until none of the $B(q)$, $q \in Q$, are updated;
14 if $\mathbf{1}_G^T v = \mathbf{1}_H^T v$ for all $q \in A$ and $v \in B(q)$ then
15     accept;
16 else
17     reject;
```

▶ **Lemma 19.** *Write $n := \max\{|V(G)|, |V(H)|\}$ and $C := |Q|$. There exists a computable function f such that Algorithm 1 runs in time $f(k, C) n^{O(k)} (\log p)^{O(1)}$.*

Proof. Consider the following individual runtimes: Counting homomorphisms of a graph on k vertices into a graph on n vertices, can be done in time $O(n^k)$. Hence Line 1 requires time $f(k)n^k$ for some computable function f.

Throughout the execution of Algorithm 1, the vectors in each $B(q)$, $q \in Q$, are linearly independent. Thus, $|B(q)| \leq \dim S(q) \leq 2n^k$ and $\sum_{q \in Q} |B(q)| \leq 2Cn^k$. Hence, the body of the loop in Line 1 is entered at most $2Cn^k$ many times.

The loop in Line 1 iterates over at most $k^2 \cdot C \cdot 2n^k$ many objects. Computing the vector w takes polynomial time in $2n^k \cdot \log p$. The same holds for checking the condition in Line 1, e.g. via Gaussian elimination. The loop in Line 1 iterates over at most $C^2 \cdot (2n^k)^2$ many objects. Finally, checking the condition in Line 1 takes $C \cdot 2n^k \cdot (\log p)^{O(1)}$ many steps. ◀

The following Lemma 20 implies that Algorithm 1 is correct.

▶ **Lemma 20.** *When Algorithm 1 terminates, $B(q)$ spans $S(q)$ for all $q \in Q$.*

Proof. First observe that the invariant $B(q) \subseteq S(q)$ for all $q \in Q$ is preserved throughout Algorithm 1. Indeed, for example in Line 1, since $v \in B(q) \subseteq S(q)$, it can be written as linear combination of $\boldsymbol{F}_G \oplus \boldsymbol{F}_H$ for $\boldsymbol{F} \in \mathcal{TW}(k)$ of state q. Because $\boldsymbol{B} \cdot \boldsymbol{F}$ is in state $b_{\boldsymbol{B}}(q)$ by Equation (2), $(\boldsymbol{B}_G \oplus \boldsymbol{B}_H)v$ is in the span of $\boldsymbol{B}_G\boldsymbol{F}_G \oplus \boldsymbol{B}_H\boldsymbol{F}_H \in S(b_{\boldsymbol{B}}(q))$.

Now consider the converse inclusion. The proof is by induction on the structure in Lemma 9. By initialisation, $\boldsymbol{1}_G \oplus \boldsymbol{1}_H$ is in the span of $B(q_0)$.

For the inductive step, suppose that $\boldsymbol{F} \in \mathcal{TW}(k)$ of state $q \in Q$ is such that $\boldsymbol{F}_G \oplus \boldsymbol{F}_H = \sum_{v \in B(q)} \alpha_v v$ for some coefficients $\alpha_v \in \mathbb{F}_p$. Let $\boldsymbol{B} \in \mathcal{B}(k)$ and $\boldsymbol{F}' := \boldsymbol{B} \cdot \boldsymbol{F}$. Then $(\boldsymbol{B}_G \oplus \boldsymbol{B}_H)v$ is in the span of $B(b_{\boldsymbol{B}}(q))$ for all $v \in B(q)$ by the termination condition. Hence, $\boldsymbol{F}'_G \oplus \boldsymbol{F}'_H = \sum_{v \in B(q)} \alpha_v (\boldsymbol{B}_G \oplus \boldsymbol{B}_H)v$ is in the span of $B(b_{\boldsymbol{B}}(q))$.

Let $\boldsymbol{F}^1, \boldsymbol{F}^2 \in \mathcal{TW}(k)$ of states $q_1, q_2 \in Q$ be such that $\boldsymbol{F}^1_G \oplus \boldsymbol{F}^1_H = \sum_{v \in B(q_1)} \alpha_v v$ and $\boldsymbol{F}^2_G \oplus \boldsymbol{F}^2_H = \sum_{w \in B(q_2)} \beta_w w$ for some coefficients $\alpha_v, \beta_w \in \mathbb{F}_p$. Since the algorithm terminated, all $v \odot w$ for $v \in B(q_1)$ and $w \in B(q_2)$ are in the span of $B(g(q_1, q_2))$. Then $(\boldsymbol{F}^1 \odot \boldsymbol{F}^2)_G \oplus (\boldsymbol{F}^1 \odot \boldsymbol{F}^2)_H = (\boldsymbol{F}^1_G \oplus \boldsymbol{F}^1_H) \odot (\boldsymbol{F}^2_G \oplus \boldsymbol{F}^2_H) = \sum_{v \in B(q_1), w \in B(q_2)} \alpha_v \beta_w (v \odot w)$ is in the span of $B(g(q_1, q_2))$. ◀

This concludes the preparations for the proof of Theorem 18:

Proof of Theorem 18. Lemma 20 implies that the conditions in Lemma 14 and Line 1 are equivalent. Thus, G and H are homomorphism indistinguishable over $\mathcal{F}_{\geq k}$ modulo p if and only if the condition in Line 1 holds. The runtime bound is given in Lemma 19. ◀

5 Randomised Polynomial Time

In this section, we give a randomised polynomial-time reduction from HomInd($\mathcal{F}$) to ModHomInd($\mathcal{F}$). Thereby, we prove Theorems 1 and 2. Theorems 13 and 17 give bounds N on the size of the largest graph in $\mathcal{F}$ which needs to be considered in order to conclude whether two graphs on at most n vertices are homomorphism indistinguishable over $\mathcal{F}$. A graph on at most N vertices may have at most n^N homomorphisms to a graph on n vertices. Thus, for graphs on at most n vertices, homomorphism indistinguishability over $\mathcal{F}$ is the same as homomorphism indistinguishability over $\mathcal{F}$ modulo any number greater than n^N. Equipped with the following Lemma 21, which is derived from the Chinese Remainder Theorem and the Prime Number Theorem, we show Theorems 1 and 2. Let log denote the logarithm to base 2.

▶ **Lemma 21.** *Let $N, n \in \mathbb{N}$ be such that $N \log n \geq e^{2000}$. Let $\mathcal{F}$ be a graph class and G and H be graphs on at most n vertices. If $G \not\equiv_{\mathcal{F}_{\leq N}} H$ then the probability that a random prime $N \log n < p \leq (N \log n)^2$ is such that $G \equiv^p_{\mathcal{F}_{\leq N}} H$ is at most $\frac{2}{N \log n}$.*

For graph classes of bounded pathwidth, the bound on N from Theorem 17 is polynomial in n. Thus, one can enumerate all primes $N \log n < p \leq (N \log n)^2$ in polynomial time and invoke ModHomInd($\mathcal{F}$).

▶ **Theorem 2.** *Let $k \geq 1$. If $\mathcal{F}$ is a k-recognisable class of graphs of pathwidth at most $k - 1$, then* HomInd($\mathcal{F}$) *is in polynomial time.*

For graph classes of bounded treewidth, the bound on N from Theorem 13 is exponential in n. The randomised algorithm yielding Theorem 1 stated in the full version samples primes of polynomial size and invokes $\textsc{ModHomInd}(\mathcal{F})$. Lemma 21 implies that a random prime p of appropriate size certifies that $G \not\equiv_{\mathcal{F}} H$ with high probability. This yields Theorem 1:

▶ **Theorem 1.** *Let $k \geq 1$. If $\mathcal{F}$ is a k-recognisable class of graphs of treewidth at most $k-1$, then $\textsc{HomInd}(\mathcal{F})$ is in* coRP*.*

6 Fixed-Parameter Tractability

In this section, we deduce Theorem 3 from Theorem 1. The challenge is to efficiently compute the data describing the graph class $\mathcal{F}_{\varphi,k}$ for Algorithm 1 from the CMSO_2-sentence φ and k. That this can be done was proven by Courcelle [15]. More precisely, Courcelle proved that for every CMSO_2-sentence φ and integer k one can compute a finite automaton processing expressions which encode (tree decompositions of) graphs of bounded treewidth. It is this automaton from which the data required by Algorithm 1 can be constructed.

▶ **Theorem 3.** *There exists a computable function $f \colon \mathbb{N} \to \mathbb{N}$ and a randomised algorithm for* $\textsc{HomInd}$ *of runtime $f(|\varphi| + k)n^{O(k)}$ for $n := \max\{|V(G)|, |V(H)|\}$ which accepts all* YES*-instances and accepts* NO*-instances with probability less than one half.*

For graph classes of bounded pathwidth, the analogous problem can be decided deterministically in the same time, cf. the full version.

7 Lasserre in Polynomial Time

By phrasing graph isomorphism as an integer program, heuristics from integer programming can be used to attempt to solve graph isomorphism. Prominent heuristics are the Sherali–Adams linear programming hierarchy and the Lasserre semidefinite programming hierarchy. While the (approximate) feasibility of each level of these hierarchies can be decided efficiently, it is known that a linear number of levels is required to decide graph isomorphism for all graphs [4, 27, 38].

In [38], for each $t \geq 1$, feasibility of the t-th level of the Lasserre hierarchy was characterised as homomorphism indistinguishability relation over the graph class $\mathcal{L}_t$ which was constructed in the same paper. Moreover, the authors asked whether there is a polynomial-time algorithm for deciding these relations. In this section, we give a randomised algorithm for this problem which is polynomial-time for every level.

The Lasserre semidefinite program can be solved approximately in polynomial time using e.g. the ellipsoid method. How to decide *exact* feasibility is generally unknown [2]. Since the graph class $\mathcal{L}_t$ is a minor-closed and of treewidth at most $3t-1$, Theorem 1 immediately yields a randomised polynomial-time algorithm for each level of the hierarchy. However, it is not clear how to compute the data describing $\mathcal{L}_t$ given t. The following Theorem 22 overcomes this problem by making the dependence on the parameter t effective:

▶ **Theorem 22.** *There exists a computable function $f \colon \mathbb{N} \to \mathbb{N}$ and a randomised algorithm deciding given graphs G and H on at most n vertices and an integer $t \geq 1$ whether the level-t Lasserre relaxation of the integer program for $G \cong H$ has an exact solution. This algorithm always runs in time $f(t)n^{O(t)}$, accepts all* YES*-instances and accepts* NO*-instances with probability less than one half.*

8 Lower Bounds

In this final section, we establish two hardness results for the problem HomInd. In both cases, we show hardness for families of minor-closed graph classes. The approaches are orthogonal in the sense that the reduction yielding coNP-hardness in Theorem 5 is from a fixed-parameter tractable problem while the reduction yielding coW[1]-hardness in Theorem 4 is not polynomial-time.

coNP-Hardness. The first hardness result concerns deciding whether two graphs are indistinguishable under the k-dimensional Weisfeiler–Leman algorithm when k is part of the input. By [20, 13, 19], k-WL indistinguishability coincides with homomorphism indistinguishability over the class of graphs of treewidth at most k. Hence, the problem in Theorem 5 is clearly a special case of HomInd, i.e. with φ set to true. Thus, when disregarding the parametrisation, HomInd is coNP-hard under polynomial-time many-one reductions. We obtain Theorem 5 by reducing the NP-complete problem of deciding whether a graph of bounded degree has treewidth $\leq k$ [10]. The reduction is based on the ubiquitous CFI construction [13].

▶ **Theorem 5.** *The problem of deciding given graphs G and H and an integer $k \in \mathbb{N}$ whether G and H are k-*WL *indistinguishable is* coNP*-hard under polynomial-time many-one reductions.*

Towards the proof of Theorem 5, we recall the following version of the classical CFI graphs [13] from [37]. Let G be a connected graph and $U \colon V(G) \to \mathbb{Z}_2$ a function from G to the group on two elements $\mathbb{Z}_2$. For a vertex $v \in V(G)$, write $E(v) \subseteq E(G)$ for the set of edges incident to v. The graph G_U has vertices (v, S) for every $v \in V(G)$ and $S \colon E(v) \to \mathbb{Z}_2$ such that $\sum_{e \in E(v)} S(e) = U(v)$. Two vertices (u, S) and (v, T) are adjacent in G_U if $uv \in E(G)$ and $S(uv) + T(uv) = 0$. Note that $|V(G_U)| = \sum_{v \in V(G)} 2^{\deg(v)-1}$. By [37, Lemma 3.2], if $\sum_{v \in V(G)} U(v) = \sum_{v \in V(G)} U'(v)$ for $U, U' \colon V(G) \to \mathbb{Z}_2$, then $G_U \cong G_{U'}$. We may thus write G_0 and G_1 for the *even* and the *odd* CFI graph of G. We recall the following properties:

▶ **Lemma 23** ([37, Corollary 3.7])**.** *Let G be a connected graph and $U \colon V(G) \to \mathbb{Z}_2$. Then the following are equivalent:*

1. $G_0 \cong G_U$,
2. $\sum_{v \in V(G)} U(v) = 0$,
3. $\hom(G, G_0) = \hom(G, G_U)$.

Proof of Theorem 5. For a graph G, write $\Delta(G)$ for its maximum vertex degree. The following problem is NP-complete by [10, Theorem 11]:

BoundedDegreeTreewidth
Input a graph G with $\Delta(G) \leq 9$, an integer k
Question Is $\operatorname{tw} G \leq k$?

By deleting isolated vertices, we may suppose that every connected component of G contains at least two vertices. If G has multiple connected components, take one vertex from each component and connect them in a pathlike fashion. This increases the maximum degree potentially by one but makes the graph connected. The treewidth is invariant under this operation. Thus, we may suppose that G is connected and $\Delta(G) \leq 10$.

Given such an instance, we produce the instance (G_0, G_1, k) of WL, i.e. the decision problem in Theorem 5. Here, G_0 and G_1 are the even and odd CFI graphs of G.

Then G_0 and G_1 are k-WL indistinguishable if and only if $\operatorname{tw} G \geq k+1$. Indeed, by [13, 20] and [36, Lemma 4.4], since G is connected, if $\operatorname{tw} G \geq k+1$, then G_0 and G_1 are k-WL indistinguishable. Conversely, if $\operatorname{tw} G < k+1$, then G_0 and G_1 are distinguished by k-WL since $\hom(G, G_0) \neq \hom(G, G_1)$ by Lemma 23 and [20]. Hence, (G, k) is a YES-instance of BoundedDegreeTreewidth if and only if (G_0, G_1, k) is a NO-instance of WL. The graphs G_0 and G_1 are of size $\sum_{v \in V(G)} 2^{\deg(v)-1} \leq 2^9 n$, which is polynomial in the input. ◀

coW[1]-Hardness. The second hardness result concerns HomInd as a parametrised problem. Write $\mathcal{G}_{\leq k}$ for the class of all graphs on at most k vertices and consider the following problem:

HomIndSize
Input Graphs G and H, an integer $k \geq 1$.
Parameter k.
Question Are G and H homomorphism indistinguishable over the class $\mathcal{G}_{\leq k}$?

The problem HomIndSize fixed-parameter reduces to HomInd. To that end, consider the first-order formula $\varphi_k := \exists x_1 \dots \exists x_k \forall y \bigvee_{i=1}^{k} (y = x_i)$ for $k \in \mathbb{N}$. Then, a graph models φ_k if and only if it has at most k vertices. Furthermore, $|\varphi_k| = O(k)$. Hence, transforming the instance (G, H, k) of HomIndSize to the instance $(G, H, \varphi_k, k-1)$ of HomInd gives the desired reduction. Since $|\varphi_k| + k = O(k)$, Theorem 24 implies Theorem 4.

▶ **Theorem 24.** *HomIndSize is* coW[1]*-hard under fpt-reductions. Unless* ETH *fails, there is no algorithm for* HomIndSize *that runs in time $f(k)n^{o(k)}$ for any computable function $f \colon \mathbb{N} \to \mathbb{N}$.*

Proof. The proof is by reduction from the parametrised clique problem Clique, which is well-known to be W[1]-complete and which does not admit an $f(k)n^{o(k)}$-time algorithm for any computable function f unless ETH fails [16, Theorems 13.25, 14.21].

Let K denote the k-vertex complete graph and K_0 and K_1 the even and odd CFI graphs of K. We first observe that $K_0 \equiv_{\mathcal{G}_{\leq k} \setminus \{K\}} K_1$. Indeed, by [37, Theorem 3.13], for every graph F, $\hom(F, K_0) \neq \hom(F, K_1)$ if and only if there exists a *weak oddomorphism* $h \colon F \to K$ as defined in [37, Definition 3.9]. By definition, a weak oddomorphism is a homomorphism which is surjective on edges and vertices, i.e. for every $uv \in E(K)$ there exists $u'v' \in E(F)$ such that $h(u'v') = uv$. Hence, if $\hom(F, K_0) \neq \hom(F, K_1)$, then F has at least k vertices and $\binom{k}{2}$ edges. The only graph in $\mathcal{G}_{\leq k}$ matching this description is K.

The reduction produces given the instance (G, k) of Clique the instance $(G \times K_0, G \times K_1, k)$ of HomIndSize where K is the k-vertex clique and × denotes the categorical product (also known as tensor product) of two graphs. Producing this instance is fixed-parameter tractable. Furthermore, the parameter k is not affected by this reduction. For correctness, consider the following argument.

If $G \times K_0 \equiv_{\mathcal{G}_{\leq k}} G \times K_1$, then $\hom(K, G) = 0$. Indeed, by assumption and [33, (5.30)], $\hom(K, G) \hom(K, K_0) = \hom(K, G \times K_0) = \hom(K, G \times K_1) = \hom(K, G) \hom(K, K_1)$. However, $\hom(K, K_0) \neq \hom(K, K_1)$ by Lemma 23, and thus $\hom(K, G) = 0$.

Conversely, it holds that $K_0 \equiv_{\mathcal{G}_{\leq k} \setminus \{K\}} K_1$ and hence also $G \times K_0 \equiv_{\mathcal{G}_{\leq k} \setminus \{K\}} G \times K_1$ by the initial observation and [33, (5.30)]. Since $\hom(K, G) = 0$, also $G \times K_0 \equiv_{\mathcal{G}_{\leq k}} G \times K_1$. ◀

9 A Trichotomy for Homomorphism Indistinguishability?

Theorem 1, our central result, asserts that deciding homomorphism indistinguishability is tractable over every recognisable graph class of bounded treewidth. In particular, Theorem 1 shows that $\textsc{HomInd}(\mathcal{F})$ is tractable for every minor-closed graph class of bounded treewidth. Notably, this result does not rely on reformulations of homomorphism indistinguishability relations in terms of logic etc. but operates with the homomorphism counts themselves.

A reasonable next step is to combine Theorem 1 with a hardness result. To that end, we propose the following working hypothesis:

▶ **Conjecture 25.** *Let $\mathcal{F}$ be a minor-closed graph class.*

1. *If $\mathcal{F}$ is the class of all graphs, then $\textsc{HomInd}(\mathcal{F})$ is graph isomorphism.*
2. *If $\mathcal{F}$ has bounded treewidth, then $\textsc{HomInd}(\mathcal{F})$ is in polynomial time.*
3. *If $\mathcal{F}$ is proper and has unbounded treewidth, then $\textsc{HomInd}(\mathcal{F})$ is undecidable.*

The first assertion is implied by [32]. The second assertions amounts to derandomising Theorem 1 and is predicted by the complexity-theoretic hypothesis $\mathrm{P} = \mathrm{BPP}$. The third assertion is wide open: The only minor-closed graph class $\mathcal{F}$ for which $\textsc{HomInd}(\mathcal{F})$ is known to be undecidable, is the class $\mathcal{P}$ of planar graphs, as shown by Mančinska and Roberson [34]. Conjecture 25 is inspired by this example and a result from graph minor theory [39] which asserts that every minor-closed graph class is either of bounded treewidth or contains all planar graphs. Intuitively, $\textsc{HomInd}(\mathcal{P})$ is undecidable since the problem amounts to solving an infinite-dimensional system of equations. Roughly speaking, the dimension corresponds to the number of labels needed to generate all planar graphs under operations like series composition. Theorem 1 makes the other direction of this vague argument precise: We show that if the number of labels is bounded (e.g. the graph class has bounded treewidth), then considering finite-dimensional spaces suffices, rendering the problem tractable. That treewidth might be the right parameter in Conjecture 25 is also suggested by the complexity dichotomy for counting homomorphisms [17].

Conjecture 25 implies a weak version of Roberson's conjecture [37, Conjecture 5] asserting that $\equiv_{\mathcal{F}}$ *is not the isomorphism relation* $\cong$ *for every proper minor-closed graph class* $\mathcal{F}$. Towards Conjecture 25, one could devise reductions between $\textsc{HomInd}(\mathcal{F}_1)$ and $\textsc{HomInd}(\mathcal{F}_2)$ for distinct minor-closed graph classes $\mathcal{F}_1$ and $\mathcal{F}_2$. We are not aware of any such reduction.

Another pathway to Conjecture 25 is suggested by Theorems 13 and 17: Instead of proving hardness of the problem $\textsc{HomInd}(\mathcal{F})$, one may attempt to give lower bounds on any function[2] $f\colon \mathbb{N} \to \mathbb{N}$ such that $G \equiv_{\mathcal{F}} H$ if and only if $G \equiv_{\mathcal{F}_{\leq f(n)}} H$ for all graphs G and H on at most n vertices. This problem is purely combinatorial and avoids the intricacies of computation. Theorems 13 and 17 give such functions for every recognisable graph class of bounded treewidth. By [32], for the class $\mathcal{G}$ of all graphs, f can be taken to be the identity $n \mapsto n$. By [34], there exists no such function for the class $\mathcal{P}$ of all planar graphs which is computable. Conjecture 25 implies that no such function is computable for any minor-closed graph class of unbounded treewidth.

▶ **Question 26.** *For a graph class $\mathcal{F}$, what is the least function $f\colon \mathbb{N} \to \mathbb{N}$ such that*

$$G \equiv_{\mathcal{F}} H \iff G \equiv_{\mathcal{F}_{\leq f(n)}} H$$

for all graphs G and H on at most n vertices?

[2] Such a function always exists since there are only finitely many equivalences classes of $\equiv_{\mathcal{F}}$ on graphs on most n vertices, each pair of which is distinguished by homomorphism counts from a single graph $F \in \mathcal{F}$.

References

1 Samson Abramsky, Tomáš Jakl, and Thomas Paine. Discrete Density Comonads and Graph Parameters. In Helle Hvid Hansen and Fabio Zanasi, editors, *Coalgebraic Methods in Computer Science*, pages 23–44, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-10736-8_2.

2 Albert Atserias and Joanna Fijalkow. Definable Ellipsoid Method, Sums-of-Squares Proofs, and the Graph Isomorphism Problem. *SIAM Journal on Computing*, 52(5):1193–1229, 2023. doi:10.1137/20M1338435.

3 Albert Atserias, Phokion G. Kolaitis, and Wei-Lin Wu. On the Expressive Power of Homomorphism Counts. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13, 2021. doi:10.1109/LICS52264.2021.9470543.

4 Albert Atserias and Elitza Maneva. Sherali-Adams Relaxations and Indistinguishability in Counting Logics. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 367–379, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2090236.2090265.

5 Albert Atserias, Laura Mančinska, David E. Roberson, Robert Šámal, Simone Severini, and Antonios Varvitsiotis. Quantum and non-signalling graph isomorphisms. *J. Comb. Theory, Ser. B*, 136:289–328, 2019. doi:10.1016/j.jctb.2018.11.002.

6 László Babai. Graph Isomorphism in Quasipolynomial Time [Extended Abstract]. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 684–697, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2897518.2897542.

7 Christoph Berkholz. Lower Bounds for Existential Pebble Games and k-Consistency Tests. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pages 25–34, Dubrovnik, Croatia, June 2012. IEEE. doi:10.1109/LICS.2012.14.

8 Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. doi:10.1007/s00224-016-9686-0.

9 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1):1–45, December 1998. doi:10.1016/S0304-3975(97)00228-4.

10 Hans L. Bodlaender and Dimitrios M. Thilikos. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79(1-3):45–61, November 1997. doi:10.1016/S0166-218X(97)00031-0.

11 Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 407–416, New York NY USA, July 2016. ACM. doi:10.1145/2933575.2934508.

12 Jan Böker, Yijia Chen, Martin Grohe, and Gaurav Rattan. The Complexity of Homomorphism Indistinguishability. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2019.54.

13 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, December 1992. doi:10.1007/BF01305232.

14 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, March 1990. doi:10.1016/0890-5401(90)90043-H.

15 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach.* Cambridge University Press, USA, 1st edition, 2012.

16 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-21275-3.

17 Víctor Dalmau and Peter Jonsson. The Complexity of Counting Homomorphisms Seen from the Other Side. *Theoretical Computer Science*, 329(1):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.

18 Anuj Dawar, Tomáš Jakl, and Luca Reggio. Lovász-Type Theorems and Game Comonads. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470609.

19 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász Meets Weisfeiler and Leman. *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 40:1–40:14, 2018. doi:10.4230/LIPICS.ICALP.2018.40.

20 Zdeněk Dvořák. On recognizing graphs by numbers of homomorphisms. *Journal of Graph Theory*, 64(4):330–342, August 2010. doi:10.1002/jgt.20461.

21 John Faben and Mark Jerrum. The Complexity of Parity Graph Homomorphism: An Initial Investigation. *Theory of Computing*, 11(2):35–57, 2015. doi:10.4086/toc.2015.v011a002.

22 Eva Fluck, Tim Seppelt, and Gian Luca Spitzer. Going Deep and Going Wide: Counting Logic and Homomorphism Indistinguishability over Graphs of Bounded Treedepth and Treewidth. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.27.

23 Martin Grohe. Equivalence in Finite-Variable Logics is Complete for Polynomial Time. *Combinatorica*, 19(4):507–532, October 1999. doi:10.1007/s004939970004.

24 Martin Grohe. Counting Bounded Tree Depth Homomorphisms. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 507–520, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394739.

25 Martin Grohe. The Logic of Graph Neural Networks. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–17. IEEE, 2021. doi:10.1109/LICS52264.2021.9470677.

26 Martin Grohe, Moritz Lichter, Daniel Neuen, and Pascal Schweitzer. Compressing CFI Graphs and Lower Bounds for the Weisfeiler–Leman Refinements. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 798–809. IEEE, 2023. doi:10.1109/FOCS57990.2023.00052.

27 Martin Grohe and Martin Otto. Pebble Games and Linear Equations. *The Journal of Symbolic Logic*, 80(3):797–844, 2015. doi:10.1017/jsl.2015.28.

28 Martin Grohe, Gaurav Rattan, and Tim Seppelt. Homomorphism Tensors and Linear Equations. In Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 70:1–70:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2022.70.

29 Neil Immerman and Eric Lander. Describing Graphs: A First-Order Approach to Graph Canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

30 Moritz Lichter, Benedikt Pago, and Tim Seppelt. Limitations of Game Comonads for Invertible-Map Equivalence via Homomorphism Indistinguishability. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic (CSL 2024)*, volume 288 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2024.36.

31 Moritz Lichter, Simon Raßmann, and Pascal Schweitzer. Computational complexity of the Weisfeiler–Leman dimension. *CoRR*, abs/2402.11531, 2024. `doi:10.48550/arXiv.2402.11531`.

32 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3):321–328, September 1967. `doi:10.1007/BF02280291`.

33 László Lovász. *Large networks and graph limits*. Number volume 60 in American Mathematical Society colloquium publications. American Mathematical Society, Providence, Rhode Island, 2012.

34 Laura Mančinska and David E. Roberson. Quantum isomorphism is equivalent to equality of homomorphism counts from planar graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 661–672, 2020. `doi:10.1109/FOCS46700.2020.00067`.

35 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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4602–4609, July 2019. `doi:10.1609/aaai.v33i01.33014602`.

36 Daniel Neuen. Homomorphism-Distinguishing Closedness for Graphs of Bounded Tree-Width. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:12, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.STACS.2024.53`.

37 David E. Roberson. Oddomorphisms and homomorphism indistinguishability over graphs of bounded degree, June 2022. URL: `http://arxiv.org/abs/2206.10321`.

38 David E. Roberson and Tim Seppelt. Lasserre Hierarchy for Graph Isomorphism and Homomorphism Indistinguishability. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 101:1–101:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ICALP.2023.101`.

39 Neil Robertson and P.D Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. `doi:10.1016/0095-8956(86)90030-4`.

40 Tim Seppelt. Logical Equivalences, Homomorphism Indistinguishability, and Forbidden Minors. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*, volume 272 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 82:1–82:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.MFCS.2023.82`.

41 Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2018. URL: `https://openreview.net/forum?id=ryGs6iA5Km`.

Leakage-Resilient Hardness Equivalence to Logspace Derandomization

Yakov Shalunov[1]

University of Chicago, IL, USA

Abstract

Efficient derandomization has long been a goal in complexity theory, and a major recent result by Yanyi Liu and Rafael Pass identifies a new class of hardness assumption under which it is possible to perform time-bounded derandomization efficiently: that of "leakage-resilient hardness." They identify a specific form of this assumption which is *equivalent* to $\mathsf{prP} = \mathsf{prBPP}$.

In this paper, we pursue an equivalence to derandomization of $\mathsf{prBP}{\cdot}\mathsf{L}$ (logspace promise problems with two-way randomness) through techniques analogous to Liu and Pass.

We are able to obtain an equivalence between a similar "leakage-resilient hardness" assumption and a slightly stronger statement than derandomization of $\mathsf{prBP}{\cdot}\mathsf{L}$, that of finding "non-no" instances of "promise search problems."

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization; Theory of computation → Complexity classes

Keywords and phrases Derandomization, logspace computation, leakage-resilient hardness, psuedorandom generators

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.83

Related Version *Full Version*: `https://arxiv.org/abs/2312.14023` [12]

Funding Funded by Chris Umans' Simons Investigator Grant and Dr. Arjun Bansal and Ms. Ria Langheim's donation through the California Institute of Technology's "Summer Undergraduate Research Fellowship" (SURF) program.

Acknowledgements Most of all, I'd like to thank Professor Chris Umans at Caltech for providing guidance and suggestions throughout this project, helping me find ways forward when I was stuck, and providing references to relevant resources. I'm also very grateful to Winter Pearson, member of the undergraduate class of 2024 at Caltech, for helping with copy-editing and proofreading.

1 Introduction

In a time-bounded setting, pseudorandom generators have historically been used to show that certain circuit-complexity lower bounds imply that $\mathsf{P} = \mathsf{BPP}$ (in particular, problems in E which require exponential size circuits [5]). Another key result showed that $\mathsf{P} = \mathsf{BPP}$ (and specifically, a deterministic polynomial-time algorithm for polynomial identity testing) implies that either NEXP requires super-polynomial boolean circuits or computing permanents requires super-polynomial arithmetic circuits [6]. However, historically the lower bounds known to imply $\mathsf{P} = \mathsf{BPP}$ and those implied by $\mathsf{P} = \mathsf{BPP}$ have not matched. Seeing these results, one might hope to find a hardness assumption H such that

"$\mathsf{P} = \mathsf{BPP}$ *if and only if there exists a problem satisfying* H."

A recent exciting paper, "Leakage-Resilient Hardness v.s. Randomness" [8] by Liu and Pass, has done exactly this, building on prior work by Chen and Tell [2]; Nisan and Wigderson [9]; Impagliazzo and Wigderson [5]; Sudan, Trevisan, and Vadhan [13]; and

[1] Research conducted as a student at the California Institute of Technology.

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 83; pp. 83:1–83:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Goldreich [4]. Liu and Pass found an alternate form of hardness assumption which they were able to formulate as equivalent to prP = prBPP using a cryptographic notion of "leakage-resilient hard functions" [11] – i.e., hard functions which are still uniformly hard even if you have some "leaked" information about the output (formalized as a "leakage function" of bounded output length taking the hard function's output as an input).

In the space-bounded derandomization setting, there's been an enormous effort to prove unconditionally that L = BPL, and while it's believed that there's no obstruction to being able to do so, this goal continues to remain elusive. However, it is also known that it is possible to get a conditional derandomization in logarithmic space with PRG machinery similar to the time-bounded randomization setting [7].

1.1 Overview

In the vein of conditional derandomization, this paper constructs a result similar to Liu and Pass for logarithmic space-bounded computation. We obtain an equivalence utilizing a space-bounded analog to the time-bounded assumption in Liu and Pass's paper.

Liu and Pass's ideas do not carry directly to the space-bounded setting because certain manipulations required by the proof cannot be carried out in logspace. In particular:

Worst-case vs. average-case hardness. A key step in the Liu and Pass proof that "hardness implies derandomization" uses error-correcting codes to convert worst-case hardness to average-case hardness. In the space-bounded setting, we do not have access to error-correcting codes capable of list-decoding the available $\frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right)$ fraction of correct bits. As a consequence, our version of leakage-resilient hardness is average case instead of worst case. Using a stronger hardness assumption then required additional work in the converse direction.

Average-case hardness. When analyzing the constructed Nisan-Wigderson generator in the standard way, Liu and Pass use rejection sampling to find a prefix fixing for Yao's distingiusher-to-predictor transformation with a sublinear number of trials. However, in the space-bounded setting, we cannot write down a single fixing (since the size of the fixing is superlogarithmic), so our hardness assumption must be strengthened until even a linearly small chance of having the correct fixing contradicts the hardness assumption. This version then allows us to sample once and write it directly to the output tape without verification.

Search problem reduction. When Liu and Pass prove that prP = prBPP implies their hardness assumption, they rely on a prior result by Goldreich to reduce the task of finding a non-no solution for a prBPP search problem to a polynomial number of prBPP decision problems [4], which can then be solved deterministically by assumption. In the space-bounded setting, we've found no equivalent transformation and, as a result, our main theorem works directly with the non-standard derandomization of search problems (which in polynomial time is equivalent to prP = prBPP). Specifically, we use a "one-sided search" which, given a "promise search problem," finds a solution which is not a no-instance (these terms are more precisely defined in the following section).

Nevertheless, we are able to obtain:

▶ **Theorem 1.1** (Main Theorem (informal)). *There is a logspace-computable function which is leakage-resilient hard if and only if it is possible, in logspace, to deterministically solve a one-sided search for any* prBP·L *search problem.*

This shows a meaningful equivalence between a hardness assumption and a derandomization of a search version of prBP·L. Notably, the derandomization necessary follows from the existence of a standard pseudorandom generator construction powerful enough to derandomize prBP·L.

Shortly after the original release of this paper as a preprint, Pass and Renard released their own, independent result "Characterizing the Power of (Persistent) Randomness in Log-space" [10] which presents a theorem very similar to our own – their result is a full derandomization of prBP·L search problems under a roughly equivalent condition. Their techniques in both directions are analogous to ours and their result is fundamentally similar.

In particular, despite presenting their result differently, Pass and Renard's work demonstrates the same one-sided search as ours. While their derandomization direction (claims 5.3, 4.12, 4.9, and 4.10 in [10]) relies on a non-promise version of BP·L search problems (where non-no instances are yes instances), their hardness direction relies on promise search like ours. (Notably, Goldreich's reduction in polynomial time also produces non-no instances rather than yes instances, suggesting this is a more fundamental limit of this formulation of search problems.) Unfortunately, this produces a mismatch that makes their results not quite correct-as-written.

Pass and Renard's paper further contributes a "partial derandomization" analogous to Liu and Pass's low-end regime (i.e. for positive γ, containment of searchBP·L in $\mathsf{searchL}^{1+\gamma}$ is equivalent to existence of $\mathcal{O}(\log^{1+\gamma} n)$ computable hard f). Though we believe it contains the same mismatch of directions, after some simple reformatting into a one-sided derandomization, it will be an elegant extension to the equivalence.

We believe our proof of the key equivalence is more direct, briefer, and clearer as it, in particular, achieves equivalence without the use of Doron and Tell's [3] recent results in logspace PRGs (with no significant increase in complexity) by using a better suited version of leakage resilient hardness. Their result does, however, show that our versions of leakage resilient hardness are equivalent to each other.

Because our result follows a similar path to Liu and Pass's, in Section 2 we provide an accessible summary and informal description of the key ideas in their result.

In Section 3, we provide a formal statement of our result and the definitions necessary for it.

In Sections 4 and 5, we prove the forward and reverse directions of the claim respectively, and in Section 6 we discuss potential further work.

2 Informal description of Liu and Pass's polynomial-time equivalence result

In this section, we provide a high-level summary of Liu and Pass's proof, which also serves as an outline for our own work.

Since it is central to the proof, we start with the definition of "leakage-resilient hardness," which is a uniform hardness assumption coined by Rivest and Shamir in 1985 [11] and modified by Liu and Pass based on Chen and Tell's [2] work with almost-all input hardness.

In the proof, leakage is used to uniformly substitute for non-uniform hard-coded bits in the proof of correctness of the Nisan-Wigderson generator, which we expand on later in the section.

▶ **Definition 2.1** (Almost-all input leakage-resilient hardness [8][11][2]). *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a function. We say that f is* almost-all-input (T,ℓ)-leakage-resilient hard *if for all T-time probabilistic algorithms* (leak, A) *satisfying* $|\mathsf{leak}(x, f(x))| \leqslant \ell(|x|)$*, for all sufficiently long strings x,* $A(x, \mathsf{leak}(x, f(x))) \neq f(x)$ *with probability at least $\frac{2}{3}$.*

To illustrate this property, consider when f is the classical example of a (potentially) hard function used in cryptography: prime factorization of a product, x, of two primes. Given $f(x)$ (a correct factorization of x), a leakage function $\mathsf{leak}(x, f(x))$ which picks the smaller prime number from $f(x)$ and leaks it would make prime factorization "easy" as it would allow the attacker A to simply be a division algorithm. Since both division and identifying the smaller factor can be done in polynomial (say, n^k) time, and the smaller factor must be at most $n/2$ bits, prime factorization is *not* $(n^k, n/2)$-leakage-resilient hard – even if, without leakage, it requires superpolynomial time.

The simplest expression of Liu and Pass's result as relevant to ours is the following:

▶ **Theorem 2.2** (Liu and Pass [8]). *There exists a constant c such that for all $\varepsilon \in (0, 1)$, the following are equivalent:*

- *There exists a function $f : \{0,1\}^n \to \{0,1\}^n$, computable in deterministic polynomial time, which is almost-all input (n^c, n^ε)-leakage-resilient hard; and*
- $\mathsf{prP} = \mathsf{prBPP}$.

2.1 Forward direction (Liu and Pass)

Liu and Pass's forward direction – hardness assumption to $\mathsf{prP} = \mathsf{prBPP}$ – adapts the original proof that the Nisan-Wigderson pseudorandom generator (NW PRG) yields derandomization, with a few key differences. The generator they construct (invented by Goldreich [4]) is "targeted," meaning both it and the distinguishers it beats get access to a "target" string as an input in addition to the seed. When performing derandomization using the generator, the appropriately padded input becomes the target. (The utility of this is explained at the end of this subsection.)

Their construction of a NW-like PRG uses the error-correcting encoded output of the leakage-resilient hard function f on the "target" x instead of the standard hard-coded truth table of a non-uniformly hard function. (In our proof, the error-correcting encoding step is replaced with average-case hardness.)

Recall that in the standard proof of correctness for a Nisan-Wigderson pseudorandom generator, when converting a distinguisher to a predictor, there is a step where a prefix is fixed and a small table of a subset of outputs from the hard truth-table is hard-coded. The key difference here is that, in order to obtain a contradiction, instead of hard-coding this prefix and table non-uniformly, it is instead computed by the leak function, which has access to the hard truth-table (since it is efficiently computed from the output of f, which leak has access to). The prefixes are found by rejection sampling to pick a "good one." (In our proof, we select one randomly.) This output can be made polynomially small compared to n to satisfy the length bound ℓ on leak.

The final step of the proof is to argue that if there is a problem in prBPP that cannot be derandomized by this PRG, we can turn this into a distinguisher by fixing the problem. Observe, however, that for a contradiction, we need derandomization to fail in infinitely many cases (since the hardness assumption is *almost*-all input). Furthermore, we are only fooling *uniform* distinguishers. This is where the targeting of the PRG is necessary.

By using (a padded version of) the input to the decision problem, x, as the target, we ensure that we can uniformly "find" the infinitely many values of x where the distinguisher succeeds. If the PRG were untargeted, we would have no way to uniformly identify which value of x the PRG fails on for a given length, so failing to derandomize some decision problem would not guarantee a distinguisher.

(In our proof, we additionally perform a one-sided search derandomization. The search problem's search algorithm (the "finder") is run a with pseudorandom string generated for each possible seed until one is found which the verifier accepts. We show that such a seed must exist and that this process produces the desired derandomization. In the polynomial time regime, this one-sided search derandomization follows from decision derandomization directly without further access to a PRG and so happens on the converse side.)

2.2 Backward direction (Liu and Pass)

The key idea of the backward direction – prP = prBPP to hardness assumption – is that, if we were able to assign a "random" string to each input, we could build such an f since leaking some information about that random string would not help you reconstruct the rest of the output.

Liu and Pass make use of a result by Goldreich [4] which states that finding non-no solutions to a prBPP search problem can be reduced to a polynomial number of prBPP decision problems. Liu and Pass then formulate finding a "good output" $f(x)$ for any given input x as a prBPP search problem (defined below). They then apply Goldreich's result and then the derandomization assumption to create a hard function whose output is deterministic but behaves sufficiently like a "random string" would.

In Goldreich's formulation, a prBPP search problem is defined as follows:

▶ **Definition 2.3** (prBPP search problems [4])**.** *$R_{\text{YES}}, R_{\text{NO}} \subseteq \{0,1\}^* \times \{0,1\}^*$ where $R_{\text{YES}} \cap R_{\text{NO}} = \varnothing$ represent a* prBPP *search problem if both of the following hold:*

- *R_{YES} and R_{NO} represent a* prBPP *decision problem. I.e., there is a PPT machine V (the "verifier") such that*

$$\forall (x,y) \in R_{\text{YES}},\ \mathbb{P}[V(x,y)=1] \geqslant \frac{2}{3} \quad \textit{and} \quad \forall (x,y) \in R_{\text{NO}},\ \mathbb{P}[V(x,y)=0] \geqslant \frac{2}{3}$$

- *There exists a PPT machine F (the "finder") such that for all x, if $\exists y, (x,y) \in R_{\text{YES}}$, then*

$$\mathbb{P}[(x,F(x)) \in R_{\text{YES}}] \geqslant \frac{2}{3}$$

Let $S_R = \{x : \exists y, (x,y) \in R_{\text{YES}}\}$ be the set of inputs for which a solution exists.

Liu and Pass formulate computing f as finding solutions to a search problem. A given pair (x,y) is a yes-instance of the problem if every pair (A, leak) of attacker and leakage functions whose program descriptions have length at most $\log n$ and which run in the desired time bound has $A(x, \mathsf{leak}(x,y)) = y$ with probability at most $1/6$. Similarly, (x,y) is a no-instance if there exists (A, leak) such that $A(x, \mathsf{leak}(x,y)) = y$ with probability at least $1/3$. (The gap between yes- and no-instance probabilities is allowed by it being a *promise* search problem and is used by the verifier.)

Only attacker pairs bounded to length $2\log n$ need to be considered because this only excludes *any particular* machine from consideration for a finite number of inputs x.

When allowed to use randomness, the finder is simple – as established, simply ignoring x and picking a random string y works with high probability. The verifier can check whether a yes-or-no instance is a yes-instance by simply iterating over all attackers and leakage functions up to length $\log n$ (truncating their run time to the required bound) and checking whether any of them compute y with probability greater than $1/4$. This runs in polynomial time since there are n machines each truncated to a polynomial time bound and each pair only needs to be run polynomially many times to be correct with high probability.

The desired hard function can then be computed by using Goldreich's reduction to reduce the search problem to a polynomial number of prBPP problems. These are then derandomized to deterministically produce a non-no instance. The y-component of the resulting non-no instance is then exactly the desired hard output, by the construction of the search problem.

(In our case, we need access to the PRG rather than merely $\mathsf{prL} = \mathsf{prBP{\cdot}L}$ to perform the one-sided search, so it is performed when proving the forward direction.)

3 Results

▶ **Theorem 3.1** (Main theorem)**.** *There exist parameters $c_1, c_2, \alpha > 0$ (in particular, $\alpha = \frac{1}{3}$ works) such that the following are equivalent:*

1. *There exists a logspace function f which is almost-all-input $(n^{c_1}, c_2 \log n, \ell, \delta)$-leakage-resilient average hard where $\ell = n^{\varepsilon}$ (for $\varepsilon = 2\alpha + \frac{\alpha^3}{5}$) and $\delta = \left(\frac{1}{2} - \frac{1}{m^2}\right)n$ (for $m = n^{\alpha^3/5}$).*
2. *For any prBP·L search problem $R_{\text{YES}}, R_{\text{NO}}$, we can create a deterministic algorithm F' such that for any $x \in S_R$, we have that $(x, F'(x)) \notin R_{\text{NO}}$. (In particular, this implies $\mathsf{prBP{\cdot}L} = \mathsf{prL}$.)*

Proof. The forward direction is theorem 4.1 and the reverse direction is theorem 5.1. ◀

3.1 Preliminaries

Note that we are working with a read-*only* random tape with two-way movement. For decision problems, this definition corresponds to the complexity class BP·L, though we work with its promise and search promise counterparts.

Our result is formulated in terms of "search problems," analogous to the prBPP search problems defined by Goldreich in [4].

▶ **Definition 3.2** (prBP·L search problems)**.** *A pair of sets $R_{\text{YES}}, R_{\text{NO}} \subseteq \{0,1\}^* \times \{0,1\}^*$ where $R_{\text{YES}} \cap R_{\text{NO}} = \varnothing$ represent a prBP·L search problem if both of the following hold*

- *R_{YES} and R_{NO} represent a prBP·L decision problem. I.e., there is a PPT logspace machine V (the "verifier") such that*

$$\forall (x,y) \in R_{\text{YES}},\ \mathbb{P}[V(x,y) = 1] \geqslant \frac{2}{3} \quad \textit{and} \quad \forall (x,y) \in R_{\text{NO}},\ \mathbb{P}[V(x,y) = 0] \geqslant \frac{2}{3}$$

- *There exists a PPT logspace machine F (the "finder") such that for all x, if $\exists y, (x,y) \in R_{\text{YES}}$, then*

$$\mathbb{P}[(x, F(x)) \in R_{\text{YES}}] \geqslant \frac{2}{3}$$

Let $S_R = \{x : \exists y, (x,y) \in R_{\text{YES}}\}$ be the set of inputs for which a solution exists.

This definition is identical to the polynomial variant with the added constraint that the machines are bounded to logarithmic space in addition to the polynomial time bound.

Our hardness assumption is a modification of leakage resilient hardness as defined in definition 2.1 of [8], which in turn builds on [11].

▶ **Definition 3.3** (Almost-all-input leakage resilient average hard)**.** *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a multi-output function. f is* almost-all-input (T, S, ℓ, d)-leakage resilient average hard *if for all S-space T-time probabilistic algorithms (A, leak) satisfying $|\mathsf{leak}(x, f(x))| \leqslant \ell(n)$, we have that for all sufficiently long strings x,*

$$\mathbb{P}[d_H(A(x, \mathsf{leak}(x, f(x))), f(x)) < d(n)] < \frac{1}{n}$$

with probability over the internal randomness of the algorithms, where d_H is Hamming distance.

Note that this is "stronger" than a direct analog to the time-bounded assumption [8] in two ways: first, and most importantly, we've added another parameter which parameterizes how "good of an approximation" is allowed, whereas in the time-bounded setting, forbidding strict equality was sufficient. Second, the allowed probability of "success" is $\frac{1}{n}$ rather than $\frac{1}{3}$.

The first difference converts the worst-case hardness to average-case hardness. It is necessary because in the time-bounded setting, we have access to much more powerful error-correcting codes which are not available in the logspace setting. In the time-bounded setting, error-correcting codes let us convert a small edge in breaking the generator to an exact computation of the hard function. Meanwhile, in the logspace setting, we're limited to just that small edge, so we bake it into the hardness assumption instead.

The second difference exists to handle issues in the distinguisher to predictor transformation. In polynomial time, the attacker functions can "try" $o(1/n)$ strings and figure out which one is the ideal choice, but because the strings are super-logarithmic in length, we cannot verify them in logspace. Thus, the hardness assumption is strengthened to allow simply choosing a random one without verification.

While much of the high-level structure of the following proof is similar to that presented in section 2, we have that:

- in the "hard function implies derandomization" direction, we show the desired derandomization of search problems; and
- in the "derandomization implies hard function" direction, the search problem is constructed such that anything which is not a no-instance is a valid value for the function to take for that x. Then, instead of Goldreich's reduction, we apply the derandomization reduction directly.

4 Existence of hard function implies derandomization

▶ **Theorem 4.1** (Existence implies derandomization). *The forward direction of the main theorem (3.1). I.e., (1) implies (2).*

Proof. Apply lemma 4.4 with $C = 3$ to get the necessary PRG, then apply lemma 4.7 to perform the desired derandomization with the PRG. ◀

The following are fairly standard definitions of a distinguisher and pseudorandom generator with the caveat of the additional "target" string. For brevity, write $D_x^m(\gamma)$ for $D(1^m, x, \gamma)$ and similarly for G.

▶ **Definition 4.2** (Targeted distinguisher). *Given a function $G : \{1\}^m \times \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, a targeted distinguisher for G with advantage $\beta > 0$ is a machine $D : \{1\}^m \times \{0,1\}^n \times \{0,1\}^m$ such that for all sufficiently large m and for all $x \in \{0,1\}^n$, we have that*

$$\left|\mathbb{P}_{s\sim\mathcal{U}_d}\left[D_x^m(G_x^m(s)) = 1\right] - \mathbb{P}_{\gamma\sim\mathcal{U}_m}\left[D_x^m(\gamma) = 1\right]\right| \geqslant \beta \tag{1}$$

where $\mathcal{U}_k$ is the uniform distribution on $\{0,1\}^k$.

Throughout, we will refer to targeted distinguishers as "distinguishers."

This definition becomes useful in the context of the following (modified from [4, 8]):

▶ **Definition 4.3** (Uniform targeted pseudorandom generator). *For parameters S, n, and d dependent on output length m, an S-secure uniform (n, d)-targeted PRG is a deterministic function*

$$G : \{1\}^m \times \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

which satisfies the property that there does not exist a targeted distinguisher with advantage β running in space S for any $\beta > 0$.

Throughout the rest of this article, "PRG" and "pseudorandom generator" refer to a uniform targeted PRG. n is the "target length" of the generator and d is the "seed length."

We show that our hardness assumption implies an $\mathcal{O}(\log m)$-secure $(\mathsf{poly}(m), \mathcal{O}(\log m))$ PRG and then that the existence of an such a PRG implies the desired derandomization.

4.1 Hard function implies existence of PRG

First, we prove that a hard function with the given parameters implies the existence of a PRG.

Fix a sufficiently small value of $\alpha > 0$. $\alpha = 1/3$ works.

▶ **Lemma 4.4** (Hard function implies existence of PRG). *For any C, there exist constants c_1, c_2 such that if there exists a logspace computable function f such that f is almost-all-input $\left(n^{c_1}, c_2 \log n, n^{\varepsilon}, \left(\frac{1}{2} - \frac{1}{m^2}\right)n\right)$-leakage resilient average hard with $m = n^{\frac{\alpha^3}{5}}$ and $\varepsilon = 2\alpha + \frac{\alpha^3}{5}$, then there exists a $C \log m$-secure $(n, \frac{\log n}{\alpha})$ PRG which is logspace computable.*

As is standard for a Nisan-Wigderson generator, we will require designs for the construction.

▶ **Definition 4.5** (Combinatorial design). *A design is a collection of (potentially large) sets with a bounded pairwise intersection size. More precisely:*

For any natural numbers (d, r, s) such that $d > r > s$, a (d, r, s)-design of size m is a collection $\mathcal{I} = \{I_1, \ldots, I_m\}$ of subsets of $[d]$ such that for each $j \in [m]$, $|I_j| = r$ and for each $k \in [m]$ such that $k \neq j$, $|I_j \cap I_k| \leqslant s$.

Because we're working in logspace, we will need the following lemma:

▶ **Lemma 4.6** (Creating designs). *There is a deterministic algorithm we call $\mathsf{GenDesign}(d, \alpha)$ which produces a $(d, \alpha d, 2\alpha^2 d)$ design of size $2^{\frac{\alpha^4 d}{5}}$ in space $S_{\mathsf{GenDesign}} = \mathcal{O}(d)$.*

Proof. Proven by Klivans and Melkebeek in the appendix of [7]. ◀

Outline of proof of 4.4. As presented in the overview, the construction of the PRG is a fairly standard Nisan-Wigderson generator [9] construction, with the caveats that it is a targeted PRG and that instead of the truth table of a hard-coded non-uniformly hard function, we use the output of our f on the target string. The key change is that the leakage function then replaces the hard-coded tables in the search-to-decision reduction.

We show that this can be done in logspace and that the hard-coded tables fit in the inverse polynomial leakage bound n^{ε}. In order to compute the tables in logspace, the probability of generating "good tables" needs to be low because it's impossible to verify the tables. However, a probability high enough to contradict leakage-resilient hardness, i.e., over $\frac{1}{n}$, is achievable.

Assuming a distinguisher that defeats the PRG, the attacker function is then able to use the tables from the leakage function together with the distinguisher to perform the search-to-decision reduction with enough accuracy to violate the average-case hardness constraint. ◀

Proof of 4.4. With an appropriate choice of c_1, c_2, we can construct a $C \log m$-secure PRG for any constant $C \geqslant 0$.

We construct a targeted PRG $G : \{1\}^m \times \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, where $n = m^{\frac{5}{\alpha^3}}$ and $d = \frac{\log n}{\alpha}$.

The algorithm is similar to that in time-bounded case [8]: it is simply a Nisan Wigderson generator [9], with the hard-coded truth table replaced with one outputted by the hard function.

On input $1^m, x, s$, it proceeds as follows:

1. Compute $z = f(x)$. Define $h(i) = z_i$.
2. Compute $\mathcal{I} = \mathsf{GenDesign}(d, \alpha)$. Note that $r = d\alpha = \log n$ and $2^{\frac{\alpha^4 d}{5}} \geqslant m$ due to our choice of n and d.
3. Output

$$G(1^m, x, s) = h(s_{I_1}) \cdots h(s_{I_m})$$

Suppose there exists a distinguisher D computable in space $C \log m$ with advantage $\beta > 0$. We will use this to approximate f with enough of an advantage to violate the average hardness assumption by a standard hybrid argument (lemma 3.15 in [8], proposition 7.16 in [14], theorem 10.12 in [1]).

We can remove the absolute value from the distinguisher definition (1) by noting that there exists $b \in \{0,1\}$ such that

$$\mathbb{P}_{s \sim \mathcal{U}_d}[D_x^m(G_x^m(s)) = b] - \mathbb{P}_{\gamma \sim \mathcal{U}_m}[D_x^m(\gamma) = b] \geqslant \beta$$

since flipping b flips the sign.

Then, for every $j \in \{0, \dots, m\}$, define

$$H_j = (h(s_{I_1}), \dots, h(s_{I_j}), w_{j+1}, \dots, w_m)$$

(with h defined as in the generator) where $s \sim \mathcal{U}_d$ and each $w_k \sim U_1$ (for $j+1 \leqslant k \leqslant m$). Notice that $H_0 = \mathcal{U}_m$ and $H_m = G(1^m, x, \mathcal{U}_d)$. Therefore it follows that

$$\frac{1}{m} \sum_{j \in 1}^{m} \Big(\mathbb{P}_{y,w}[D_x^m(H_j) = b] - \mathbb{P}_{y,w}[D_x^m(H_{j-1}) = b] \Big)$$
$$= \frac{1}{m} \Big(\mathbb{P}_{y,w}[D_x^m(H_m) = b] - \mathbb{P}_{y,w}[D_x^m(\mathcal{U}_0) = b] \Big) \geqslant \frac{\beta}{m}$$

Considering j as a random variable distributed uniformly over $[m]$, we get that

$$\mathbb{E}_{j \in [m]} \Big[\mathbb{P}_{y,w}[D_x^m(H_j) = b] - \mathbb{P}_{y,w}[D_x^m(H_{j-1}) = b] \Big] \geqslant \frac{\beta}{m}$$

Since $\mathbb{P}_{y,w}[D_x^m(H_j) = b] - \mathbb{P}_{y,w}[D_x^m(H_{j-1}) = b]$ is upper bounded by 1, by an averaging argument, with probability at least $\frac{\beta}{2m}$ over the choice of $j \leftarrow [m]$, $y_{[d] \setminus I_j} \leftarrow \{0,1\}^{d-r}$, and $w_{[m] \setminus [j]} \leftarrow \{0,1\}^{m-j}$, the strings j, $y_{[d] \setminus I_j}$, and $w_{[m] \setminus [j]}$ will satisfy:

$$\mathbb{P}_{y_{I_j} \sim \mathcal{U}_r}[D_x^m(H_j) = b] - \mathbb{P}_{(y_{I_j}, w_j) \sim \mathcal{U}_{r+1}}[D(H_{j-1}) = b] \geqslant \frac{\beta}{2m} \tag{2}$$

Suppose we have a choice j, $y_{[d] \setminus I_j}$, and $w_{[m] \setminus [j]}$ satisfying the above equation. By Yao's prediction vs. indistinguishability theorem [15], it holds that

$$\mathbb{P}_{(y_{I_j}, w_j) \sim \mathcal{U}_{r+1}}\big[D_x^m(H_{j-1}) \oplus b \oplus w_j = h(y_{I_j})\big] \geqslant \frac{1}{2} + \frac{\beta}{2m}$$

There then must exist a choice of w_j such that

$$\mathbb{P}_{y_{I_j} \sim \mathcal{U}_r}\big[D_x^m(H_{j-1}) \oplus b \oplus w_j = h(y_{I_j})\big] \geqslant \frac{1}{2} + \frac{\beta}{2m} \tag{3}$$

Observe that in order to compute H_{j-1} as a function of y_{I_j}, one does not need to know the entire truth table of h. Instead, note that the overlap between I_j and I_i, $i < j$, is at most $2\alpha^2 d = 2\alpha r$. Thus, as a function of y_{I_j}, $h(y_{I_i})$ can only take $2^{2\alpha r}$ possible values. Thus, to compute all of H_{j-1}, one needs $j2^{2\alpha r} \leqslant m2^{2\alpha r}$ bits of the truth table of h.

Then, given a distinguisher D with advantage β, we can approximate f with the functions (A, leak) described below.

On input $x, z = f(x)$, leak proceeds as follows (denote $h(i) = z_i$):

1. Evaluates $\mathcal{I} = \mathsf{GenDesign}(d, \alpha)$, with d chosen the same way as the PRG.
2. Chooses a random $j \in [m]$ and a random $y_{[d]\setminus I_j}$, storing them on its work tape.
3. Writes j and $y_{[d]\setminus I_j}$ to the output tape.
4. For $0 < i < j$ and for each value of y obtained by ranging over possible values of y_{I_j} (since $y_{[d]\setminus I_j}$ is already fixed), leak writes $h(y_{I_i})$ to the output tape.
5. Writes random bits b, w_j and string $w_{[m]-[j]}$ to the output tape.

Then, on input $x, w = \mathsf{leak}(x, f(x))$, A proceeds as follows:

1. A evaluates $\mathcal{I} = \mathsf{GenDesign}(d, \alpha)$, with d chosen the same way as the PRG.
2. Defines b, j, $y_{[d]\setminus I_j}$, w_j, and $w_{[m]\setminus[j]}$ to be those read from its input tape.
3. For each $y_{I_j} = k$ in $[n]$ yielding a full string y,
   a. Computes H_{j-1}, with bit $i < j$ found by looking in position k in the ith table output by leak in step 4 and $i \geqslant j$ found by taking the appropriate bit of w, also output by leak.
   b. Computes $D_x^m(H_{j-1}) \oplus b \oplus w_j$ and writes it to the output tape.

We will show that the result is a pair which runs in the necessary time and space bounds and, with high probability, is a good enough approximation of $f(x)$ to achieve contradiction.

Space-bound. leak step 1 requires space $S_{\mathsf{GenDesign}} = \mathcal{O}(d) = O\left(\frac{\log n}{\alpha}\right)$. Since it is locally computed by future steps, this becomes $\mathcal{O}(S_{\mathsf{GenDesign}})$. The second step requires space $d + \log m$, and uses the local computation of step 1. The third step requires space $\log(\log m + d)$, since it is simply iteration and printing. The fourth step uses space $\log m + d$ again since it iterates over indices i and values of y. For each value, it simply performs a lookup into the input, which requires at most space equal to the size of the index, bounded by $d + 1$. Finally, step 5 requires space $\log m$, since it needs to keep track of the number of bits it needs to write.

Thus, the overall space used by leak is $\mathcal{O}(d) + \mathcal{O}(\log n) + \mathcal{O}(\log m) = \mathcal{O}(\log n + \log \alpha) = \mathcal{O}(\log n)$.

For A, step 1 again requires space $\mathcal{O}(S_{\mathsf{GenDesign}})$. Step 2 is simply a definition and thus requires no space. Step 3 requires $\log n$ bits of iteration overhead plus the space used by 3.(a) and 3.(b). 3.(a) is simply indexing into the input based on k and so requires space $\log n$. This is the locally recomputed by step 3.(b), so 3.(a) requires space $\mathcal{O}(\log n)$. Finally, 3.(b) requires space $C \log m$.

Thus, A and leak run in space $c_2 \log n$ for a constant c_2 dependent only on C.

Time-bound. Due to space constraints, the straightforward time complexity analysis can be found in the full version [12].

Contradiction of hardness. First, note that leak outputs at most $\log m + d - r + m + m2^{2\alpha^2 d}$ bits. The $\log m$ and $d - r$ terms are logarithmic and thus fall within any polynomial bound. The constraints on d and α, chosen such that $d\alpha = \log n$ and $2^{\frac{\alpha^4 d}{5}} = m$ tell us that for the $m2^{2\alpha^2 d}$ term, we have that

$$2\alpha^2 d = \frac{10}{\alpha^2}\log m \quad \text{so} \quad 2^{2\alpha^2 d} = m^{\frac{10}{\alpha^2}}$$

Thus, $m2^{2\alpha^2 d} = m^{\frac{10}{\alpha^2}+1}$, which is

$$m^{\frac{10}{\alpha^2}+1} = \left(n^{\frac{\alpha^3}{5}}\right)^{\frac{10}{\alpha^2}+1} = n^{2\alpha+\frac{\alpha^3}{5}}$$

Thus, if α is small enough (e.g., $\frac{1}{3}$), then for $\varepsilon = 2\alpha + \frac{\alpha^3}{5} < 1$, we have that $|\mathsf{leak}(x, f(x))| \leqslant n^{\varepsilon}$.

Observe that (2) is satisfied with probability $\frac{\beta}{2m}$ by leak's random choice of j, $y_{[d]\setminus I_j}$, and $w_{[m]\setminus[j]}$. Furthermore, with probability 1/4, leak's random choices of b and w_j are the correct choices such that (3) holds. Thus, with probability $\frac{\beta}{8m}$, we have that (3) holds for our choices.

Consider the case where it does. Note that the randomness in (3) is over the argument to h, which is the index into $f(x)$. Then, when A evaluates $D_x^m(H_{j-1}) \oplus b \oplus w_j$ for each index, we conclude that at least a $\frac{1}{2} + \frac{\beta}{2m}$ fraction of bits output by A agree with $f(x)$. We thus have that

$$\mathbb{P}\left[d_H(A(x, \mathsf{leak}(x, f(x))), f(x)) \leqslant \left(\frac{1}{2} - \frac{\beta}{2m}\right)n\right] \geqslant \frac{\beta}{8m}$$

However, note that $\frac{\beta}{8m} > \frac{1}{n}$ for sufficiently large n (since m is polynomially smaller than n) and $\frac{1}{2} - \frac{\beta}{2m} < \frac{1}{2} - \frac{1}{m^2}$ for any constant β and sufficiently large n. This contradicts the hardness assumption of f, so such a distinguisher can't exist and we must have that G is a $C\log m$-secure PRG. ◀

4.2 Derandomization of search problems using the PRG

Now, we prove that the PRG is sufficient. This formulation is similar to work by Goldreich [4] which was seen in Liu and Pass's [8] reproduction.

▶ **Lemma 4.7** (Derandomization of search problems using the PRG). *The existence of a* $3\log m$*-secure* $(\mathsf{poly}(m), \mathcal{O}(\log m))$*, logspace-computable PRG implies (2) of main theorem (3.1).*

As is typical for derandomization using a PRG, we will need the ability to pad strings. We will treat functions $\mathsf{pad}(x, k)$ and $\mathsf{unpad}(x')$ as having "negligible" complexity. The exact definition and proof of this can be seen in the "efficient padding" lemma of the full version [12], but we effectively use an expanded alphabet (encoded as pairs of bits) to add an "pad" character.

We separate our proof of 4.7 into two parts: derandomizing $\mathsf{prBP{\cdot}L}$ using a PRG and then derandomizing $\mathsf{prBP{\cdot}L}$ search problems using both $\mathsf{prBP{\cdot}L} = \mathsf{prL}$ and the PRG.

▶ **Lemma 4.8** (Derandomization of promise problems). *The existence of a* $3\log m$*-secure* $(\mathsf{poly}(m), \mathcal{O}(\log m))$*, logspace-computable PRG implies* $\mathsf{prBP{\cdot}L} = \mathsf{prL}$.

Proof sketch. This is an adaptation of the time-bounded derandomization construction as seen in Goldreich's and Liu and Pass's work. The detailed working is omitted due to space constraints, but can be found in the full version [12] to verify that there are no hiccups related to logarithmic space. (In particular, the unpadded case becomes time $\mathcal{O}(k)$ and space $\log k + \mathcal{O}(1)$, and the padding exponent becomes the maximum of the exponent on time and the multiplier on space.) ◀

Proof of 4.7. Suppose G is an $3\log m$-secure (n, d) PRG with parameters $n = m^\theta$ for some θ and $d = \mathcal{O}(\log m)$. By lemma 4.8, we have that $\mathsf{prBP{\cdot}L} = \mathsf{prL}$.

Suppose $R_{\mathrm{YES}}, R_{\mathrm{NO}}$ is a $\mathsf{prBP{\cdot}L}$ search problem with verifier $V(x, y; \omega)$ and finder $F(x; \gamma)$ where $|\gamma| \leqslant t(|x|)$. Observe that we can derandomize V to $V(x, y)$ (since it is a $\mathsf{prBP{\cdot}L}$ decision problem) such that $(x, y) \in R_{\mathrm{NO}} \implies V(x, y) = 0$ and $(x, y) \in R_{\mathrm{YES}} \implies V(x, y) = 1$.

First we will perform this derandomization for search problems where $F(x; \gamma)$ has time complexity $t(k) = \mathcal{O}(k)$ and the random procedure $M(x; \gamma) = V(x, F(x; \gamma))$ has space complexity $\log k + \mathcal{O}(1)$. Define $m = t(k)$ as the maximum amount of randomness used.

Consider the following deterministic procedure F': On input x,

1. For each seed $s \in \{0, 1\}^d$,
- Compute $b \leftarrow V(x, F(x; G^m_{x'}(s)))$, where $x' = \mathsf{pad}(x, n)$.
- If $b = 1$, output $F(x; G^m_{x'}(s))$.

2. Output the empty string.

Note that this procedure operates in logspace since iterating seeds requires space $d = \mathcal{O}(\log m) = \mathcal{O}(\log k)$ and step 3.(a) and 3.(b) are logspace compositions (a more detailed treatment can be see in the proof of 4.8).

Observe also that if this procedure terminates within the loop, it outputs a string y such that $V(x, y) = 1$ and thus such that $(x, y) \notin R_{\mathrm{NO}}$. It remains to show that this procedure terminates within the loop on all but finitely many inputs (since we can hard-code any finite number of misbehaving cases).

Suppose the contrary. Then there exist infinitely many inputs for which,

$$\forall s \in \{0, 1\}^d \quad V(x, F(x; G^m_{x'}(s))) = 0$$

Further, by definition of a finder, $\mathbb{P}_\gamma[(x, F(x; \gamma)) \in R_{\mathrm{YES}}] \geqslant \frac{2}{3}$. Thus,

$$\mathbb{P}_s[V(x, F(x; G^m_{x'}(s))) = 1] = 0 \quad \text{and} \quad \mathbb{P}_\gamma[(x, F(x; \gamma)) \in R_{\mathrm{YES}}] \geqslant \frac{2}{3}$$

This tells us that $D(1^m, x', r) = M(\mathsf{unpad}(x); r) = M(x; r)$ is a distinguisher with advantage $\beta \geqslant \frac{2}{3}$ which works for infinitely many values of x' and runs in space $2\log m$ (since it is the truncation machine composed with M; again, a more detailed treatment can be seen in the proof of 4.8). This contradicts the security of the generator.

Thus, $(x, F'(x)) \notin R_{\mathrm{NO}}$ for all but finitely many x (which can be hard-coded).

Now consider an arbitrary $\mathsf{prBP{\cdot}L}$ search problem, $(R_{\mathrm{YES}}, R_{\mathrm{NO}})$ with derandomized verifier V and finder $F(x; \gamma)$ such that the time complexity of F is $\mathcal{O}(k^a)$ and the space complexity of $V(x, F(x; \gamma))$ is $b \log k + \mathcal{O}(1)$. Again, let $c = \max(a, b)$ and define $x' = \mathsf{pad}(x, k^c)$. We create $R_{\mathrm{YES}\,\mathsf{pad}} = \{(x', y) : (x, y) \in R_{\mathrm{YES}}\}$ and $R_{\mathrm{NO}\,\mathsf{pad}} = \{(x', y) : (x, y) \in R_{\mathrm{NO}}\}$.

Then $V_{\mathsf{pad}}(x', y) = V(\mathsf{unpad}(x'), y)$, $F_{\mathsf{pad}}(x') = F(\mathsf{unpad}(x'))$ satisfy

$$M_{\mathsf{pad}}(x'; r) = V_{\mathsf{pad}}(x', F_{\mathsf{pad}}(x'; r))$$

being a machine with space complexity $\log k' + \mathcal{O}(1)$ and F_{unpad} having time complexity $\mathcal{O}(k')$. Thus, we have deterministic algorithm $F'_{\mathsf{pad}}(x')$ such that $(x', F'_{\mathsf{pad}}(x')) \notin R_{\mathrm{NO}\,\mathsf{pad}}$.

We can then define F' as the function which on input x computes $F'_{\mathsf{pad}}(\mathsf{pad}(x, k^c))$. Observe that if $(x, F'(x)) \in R_{\mathrm{NO}}$ then $(x', F'_{\mathsf{pad}}(x')) = (x', F'(x)) \in R_{\mathrm{NO}\,\mathsf{pad}}$, which contradicts F'_{pad} being a derandomized finder for $R_{\mathrm{NO}\,\mathsf{pad}}$.

Thus, we have achieved the desired partial derandomization of $\mathsf{prBP{\cdot}L}$ search problems. ◀

5 Derandomization implies existence of hard function

▶ **Theorem 5.1** (Derandomization implies existence). *Suppose that for any* $\mathsf{prBP{\cdot}L}$ *search problem* $(R_{\mathrm{YES}}, R_{\mathrm{NO}})$*, we can create a deterministic algorithm* F' *such that for any* $x \in S_R$*, we have that* $(x, F'(x)) \notin R_{\mathrm{NO}}$*.*

We will show that for any constants c_1, c_2 *there exists a logspace computable function* f *such that* f *is almost-all-input* $(T, S, \ell, d) = \left(n^{c_1}, c_2 \log n, n^{\varepsilon}, \left(\frac{1}{2} - \frac{1}{m^2}\right)n\right)$*-leakage-resilient average hard where* $\varepsilon = 2\alpha + \frac{\alpha^3}{5}$ *and* $m = n^{\frac{\alpha^3}{5}}$ *for* $\alpha = \frac{1}{3}$*.*

The "Random is hard" lemma is similar to claim 1 in section 3.1 of Liu and Pass's paper [8] but needs a much stronger claim than in the time-bounded setting, corresponding to the use of an average-case hardness assumption.

5.1 Random is hard

▶ **Lemma 5.2** (Random is hard). *For* any *probabilistic algorithms* (A, leak) *and for all* $n \in \mathbb{N}$*,* $x \in \{0,1\}^n$*, it holds that*

$$\mathbb{P}[|\mathsf{leak}(x,r)| \leqslant \ell \wedge d_H(A(x, \mathsf{leak}(x,r)), r) < d] \geqslant \frac{1}{2n}$$

with probability at most $n \cdot 2^{n\cdot(H(d/n)-1)+\ell+\mathcal{O}(1)}$ *(where* $H(p)$ *is the binary entropy function* $H(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$*) over* $r \sim \mathcal{U}_n$*.*

Proof. Consider any $n \in \mathbb{N}$, $x \in \{0,1\}^n$, and any probabilistic algorithm A. We will show that for any *deterministic* function leak' that outputs at most ℓ bits,

$$\mathbb{P}_{r\sim\mathcal{U}_n}\left[\mathbb{P}\big[d_H(A(x, \mathsf{leak}'(x,r)), r) < d\big] \geqslant \frac{1}{2n}\right] \leqslant n \cdot 2^{n\cdot(H(d/n)-1)+\ell+\mathcal{O}(1)} \tag{4}$$

The claim for any probabilistic algorithm leak follows immediately by letting leak' be leak with the best random tape fixed. To show (4), consider the set of "bad" r's defined as

$$B = \left\{r \in \{0,1\}^n : \exists w \in \{0,1\}^{\ell} \text{ s.t. } \mathbb{P}[d_H(A(x, \mathsf{leak}(x,r)), r) < d] \geqslant \frac{1}{2n}\right\}$$

For any r, if there exists a leak' such that $|\mathsf{leak}'(x,r)| \leqslant \ell$ and $\mathbb{P}\big[d_H(A(x, \mathsf{leak}'(x,r)), r) < d\big] \geqslant \frac{1}{2n}$, then $r \in B$. Thus, the probability that $A(x, \mathsf{leak}(x,r))$ successfully approximates r is bounded by the probability that $r \in B$.

We now bound B. Observe that for a given choice of $w \in \{0,1\}^{\leqslant \ell}$, there exists $2n$ Hamming balls (which, in the worst case, do not overlap) of radius d contained in B, since there are at most $2n$ Hamming balls which $g(x,w)$ can occupy with probability $\geqslant \frac{1}{2n}$.

Finally, note that a Hamming ball of radius d on strings of length n has volume

$$2^{nH(d/n)-\frac{1}{2}\log n+\mathcal{O}(1)} \leqslant 2^{nH(d/n)+\mathcal{O}(1)}. \quad \text{Thus,} \quad |B| \leqslant 2^{\ell+1} \cdot 2n \cdot 2^{nH(d/n)+\mathcal{O}(1)}$$

We can roll the 2 in $2n$ into the $\mathcal{O}(1)$ in the final exponent, along with the $+1$ in $\ell + 1$.

Thus, the probability that a randomly chosen r satisfies the condition in (4) is at most

$$\frac{|B|}{2^n} \leqslant n \cdot 2^{n(H(d/n)-1)+\ell+\mathcal{O}(1)}$$

◀

5.2 Construction of hard function

Proof of theorem 5.1. We can construct a prBP·L search problem to represent computing our desired hard problem. This search problem is similar to the corresponding search problem from [8] with the changes corresponding directly to the differences between the time-bounded worst-case version of leakage-resilient hardness and the space-bounded average-case version presented in this work.

The search problem. Note that, since we consider only efficiently and uniformly computable (A, leak), a given pair has some constant length. We can thus consider only pairs (A, leak) with descriptions of length at most $\log n$. For any given attacker pair, then, we will only skip it under our search problem in a finite number of cases, which is covered by the almost-all-input hardness.

Let R_{YES} be a binary relation such that $(x, r) \in R_{\mathrm{YES}}$ if both of the following are true:

1. $|x| = |r|$.
2. For all probabilistic (A, leak) such that $|A|, |\mathsf{leak}| \leqslant \log n$ (where $|A|$ (resp. leak) refers to the length of the description of the program A in some arbitrary encoding scheme), it holds that

$$\mathbb{P}\big[|\mathsf{leak}'(x, r)| \leqslant \ell \wedge d_H(A'(x, \mathsf{leak}'(x, r)), r) < d\big] < \frac{1}{2n} \tag{5}$$

where A' and leak' are time-space-truncated versions of A and leak which are only executed until they run for n^{c_1} steps or try to use more than $c_2 \log n$ space (where c_1, c_2 are from theorem 5.1).

Let R_{NO} be a binary relation such that $(x, r) \in R_{\mathrm{NO}}$ if $|x| \neq |r|$ or for *at least one pair* of (A, leak) (within $\log n$ size bound), (5) doesn't hold with a $\frac{1}{n}$ lower bound instead of an $\frac{1}{2n}$ upper bound.

We then show this is a prBP·L search problem by finding a verifier and finder.

Verifier. On input (x, r), the verifier immediately rejects if $|x| \neq |r|$. Otherwise, it enumerates all probabilistic machines (A, leak) such that $|A|, |\mathsf{leak}| \leqslant \log n$. For each one, V estimates the value

$$p_{A,\mathsf{leak}} = \mathbb{P}\big[|\mathsf{leak}'(x, r)| \leqslant \ell \wedge d_H(A'(x, \mathsf{leak}'(x, r)), r) < d\big]$$

by simulating $A'(x, \mathsf{leak}'(x, r))$ polynomially many times and recording whether the Hamming distance is at most d. If during this process, for any (A, leak), we have that the sample estimate $\hat{p}_{A,\mathsf{leak}}$ exceeds $\frac{3}{4n}$, V outputs 0 and terminates. Otherwise, the value $p_{A,\mathsf{leak}}$ is discarded for this attacker pair and estimated for the next attacker pair. If we iterate every attacker pair without terminating, we output 1 and terminate. By the Chernoff bound and the union bound, V will accept with high probability if $(x, r) \in R_{\mathrm{YES}}$ and reject with high probability if $(x, r) \in R_{\mathrm{NO}}$.

This step is exactly where two-way randomness becomes necessary. We can compute $A'(x, \mathsf{leak}'(x, r))$ in logspace since with two-way randomness of BP·L, we can compose functions. The rest is logspace, since computing Hamming distance can be done in a read-once manner by tallying number of agreeing bits, which requires $\log n$ bits, and keeping a tally of successes vs. total count represents the success probability in $\log \mathsf{num\_trials}$ space. Since only a polynomial number of trials is necessary, this is all logspace.

We can compute it in polynomial time, since we only need polynomially many samples and each of A' and leak' is time truncated to a polynomial time-bound.

Finder. On input x, the finder outputs a random string of the same length. Observe that for any fixed $x \in \{0,1\}^n$, by the "random is hard" lemma (5.2), and a union bound over choices of (A, leak), we conclude that $F(x)$ outputs an invalid witness with probability at most

$$n^2 \cdot n \cdot 2^{n(H(d/n)-1)+\ell+\mathcal{O}(1)}$$

When $\ell = n^{\varepsilon} = n^{2\alpha+\frac{\alpha^3}{5}}$ and $\frac{d}{n} = \frac{1}{2} - \frac{1}{m^2}$ with $m = n^{\frac{\alpha^3}{5}}$ and $\alpha = \frac{1}{3}$, this converges to 0, and thus satisfies the search problem requirement of error probability less than $\frac{1}{3}$ (except for perhaps some finite number of small samples where answers can be hard-coded).

Computing the hard function. By the derandomization assumption, there exists a deterministic, logspace function F' such that $(x, F'(x)) \notin R_{\mathrm{NO}}$.

Note that $F'(x) \notin R_{\mathrm{NO}}$, so $|F'(x)| = |x|$ and for all machines (A, leak) which run within the time and space bounds, (5) (with $\frac{1}{2n}$ replaced with $\frac{1}{n}$) holds for all but the finitely many strings too short to include the descriptions of A, leak.

Therefore, by construction, $f = F'$ is a logspace computable $(n^{c_1}, c_2 \log n, \ell, d)$-leakage-resilient average hard function as desired. ◀

6 Discussion

We suspect it is possible to reduce the derandomization part of our assumption to $\mathsf{prBP{\cdot}L} = \mathsf{prL}$, such that we do not explicitly need the separate partial derandomization of $\mathsf{prBP{\cdot}L}$ search problems. However, we were not able to identify such a reduction within the duration of this research.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: a Modern Approach.* Cambridge University Press, 2009.

2 Lijie Chen and Roei Tell. Hardness vs randomness, revised: Uniform, non-black-box, and instance-wise. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 125–136, 2022. doi:10.1109/FOCS52979.2021.00021.

3 Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In *Proceedings of the Conference on Proceedings of the 38th Computational Complexity Conference*, CCC '23, Dagstuhl, DEU, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2023.11.

4 Oded Goldreich. *In a World of P=BPP*, pages 191–232. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-22670-0_20.

5 Russell Impagliazzo and Avi Wigderson. P = bpp if e requires exponential circuits: derandomizing the xor lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 220–229, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258590.

6 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13:1–46, 2004. doi:10.1007/s00037-004-0182-6.

7 Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31(5):1501–1526, 2002. doi:10.1137/S0097539700389652.

8 Yanyi Liu and Rafael Pass. Leakage-resilient hardness vs randomness. In *Proceedings of the Conference on Proceedings of the 38th Computational Complexity Conference*, CCC '23, Dagstuhl, DEU, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2023.32.

9 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, 1994. doi:10.1016/S0022-0000(05)80043-1.

10 Rafael Pass and Oren Renard. Characterizing the power of (persistent) randomness in log-space. Electronic Consortium on Computational Complexity, 2023. URL: https://eccc.weizmann.ac.il/report/2023/211/.

11 Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In Franz Pichler, editor, *Advances in Cryptology—EUROCRYPT '85, Workshop on the Theory and Applications of Cryptographic Techniques*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34. Springer, April 1985. doi:10.1007/3-540-39805-8_3.

12 Yakov Shalunov. Leakage-resilient hardness equivalence to logspace derandomization. arXiv. arXiv:2312.14023.

13 Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001. doi:10.1006/jcss.2000.1730.

14 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012. doi:10.1561/0400000010.

15 Andrew C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 80–91, 1982. doi:10.1109/SFCS.1982.45.

Faster Approximation Schemes for (Constrained) k-Means with Outliers

Zhen Zhang

School of Advanced Interdisciplinary Studies, Hunan University of Technology and Business, Changsha, China
Xiangjiang Laboratory, Changsha, China

Junyu Huang

School of Computer Science and Engineering, Central South University, Changsha, China

Qilong Feng

School of Computer Science and Engineering, Central South University, Changsha, China

Abstract

Given a set of n points in $\mathbb{R}^d$ and two positive integers k and m, the Euclidean k-means with outliers problem aims to remove at most m points, referred to as outliers, and minimize the k-means cost function for the remaining points. Developing algorithms for this problem remains an active area of research due to its prevalence in applications involving noisy data. In this paper, we give a $(1+\varepsilon)$-approximation algorithm that runs in $n^2d((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time for the problem. When combined with a coreset construction method, the running time of the algorithm can be improved to be linear in n. For the case where k is a constant, this represents the first polynomial-time approximation scheme for the problem: Existing algorithms with the same approximation guarantee run in polynomial time only when both k and m are constants. Furthermore, our approach generalizes to variants of k-means with outliers incorporating additional constraints on instances, such as those related to capacities and fairness.

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

Keywords and phrases Approximation algorithms, clustering

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.84

Funding This work was supported by National Natural Science Foundation of China (62202161, 62172446), Natural Science Foundation of Hunan Province (2023JJ40240), and Scientific Research Fund of Hunan Provincial Education Department (23B0597).

1 Introduction

Clustering is a frequently encountered task in many fields related to machine learning, aiming to partition a given set of points into several cohesive clusters. Among the various ways of formalizing the task of clustering, the Euclidean k*-means* problem is perhaps the most commonly studied one. In this problem, we are given a set $\mathcal{P} \subset \mathbb{R}^d$ of points and a positive integer k, and the goal is to identify a set $\mathcal{C} \subset \mathbb{R}^d$ of no more than k centers so that the objective function $\sum_{p\in\mathcal{P}} \min_{c\in\mathcal{C}} ||p-c||^2$ is minimized. Here, the points are partitioned into different clusters according to the disparities in their corresponding centers, namely, the nearest ones. In most applications of the problem, the upper bound on the number of centers (i.e., k) is significantly smaller than the number of points to be clustered. This prompts considerable efforts in developing algorithms for the case where k is fixed. Specifically, it is known that the problem admits *polynomial-time approximation schemes* (PTASs) when k is a constant [14, 18, 11, 37, 33, 34].

Despite extensive study, algorithms developed for the Euclidean k-means problem often exhibit poor performance. The main issue lies in the lack of robustness of the objective function to noisy data: A few *outliers* within the point set can significantly impact the

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 84; pp. 84:1–84:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

value of the function. Removing these outliers typically leads to a better clustering result. Motivated thus, we consider the Euclidean *k-means with outliers* (k-MeansOut) problem, which can be defined as follows.

▶ **Definition 1** (Euclidean k-MeansOut). *An instance of Euclidean k-MeansOut is specified by a set $\mathcal{P} \subset \mathbb{R}^d$ of points and two positive integers k and m. A feasible solution to the instance is a set $\mathcal{C} \subset \mathbb{R}^d$ of centers satisfying $|\mathcal{C}| \leq k$ and a set $\mathcal{O} \subseteq \mathcal{P}$ of outliers satisfying $|\mathcal{O}| \leq m$. The cost of such a solution is $\sum_{p\in\mathcal{P}\setminus\mathcal{O}} \min_{c\in\mathcal{C}} ||p-c||^2$. The goal of Euclidean k-MeansOut is to find a feasible solution with minimum cost.*

Solving the Euclidean k-MeansOut problem helps to the removal of more interpretable outliers that can be contextualized by the clusters. This, in turn, results in more cohesive clusters. Notably, it has been observed that adopting this joint perspective on outlier detection and clustering leads to improved performance, even when solely focusing on the task of outlier removal [9, 25]. Given its important role in dealing with noisy data, the Euclidean k-MeansOut problem has received lots of attention from both theoretical and practical points of view. A series of algorithms have been proposed for the problem, including heuristics [9], distributed algorithms [10, 39, 24, 23, 27], approximation algorithms [25, 36, 6, 15, 30, 41], and coreset-construction methods [17, 19, 28, 29].

A commonly used way for relaxing the Euclidean k-MeansOut problem is to assume that the upper bounds on the numbers of centers and outliers (i.e., k and m) are small constants. Under this assumption, several PTASs exist for the problem. Feldman and Schulman [19] showed that a coreset-based approach yields a $(1+\varepsilon)$-approximation algorithm running in $nd(k+m)^{O(k+m)} + (\varepsilon^{-1}k\log n)^{O(1)}$ time, where n denotes the size of the given point set. Bhattacharya et al. [7] later gave an outlier-to-outlier-free reduction, where they mapped an instance of k-MeansOut to an instance of the standard k-means problem. This incurs an arbitrarily small loss in the approximation ratio and a $(k+m)^{m\varepsilon^{-O(1)}}$ multiplicative overhead on the running time of the executed algorithm. Subsequently, Agrawal et al. [1] and Jaiswal and Kumar [32] gave different reductions that impose multiplicative overheads of $n^{O(1)}((k+m)\varepsilon^{-1})^{O(m)}$ and $((k+m)\varepsilon^{-1})^{O(m)}$ on the running time, respectively. When combined with the state-of-the-art approximation scheme running in $O(ndk + d(k\varepsilon^{-1})^{O(1)} + (k\varepsilon^{-1})^{O(k\varepsilon^{-1})})$ time for the Euclidean k-means problem [18], these reductions yield $(1+\varepsilon)$-approximation algorithms with running times exponentially dependent on k and m. When k and m are not fixed, PTASs for Euclidean k-MeansOut exist for the case where d is a constant and the upper-bound constraint on the number of centers can be slightly violated, including the $(d\varepsilon^{-1})^{O(d)}$-swap local-search algorithm given by Friggstad et al. [20] and the algorithm based on split-tree decomposition given by Cohen-Addad et al. [12]. These existing $(1+\varepsilon)$-approximation results are summarized in Table 1.

1.1 Our Results

As described above, there are $(1+\varepsilon)$-approximation algorithms for Euclidean k-MeansOut with running time exponential in both k and m. We cannot hope to achieve a better solution (i.e., an optimal one) in the same time frame: Euclidean k-MeansOut has been shown to be NP-hard, even for the case where $k = 2$ and $m = 0$ [38]. Nevertheless, this negative result does not rule out the possibility of achieving a $(1+\varepsilon)$-approximation solution in a more efficient manner. In particular, given that the outliers often constitute a constant fraction of the entire point set [21, 22, 15], it is interesting to consider whether a $(1+\varepsilon)$-approximation algorithm without exponential dependence on m exists in high-dimensional spaces. The main result in this paper is the first affirmative answer to this question, as described in Theorem 2.

Table 1 $(1+\varepsilon)$-approximation algorithms for Euclidean k-MeansOut. The first two are bi-criteria approximation algorithms that violate the upper-bound constraint on the number of centers by a factor of $1+O(\varepsilon)$. $T(n,d,k,\varepsilon) = O(ndk + d(k\varepsilon^{-1})^{O(1)} + (k\varepsilon^{-1})^{O(k\varepsilon^{-1})})$ denotes the running time of the state-of-the-art approximation scheme for the Euclidean k-means problem.

Running time	Parameter(s) in the exponent	Reference
$(nk)^{(d\varepsilon^{-1})^{O(d)}}$	d	[20]
$2^{\varepsilon^{-O(d^2)}} n \log^{O(1)} n + n^{O(1)}$	d	[12]
$nd(k+m)^{O(k+m)} + (\varepsilon^{-1} k \log n)^{O(1)}$	k, m	[19]
$(k+m)^{m\varepsilon^{-O(1)}} T(n,d,k,\varepsilon)$	k, m	[7]
$n^{O(1)}((k+m)\varepsilon^{-1})^{O(m)} T(n,d,k,\varepsilon)$	k, m	[1]
$((k+m)\varepsilon^{-1})^{O(m)} T(n,d,k,\varepsilon)$	k, m	[32]
$nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$	k	This work

▶ **Theorem 2.** *Given a constant $\varepsilon \in (0,1)$ and an instance $(\mathcal{P}, k, m)$ of Euclidean k-MeansOut satisfying $\mathcal{P} \subset \mathbb{R}^d$ and $|\mathcal{P}| = n$, there is a $(1+\varepsilon)$-approximation algorithm running in $n^2 d((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time.*

Leveraging a coreset construction method that reduces the point set to a weighted set of size poly(m, k, ε^{-1}) [29], we can improve the running time of the algorithm in Theorem 2 to be linear in n. A detailed analysis is given in Section 4.2.

Awasthi et al. [3] showed that obtaining a PTAS for Euclidean k-MeansOut for arbitrary k and $d = \Omega(\log n)$ is also NP-hard. Given that we avoid the exponential dependence on d, the exponential dependence on k exhibited in Theorem 2 is unavoidable. Existing approximation schemes without exponential dependence on k, like the ones in [20, 12], work only in low-dimensional spaces and select more than k centers.

The optimal solutions to instances of k-MeansOut exhibit a useful property: The center associated with each point is simply the one nearest to the point. This property, called the *locality* property, guides the estimation of the locations of centers selected by an optimal solution. One advantage of our approach is that it no longer relies on the locality property. This enhances the versatility of the approach, allowing us to deal with problems not satisfying the locality property. Indeed, we show that our approach establishes a unified framework for addressing generalizations of Euclidean k-MeansOut that invalidate the locality property, including the Euclidean versions of *capacitated* and *fair* k-MeansOut [32, 13]. As in the unconstrained case, we give the first PTAS for each considered generalization of Euclidean k-MeansOut, assuming k is a constant.

1.2 Our Techniques

Most existing approximation schemes for k-MeansOut are built heavily on the following natural idea: The m outliers can be viewed as m virtual centers, each corresponding to a cluster containing only itself, and solving a $(k+m)$-clustering problem enables the identification of the k centers and m outliers. This provides a clear strategy for constructing the desired approximation solution. However, the complexities of clustering problems increase with the number of centers to be identified, and considering the additional m virtual centers incurs an exponential time-dependence on m, as exhibited in the running times of the approximation schemes given in [19, 7, 1, 32]. To deal with the case where m is super-constant, we employ a different sampling-based method.

Given a small positive constant ε, it is well-known that the centroid of $O(\varepsilon^{-1})$ uniformly sampled points is close to the optimal 1-means clustering center of the entire point set (Lemma 6). Building upon this insight, we uniformly sample from the point set for each cluster defined by an optimal solution, and enumerate the sampled points to find a subset of $O(\varepsilon^{-1})$ points uniformly distributed in the cluster, such that the corresponding center can be approximated by the centroid of the subset. This idea follows the one for the k-means problem outlined in [37], while the case we consider poses more challenges. The first issue we encounter lies in the presence of a non-fixed number of outliers, which reduces the proportion of some small clusters within the point set, and so, the likelihood of obtaining members of these clusters through randomly sampling may not be sufficiently high. To address this issue, we carefully adjust the sampling region for each cluster in a recursive way, ensuring a sufficient proportion of the cluster within the defined region. Another issue emerges as we extend our consideration to the constrained variants of k-MeansOut that do not satisfy the locality property. In these variants, the points are not guaranteed to be close to their corresponding centers. This leads to the lack of a distinct pattern in the distributions of the clusters, making it more difficult to determine appropriate sampling regions. In dealing with this issue, it is essential to address the points that violate the locality property, meaning those far from their corresponding centers. Instead of attempting to find these points through sampling, we regard previously identified approximate centers close to these points as substitutes, enumerating the union of the sampled points and the previously selected centers to construct a small representative set for the considered cluster.

2 Preliminaries

Given a positive integer λ, define $[\lambda] = \{1, \cdots, \lambda\}$. Given a set $\mathcal{X} \subset \mathbb{R}^d$ and a point $y \in \mathbb{R}^d$, let $\Delta(y, \mathcal{X}) = \min_{x \in \mathcal{X}} ||y - x||^2$ denote the squared distance from y to the nearest point in $\mathcal{X}$, and let $\Delta(\mathcal{X}, y) = \sum_{x \in \mathcal{X}} ||x - y||^2$ denote the sum of squared distances from y to the points in $\mathcal{X}$. Additionally, define $c(\mathcal{X}) = |\mathcal{X}|^{-1} \sum_{x \in \mathcal{X}} x$ as the centroid of $\mathcal{X}$, and let $\Delta(\mathcal{X}) = \min_{c \in \mathbb{R}^d} \Delta(\mathcal{X}, c)$ denote the minimum 1-means clustering cost of $\mathcal{X}$.

The following two lemmas provide ways of estimating the squared distances from the points to the centers selected by an approximate solution. As a corollary of the first one, we know that $\Delta(\mathcal{X}) = \Delta(\mathcal{X}, c(\mathcal{X}))$ for each $\mathcal{X} \subset \mathbb{R}^d$.

▶ **Lemma 3** ([35]). *Given a point $x \in \mathbb{R}^d$ and a set $\mathcal{X} \subset \mathbb{R}^d$, we have $\Delta(\mathcal{X}, x) = \Delta(\mathcal{X}) + |\mathcal{X}| \cdot ||c(\mathcal{X}) - x||^2$.*

▶ **Lemma 4** ([16]). *Given a set $\mathcal{X} \subset \mathbb{R}^d$, a real number $\lambda \in (0, 1]$, and a subset $\mathcal{X}' \subseteq \mathcal{X}$ satisfying $|\mathcal{X}'| \geq \lambda|\mathcal{X}|$, we have $||c(\mathcal{X}') - c(\mathcal{X})||^2 \leq (1 - \lambda)(\lambda|\mathcal{X}|)^{-1}\Delta(\mathcal{X})$.*

The following lemma is an extensive version of triangle inequality.

▶ **Lemma 5.** *Given three points x, y, and z in $\mathbb{R}^d$ and a real number $\lambda > 0$, we have $||x - z||^2 \leq (1 + \lambda)||x - y||^2 + (1 + \lambda^{-1})||y - z||^2$.*

Proof. Using triangle inequality, we have$||x - z|| \leq ||x - y|| + ||y - z||$, which implies that

$$\begin{aligned} ||x - z||^2 \leq& (||x - y|| + ||y - z||)^2 \\ =& ||x - y||^2 + ||y - z||^2 + 2\sqrt{\lambda}||x - y||\frac{1}{\sqrt{\lambda}}||y - z|| \\ \leq& ||x - y||^2 + ||y - z||^2 + \lambda||x - y||^2 + \frac{1}{\lambda}||y - z||^2, \end{aligned}$$

as desired. ◀

The following result says that uniform sampling works for the 1-means problem.

▶ **Lemma 6** ([31]). *Given a set $\mathcal{X} \subset \mathbb{R}^d$, a multi-set $\mathcal{S}$ constructed by sampling points from $\mathcal{X}$ independently and uniformly, and a positive real number λ, inequality $||c(\mathcal{S}) - c(\mathcal{X})||^2 \leq (\lambda|\mathcal{S}||\mathcal{X}|)^{-1}\Delta(\mathcal{X})$ holds with probability at least $1 - \lambda$.*

The following result is known as Chernoff bound, which has been widely used in analysis of sampling-based algorithms.

▶ **Lemma 7** ([26]). *Given a set of t independent random variables $a_1, \cdots, a_t$ and a real number $p \in (0,1)$, if $a_i \in \{0,1\}$ and $\mathbf{Pr}[a_i = 1] \geq p$ hold for each $i \in [t]$, then each real number $\lambda \in (0,1)$ satisfies $\mathbf{Pr}\left[\sum_{i=1}^{t} a_i < (1-\lambda)pt\right] < e^{-\frac{1}{2}\lambda^2 pt}$.*

As a corollary of Lemma 7, we have the following result about uniform sampling.

▶ **Lemma 8.** *Given a set $\mathcal{X} \subset \mathbb{R}^d$, a subset $\mathcal{S} \subseteq \mathcal{X}$, a positive integer t, and a real number $\lambda \in (0,1)$, the following event happens with probability more than $1 - e^{-\frac{\lambda^2 t|\mathcal{S}|}{2|\mathcal{X}|}}$: A multi-set of more than t points independently and uniformly sampled from $\mathcal{X}$ contains no less than $(1-\lambda)t|\mathcal{S}||\mathcal{X}|^{-1}$ points in $\mathcal{S}$.*

Proof. We define a set of independent random variables $a_1, \cdots, a_t$ as follows: For each $i \in [t]$, let $a_i = 1$ if the i-th point sampled from $\mathcal{X}$ is in $\mathcal{S}$, and let $a_i = 0$ otherwise. We have $\mathbf{Pr}[a_i = 1] = |\mathcal{S}||\mathcal{X}|^{-1}$ for each $i \in [t]$. Lemma 7 implies that

$$\mathbf{Pr}[\sum_{i=1}^{t} a_i \geq (1-\lambda)t|\mathcal{S}||\mathcal{X}|^{-1}] = 1 - \mathbf{Pr}[\sum_{i=1}^{t} a_i < (1-\lambda)t|\mathcal{S}||\mathcal{X}|^{-1}] > 1 - e^{-\frac{\lambda^2 t|\mathcal{S}|}{2|\mathcal{X}|}}.$$

This completes the proof of Lemma 8. ◀

3 The Sampling Algorithm

In this section we give a sampling-based approach for constructing candidate center sets, as described in Algorithm 1. Taking as inputs three real numbers k, m, and ε, two sets $\mathcal{C}'$ and $\mathcal{P}^\dagger$, and a collection $\mathbb{C}$, the algorithm recursively augments $\mathbb{C}$ with some center sets. Here, k is the upper bound on the size of a center set, m is the upper bound on the number of outliers, ε is the factor trading off the approximation ratio and running time, $\mathcal{C}'$ is a center set that needs to be updated or added to $\mathbb{C}$, $\mathcal{P}^\dagger$ is the sampling region, and $\mathbb{C}$ contains the center sets that have been constructed. The algorithm constructs a multi-set $\mathcal{S}$ as follows: It independently and uniformly samples $O((k+m)\varepsilon^{-3})$ points from $\mathcal{P}^\dagger$ and then adds to the set $O(\varepsilon^{-1})$ copies of each center in $\mathcal{C}'$. After constructing $\mathcal{S}$, the algorithm considers each subset of size $O(\varepsilon^{-1})$ of $\mathcal{S}$, adding the centroid of the subset to $\mathcal{C}'$ and recursively invoking itself with the updated center set. Finally, it throws away half of the points in $\mathcal{P}^\dagger$ that are close to the centers in $\mathcal{C}'$, and invokes itself again with the reduced point set.

By inducting on the sizes of the given sets of centers and points, we obtain the following upper bounds on the running time of our sampling algorithm and the quantity of center sets it generates.

▶ **Lemma 9.** *Given a constant $\varepsilon \in (0,1)$, a collection $\mathbb{C}$, and an instance $(\mathcal{P}, k, m)$ of Euclidean k-MeansOut with $|\mathcal{P}| \leq n$ and $\mathcal{P} \subset \mathbb{R}^d$,* **Sampling**$(k, m, \varepsilon, \emptyset, \mathcal{P}, \mathbb{C})$ *runs in $nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time and adds at most $n((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ center sets to $\mathbb{C}$.*

Algorithm 1 **Sampling**$(k, m, \varepsilon, \mathcal{C}', \mathcal{P}^\dagger, \mathbb{C})$.

```
   Input: Two positive integers $k$ and $m$, a constant $\varepsilon \in (0,1)$, a set $\mathcal{C}' \subset \mathbb{R}^d$ of no more
          than $k$ centers, a set $\mathcal{P}^\dagger \subset \mathbb{R}^d$ of points, and a collection $\mathbb{C}$ of center sets
1  $N \Leftarrow \lceil (17400k + 60m)\varepsilon^{-3} \rceil$, $M \Leftarrow \lceil 25\varepsilon^{-1} \rceil$;
2  if $|\mathcal{C}'| = k$ then
3      $\mathbb{C} \Leftarrow \mathbb{C} \cup \{\mathcal{C}'\}$;
4  else
5      Sample a multi-set $\mathcal{S}$ of $N$ points from $\mathcal{P}^\dagger$ independently and uniformly;
6      $\mathcal{S} \Leftarrow \mathcal{S} \uplus \{M$ copies of each $c \in \mathcal{C}'\}$;
7      for each $\mathcal{S}' \subset \mathcal{S}$ satisfying $|\mathcal{S}'| = M$ do
8          Calculate the centroid $c(\mathcal{S}')$ of $\mathcal{S}'$;
9          Sampling$(k, m, \varepsilon, \mathcal{C}' \uplus \{c(\mathcal{S}')\}, \mathcal{P}^\dagger, \mathbb{C})$;
10     if $\mathcal{C}' \neq \emptyset$ and $|\mathcal{P}^\dagger| > 1$ then
11         Let $\mathcal{P}^\ddagger$ be the set of the $\lfloor \frac{|\mathcal{P}^\dagger|}{2} \rfloor$ points $p \in \mathcal{P}^\dagger$ with the largest values of
           $\Delta(p, \mathcal{C}')$;
12         Sampling$(k, m, \varepsilon, \mathcal{C}', \mathcal{P}^\ddagger, \mathbb{C})$.
```

3.1 An Overview of Analysis

We now introduce some notations to be used throughout this section. We consider a constant $\varepsilon \in (0,1)$ and an instance $\mathcal{I} = (\mathcal{P}, k, m)$ of Euclidean k-MeansOut, where $\mathcal{P} \subset \mathbb{R}^d$ and $|\mathcal{P}| = n$. Let $N = \lceil (17400k + 60m)\varepsilon^{-3} \rceil$ and $M = \lceil 25\varepsilon^{-1} \rceil$. Let $\mathcal{P}_1, \cdots, \mathcal{P}_k, \mathcal{O}$ denote $k+1$ arbitrary disjoint subsets of $\mathcal{P}$ satisfying $|\mathcal{O}| = m$, $\bigcup_{i=1}^k \mathcal{P}_i \cup \mathcal{O} = \mathcal{P}$, and $|\mathcal{P}_1| \geq |\mathcal{P}_2| \geq \cdots \geq |\mathcal{P}_k|$. For each $i \in [k]$, let $c_i^* = c(\mathcal{P}_i)$ be the centroid of $\mathcal{P}_i$. Define $\Delta(\mathbb{P}) = \sum_{i=1}^k \Delta(\mathcal{P}_i)$. Let $\mathbb{C}$ denote the collection of center sets constructed by **Sampling**$(k, m, \varepsilon, \emptyset, \mathcal{P}, \emptyset)$. We will show that $\mathbb{C}$ contains a center set approximating $\{c_1^*, \ldots, c_k^*\}$ well with high probability. More formally, we will prove the correctness of the following result.

▶ **Lemma 10.** *The following event happens with probability no less than 15^{-k}: There is a center set $\mathcal{C} \in \mathbb{C}$ satisfying $\sum_{i=1}^k \min_{c \in \mathcal{C}} \Delta(\mathcal{P}_i, c) \leq (1+\varepsilon)\Delta(\mathbb{P})$.*

The proof of Lemma 10, presented in Section 3.2, is based on an inductive method. Specifically, for a given integer $i \in \{2, \cdots, k\}$, we assume that a set $\mathcal{C}_{i-1} = \{c_1, \cdots, c_{i-1}\}$ of centers, where c_j is close to c_j^* for each $j \in [i-1]$, has been constructed, and prove that a center close to c_i^* can be identified and added to $\mathcal{C}_{i-1}$ when invoking Algorithm 1 with $\mathcal{C}_{i-1}$. Define (informally) $\mathcal{B}_{i-1}$ as the set of points close to one of the centers in $\mathcal{C}_{i-1}$. Given a real number λ defined based on the value of ε, we divide the analysis into the following two cases: (1) $|\mathcal{P}_i \backslash \mathcal{B}_{i-1}| \leq \lambda |\mathcal{P}_i|$, and (2) $|\mathcal{P}_i \backslash \mathcal{B}_{i-1}| > \lambda |\mathcal{P}_i|$.

The case of $|\mathcal{P}_i \backslash \mathcal{B}_{i-1}| \leq \lambda |\mathcal{P}_i|$ captures the scenario where most points in $\mathcal{P}_i$ are from $\mathcal{B}_{i-1}$ and in close proximity to one of the centers in $\mathcal{C}_{i-1}$. Conditioned on this, the centers copied in step 6 of Algorithm 1 can be regarded as proxies for the points in $\mathcal{P}_i$. Based on Lemma 4 and Lemma 6, we are able to show that the centroid of a subset of the copies is close to c_i^* and can be added to $\mathcal{C}_{i-1}$ by the algorithm.

For the case where $|\mathcal{P}_i \backslash \mathcal{B}_{i-1}| > \lambda |\mathcal{P}_i|$, we handle the points from $\mathcal{P}_i \cap \mathcal{B}_{i-1}$ similarly to the above approach: We regard the copies of centers in $\mathcal{C}_{i-1}$ as proxies of these points. It remains to consider how to deal with the points from $\mathcal{P}_i \backslash \mathcal{B}_{i-1}$. When formally defining $\mathcal{B}_{i-1}$ in Section 3.2, we carefully establish its range such that the ratio of $|\mathcal{P} \backslash \mathcal{B}_{i-1}|$ to $|\mathcal{P}_i \backslash \mathcal{B}_{i-1}|$ is polynomial in m, k, and ε if $|\mathcal{P}_i \backslash \mathcal{B}_{i-1}| > \lambda|\mathcal{P}_i|$ (as exhibited in Claim 15), which implies that a limited number of points uniformly sampled from $\mathcal{P}\backslash\mathcal{B}_{i-1}$ contains a representative subset of $\mathcal{P}_i\backslash\mathcal{B}_{i-1}$ with a good chance. Furthermore, it is shown that Algorithm 1 can recursively adjust the sampling region to make it close to $\mathcal{P}\backslash\mathcal{B}_{i-1}$, based on the operation in step 12. Putting everything together, we know that a representative subset of $\mathcal{P}_i\backslash\mathcal{B}_{i-1}$ is involved in the points sampled by the algorithm.

3.2 Proof of Lemma 10

It can be seen that the algorithm **Sampling** makes multiple recursive calls to itself. We conceptualize the execution of **Sampling**$(k, m, \varepsilon, \emptyset, \mathcal{P}, \emptyset)$ as a tree denoted by $\mathcal{T}$. Each node within the tree, identified by $(\mathcal{C}', \mathcal{P}^\dagger)$, corresponds to an invocation of the algorithm with center set $\mathcal{C}'$ and sampling region $\mathcal{P}^\dagger$. The children of a node symbolize the recursive calls made in the corresponding invocation of the algorithm, and each leaf of the tree is associated with a set of k centers added to $\mathbb{C}$.

Prior to showing the correctness of Lemma 10, we establish the following invariant for each $i \in [k]$.

$\tau(i)$**:** With probability at least 15^{-i}, there exists a node $(\mathcal{C}_i, \mathcal{P}^\dagger)$ in $\mathcal{T}$ such that (1) $\mathcal{C}_i$ consists of i centers $c_1, \cdots, c_i$ (added in this order), (2) each $j \in [i]$ satisfies $\Delta(\mathcal{P}_j, c_j) \leq (1 + \frac{\varepsilon}{2})\Delta(\mathcal{P}_j) + \frac{\varepsilon}{2k}\Delta(\mathbb{P})$, and (3) $\{p \in \mathcal{P} : \Delta(p, \mathcal{C}_i) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} \subseteq \mathcal{P}^\dagger$.

We prove the invariant by induction on i. We first consider the base case of $i = 1$. It can be seen that **Sampling**$(k, m, \varepsilon, \emptyset, \mathcal{P}, \emptyset)$ independently and uniformly samples a multi-set $\mathcal{S}$ of N points from $\mathcal{P}$, and $\mathcal{T}$ has a node $(\{c(\mathcal{S}')\}, \mathcal{P})$ for each $\mathcal{S}' \subset \mathcal{S}$ with $|\mathcal{S}'| = M$. We have $|\mathcal{P}||\mathcal{P}_1|^{-1} = (\sum_{j=1}^{k} |\mathcal{P}_j| + |\mathcal{O}|)|\mathcal{P}_1|^{-1} \leq k + m$ due to the fact that $|\mathcal{P}_1| \geq |\mathcal{P}_2| \geq \cdots \geq |\mathcal{P}_k|$. This inequality and Lemma 8 (with $t = N$ and $\lambda = 1 - M(k + m)N^{-1}$) imply that a node $(\{c(\mathcal{S}')\}, \mathcal{P})$ satisfying $|\mathcal{S}'| = M$ and $\mathcal{S}' \subseteq \mathcal{P}_1$ exists in $\mathcal{T}$ with probability more than $1 - e^{-(1-M(k+m)N^{-1})^2N)/(2(k+m))} > \frac{1}{3}$. Using Lemma 6 (with $\lambda = \frac{4}{5}$), we know that if such a node $(\{c(\mathcal{S}')\}, \mathcal{P})$ exists, then inequality

$$||c(\mathcal{S}') - c_1^*||^2 \leq \frac{5\Delta(\mathcal{P}_1)}{4|\mathcal{P}_1||\mathcal{S}'|} = \frac{\varepsilon\Delta(\mathcal{P}_1)}{20|\mathcal{P}_1|} \tag{1}$$

holds with probability at least $\frac{1}{5}$. Inequality (1) and Lemma 3 imply that

$$\Delta(\mathcal{P}_1, c(\mathcal{S}')) = \Delta(\mathcal{P}_1) + |\mathcal{P}_1| \cdot ||c(\mathcal{S}') - c_1^*||^2 \leq (1 + \frac{\varepsilon}{20})\Delta(\mathcal{P}_1),$$

which in turn implies that $\tau(1)$ is true.

We now assume that $\tau(i - 1)$ holds for an integer $i \in \{2, \cdots, k\}$, and prove that $\tau(i)$ also holds. Let $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ be a node satisfying

$$\Delta(\mathcal{P}_j, c_j) \leq (1 + \frac{\varepsilon}{2})\Delta(\mathcal{P}_j) + \frac{\varepsilon}{2k}\Delta(\mathbb{P}) \tag{2}$$

for each $j \in [i - 1]$ and

$$\{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_{i-1}|}\} \subseteq \mathcal{P}^\dagger, \tag{3}$$

where $\mathcal{C}_{i-1} = \{c_1, \cdots, c_{i-1}\}$. $\tau(i - 1)$ implies that such a node $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ exists in $\mathcal{T}$ with probability no less than 15^{1-i}. Conditioning on the existence of this node, we define $\mathcal{B}_{i-1} = \{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1}) \leq \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\}$. Let $\mathcal{P}_i^n = \mathcal{P}_i \cap \mathcal{B}_{i-1}$ and $\mathcal{P}_i^f = \mathcal{P}_i\backslash\mathcal{B}_{i-1}$ for brevity. As described in Section 3.1, we prove $\tau(i)$ differently based on the size of $\mathcal{P}_i^f$. Specifically, we consider the following two cases: (1) $|\mathcal{P}_i^f| \leq \frac{\varepsilon}{17}|\mathcal{P}_i|$, and (2) $|\mathcal{P}_i^f| > \frac{\varepsilon}{17}|\mathcal{P}_i|$. These two cases are respectively analyzed in the following two subsections.

Case (1): $|\mathcal{P}_i^f| \leq \frac{\varepsilon}{17}|\mathcal{P}_i|$

In this case, most points in $\mathcal{P}_i$ are close to the centers in $\mathcal{C}_{i-1}$, based on which we show that a convex combination of the latter's members effectively approximates c_i^*. We consider a multi-set $\mathcal{P}_i' = \{c(p) : p \in \mathcal{P}_i^n\}$, where $c(p)$ is the center in $\mathcal{C}_{i-1}$ nearest to p. The proximity of each point in $\mathcal{P}_i^n$ to its counterpart in $\mathcal{P}_i'$, combined with the substantial proportion of $\mathcal{P}_i^n$ in $\mathcal{P}_i$, implies that the centroid of $\mathcal{P}_i'$ is close to c_i^*. This is confirmed by the following lemma.

► **Lemma 11.** *If $|\mathcal{P}_i^f| \leq \frac{\varepsilon}{17}|\mathcal{P}_i|$, then we have $||c(\mathcal{P}_i') - c_i^*||^2 \leq \frac{\varepsilon\Delta(\mathcal{P}_i)}{8|\mathcal{P}_i|} + \frac{\varepsilon\Delta(\mathbb{P})}{4k|\mathcal{P}_i|}$.*

Lemma 11 suggests that $c(\mathcal{P}_i')$ is close to c_i^*. Unfortunately, directly approximating c_i^* using $c(\mathcal{P}_i')$ is not feasible, as the members of both $\mathcal{P}_i^n$ and $\mathcal{P}_i'$ are unknown. The idea of Algorithm 1 is to take M copies of each center from $\mathcal{C}_{i-1}$ and simulate $\mathcal{P}_i'$ using a subset of these copies. The following lemma implies that this yields a center approximating c_i^* well with high probability and, furthermore, generates the node claimed in $\tau(i)$.

► **Lemma 12.** *If $|\mathcal{P}_i^f| \leq \frac{\varepsilon}{17}|\mathcal{P}_i|$, then the following event happens with probability at least $\frac{1}{5}$: $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ has a child $(\mathcal{C}_{i-1} \uplus \{c_i\}, \mathcal{P}^\dagger)$ satisfying $\{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1} \uplus \{c_i\}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} \subseteq \mathcal{P}^\dagger$ and $\Delta(\mathcal{P}_i, c_i) < (1 + \frac{\varepsilon}{2})\Delta(\mathcal{P}_i) + \frac{\varepsilon}{2k}\Delta(\mathbb{P})$.*

Proof. The fact that $|\mathcal{P}_1| \geq |\mathcal{P}_2| \geq \cdots \geq |\mathcal{P}_k|$ and $\Delta(p, \mathcal{C}_{i-1}) \geq \Delta(p, \mathcal{C}_{i-1} \uplus \{c\})$ for each $p \in \mathcal{P}$ and $c \in \mathbb{R}^d$ suggests that

$$\{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1} \uplus \{c\}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} \subseteq \{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_{i-1}|}\} \subseteq \mathcal{P}^\dagger \tag{4}$$

for each $c \in \mathbb{R}^d$, where the last step is due to inequality (3).

In the invocation of Algorithm 1 corresponding to $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$, the algorithm takes M copies of each center from $\mathcal{C}_{i-1}$ and calculates the centroid of each subset of the copies with size M. By Lemma 6 (with $\lambda = \frac{4}{5}$) and the fact that $\mathcal{P}_i' \subseteq \mathcal{C}_{i-1}$, we know that a center c_i identified in this way satisfies

$$||c_i - c(\mathcal{P}_i')||^2 \leq \frac{5\Delta(\mathcal{P}_i')}{4M|\mathcal{P}_i'|} \leq \frac{\varepsilon\Delta(\mathcal{P}_i')}{20|\mathcal{P}_i'|} = \frac{\varepsilon\Delta(\mathcal{P}_i')}{20|\mathcal{P}_i^n|} \tag{5}$$

with probability no less than $\frac{1}{5}$.

Denote by c_i a center satisfying inequality (5). Intuitively, inequality (5) and Lemma 11 imply an upper bound on the squared distance from c_i to c_i^*. Combining this insight with Lemma 3, we can derive the following claim.

▷ Claim 13. If $|\mathcal{P}_i^f| \leq \frac{\varepsilon}{17}|\mathcal{P}_i|$, then we have $\Delta(\mathcal{P}_i, c_i) < (1 + \frac{\varepsilon}{2})\Delta(\mathcal{P}_i) + \frac{\varepsilon}{2k}\Delta(\mathbb{P})$.

Using Claim 13 and inequality (4), we complete the proof of Lemma 12. ◄

Case (2): $|\mathcal{P}_i^f| > \frac{\varepsilon}{17}|\mathcal{P}_i|$

As in the previous case, we simulate the points in $\mathcal{P}_i^n$ using the centers in $\mathcal{C}_{i-1}$. The main challenge in the current case is that we cannot ignore the points from $\mathcal{P}_i^f$ as we did previously, since their proportion in $\mathcal{P}_i$ is no longer bounded by a small value. As a remedy, we argue that we can sample sufficient points from $\mathcal{P}_i^f$ in step 5 of Algorithm 1 by recursively adjusting the sampling region. Furthermore, we will show that a combination of these sampled points and the centers in $\mathcal{C}_{i-1}$ approximates c_i^* well.

The following lemma suggests a lower bound on the proportion of $\mathcal{P}_i^f$ within a carefully selected sampling region.

▶ **Lemma 14.** *If $|\mathcal{P}_i^f| > \frac{\varepsilon}{17}|\mathcal{P}_i|$, then there is a node $(\mathcal{C}_{i-1}, \mathcal{P}^\ddagger)$ in the descendants of $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ (including itself) that satisfies $\mathcal{P}\backslash\mathcal{B}_{i-1} \subseteq \mathcal{P}^\ddagger$ and $\varepsilon^{-2}(580k + 2m)|\mathcal{P}_i^f| > |\mathcal{P}^\ddagger|$.*

Proof. Our idea for proving Lemma 14 is to show that $|\mathcal{P}_i^f|$ is not too small compared to $|\mathcal{P}\backslash\mathcal{B}_{i-1}|$, and then argue that the invocation of Algorithm 1 corresponding to $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ can find a sampling region $\mathcal{P}^\ddagger$ close to $\mathcal{P}\backslash\mathcal{B}_{i-1}$.

The following claim establishes a lower bound on the ratio of $|\mathcal{P}_i^f|$ to $|\mathcal{P}\backslash\mathcal{B}_{i-1}|$.

▷ Claim 15. If $|\mathcal{P}_i^f| > \frac{\varepsilon}{17}|\mathcal{P}_i|$, then $\varepsilon^{-2}(290k + m)|\mathcal{P}_i^f| > |\mathcal{P}\backslash\mathcal{B}_{i-1}|$.

The fact that $|\mathcal{P}_1| \geq |\mathcal{P}_2| \geq \cdots \geq |\mathcal{P}_k|$ and inequality (3) imply that

$$\mathcal{P}\backslash\mathcal{B}_{i-1} = \{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} \subseteq \mathcal{P}^\dagger. \tag{6}$$

We sort the points $p \in \mathcal{P}^\dagger$ by decreasing values of $\Delta(p, \mathcal{C}_{i-1})$. Let p_t be the t-th point in this order for each $t \in [|\mathcal{P}^\dagger|]$, and define $\mathcal{P}_s^\dagger = \{p_t : t \in [\lfloor 2^{-s}|\mathcal{P}^\dagger|\rfloor]\}$ for each integer $s \in [0, \lfloor\log|\mathcal{P}^\dagger|\rfloor]$. Equality (6) implies the existence of an integer $\tilde{s} \in [0, \lfloor\log|\mathcal{P}^\dagger|\rfloor]$ satisfying $\mathcal{P}\backslash\mathcal{B}_{i-1} \subseteq \mathcal{P}_{\tilde{s}}^\dagger$ and $2|\mathcal{P}\backslash\mathcal{B}_{i-1}| \geq |\mathcal{P}_{\tilde{s}}^\dagger|$, and we have $\varepsilon^{-2}(580k+2m)|\mathcal{P}_i^f| > |\mathcal{P}_{\tilde{s}}^\dagger|$ due to Claim 15. Moreover, the operations performed in steps 12 and 13 of Algorithm 1 ensure that $(\mathcal{C}_{i-1}, \mathcal{P}_{\tilde{s}}^\dagger)$ is a descendant of $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$. This completes the proof of Lemma 14. ◀

Following the approach in Case (1), we consider a multi-set where each $p \in \mathcal{P}_i^n$ is replaced by the nearest center $c(p)$ in $\mathcal{C}_{i-1}$. We define the multi-set as $\mathcal{P}_i' = \{c(p) : p \in \mathcal{P}_i^n\} \cup \mathcal{P}_i^f$. Intuitively, we can closely simulate this multi-set using the union of a subset of $\{c(p) : p \in \mathcal{P}_i^n\}$ and a set of points sampled from $\mathcal{P}_i^f$, and the centroid of the simulated set is close to c_i^*, given that the squared distance from each $p \in \mathcal{P}_i^n$ to $c(p)$ is upper-bounded by a small value and sufficient points from $\mathcal{P}_i^f$ can be sampled. This motivates the following lemma.

▶ **Lemma 16.** *If $|\mathcal{P}_i^f| > \frac{\varepsilon}{17}|\mathcal{P}_i|$, then the following event happens with probability at least $\frac{1}{15}$: $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ has a descendant $(\mathcal{C}_{i-1} \uplus \{c_i\}, \mathcal{P}^\ddagger)$ with $\{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1} \uplus \{c_i\}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} \subseteq \mathcal{P}^\ddagger$ and $\Delta(\mathcal{P}_i, c_i) < (1 + \frac{\varepsilon}{2})\Delta(\mathcal{P}_i) + \frac{\varepsilon}{2k}\Delta(\mathbb{P})$.*

Proof. Denote by $(\mathcal{C}_{i-1}, \mathcal{P}^\ddagger)$ the descendant of $(\mathcal{C}_{i-1}, \mathcal{P}^\dagger)$ claimed in Lemma 14. When calling Algorithm 1 with $(\mathcal{C}_{i-1}, \mathcal{P}^\ddagger)$, we independently and uniformly sample N points from $\mathcal{P}^\ddagger$ and take M copies of each center from $\mathcal{C}_{i-1}$ to construct a multi-set $\mathcal{S}$. $(\mathcal{C}_{i-1}, \mathcal{P}^\ddagger)$ has a child $(\mathcal{C}_{i-1} \uplus \{c(\mathcal{S}')\}, \mathcal{P}^\ddagger)$ for each $\mathcal{S}' \subset \mathcal{S}$ with $|\mathcal{S}'| = M$. By Lemma 8 (with $t = N$ and $\lambda = \frac{1}{6}$) and the fact that $\mathcal{P}_i^f \subseteq \mathcal{P}\backslash\mathcal{B}_{i-1} \subseteq \mathcal{P}^\ddagger$ and $\varepsilon^{-2}(580k + 2m)|\mathcal{P}_i^f| > |\mathcal{P}^\ddagger|$ (due to Lemma 14), we know that $\mathcal{S}$ contains no less than M points uniformly distributed in $\mathcal{P}_i^f$ with probability at least $1 - e^{-\varepsilon^2 N/(72(580k+2m))} > \frac{1}{3}$. Moreover, the probability that $\mathcal{S}$ has a subset $\mathcal{S}'$ consisting of M points uniformly distributed in $\mathcal{P}_i'$ can be lower-bounded by the same constant, given that there are M copies of each distinct member of $\mathcal{P}_i'\backslash\mathcal{P}_i^f$ within $\mathcal{S}$. Under the assumption that such a subset $\mathcal{S}'$ exists, Lemma 4 (with $\lambda = \frac{4}{5}$) implies that inequality

$$||c_i - c(\mathcal{P}_i')||^2 \leq \frac{5\Delta(\mathcal{P}_i')}{4M|\mathcal{P}_i'|} = \frac{\varepsilon\Delta(\mathcal{P}_i')}{20|\mathcal{P}_i|} \tag{7}$$

holds with probability at least $\frac{1}{5}$, where $c_i = c(\mathcal{S}')$ is the centroid of $\mathcal{S}'$. Putting everything together, we know that $(\mathcal{C}_{i-1}, \mathcal{P}^\ddagger)$ has a child $(\mathcal{C}_{i-1} \uplus \{c_i\}, \mathcal{P}^\ddagger)$ satisfying inequality (7) with probability at least $\frac{1}{15}$.

We now show that $(\mathcal{C}_{i-1} \uplus \{c_i\}, \mathcal{P}^\ddagger)$ satisfies the properties claimed in Lemma 16. By a similar argument as in the proof of Claim 13, we can obtain the following upper bound on $\Delta(\mathcal{P}_i, c_i)$.

Algorithm 2 The Algorithm for Euclidean k-MeansOut.

```
Input: A constant $\varepsilon \in (0,1)$ and an instance $(\mathcal{P},k,m)$ of Euclidean $k$-MeansOut
       satisfying $\mathcal{P} \subset \mathbb{R}^d$
Output: A set $\mathcal{C} \subset \mathbb{R}^d$ of no more than $k$ centers and a set $\mathcal{O} \subseteq \mathcal{P}$ of no more than
        $m$ outliers
$\mathbb{C} \Leftarrow \emptyset$;
for $t \Leftarrow 1$ to $15^k$ do
    Sampling$(k, m, \varepsilon, \emptyset, \mathcal{P}, \mathbb{C})$;
for each $\mathcal{C}' \in \mathbb{C}$ do
    cost$(\mathcal{C}') \Leftarrow \min_{\mathcal{O}' \subseteq \mathcal{P} \wedge |\mathcal{O}'| \leq m} \sum_{p \in \mathcal{P} \setminus \mathcal{O}'} \Delta(p, \mathcal{C}')$;
$\mathcal{C} \Leftarrow \arg\min_{\mathcal{C}' \in \mathbb{C}}$ cost$(\mathcal{C}')$;
$\mathcal{O} \Leftarrow \arg\max_{\mathcal{O}' \subseteq \mathcal{P} \wedge |\mathcal{O}'| \leq m} \sum_{p \in \mathcal{O}'} \Delta(p, \mathcal{C})$;
return $\mathcal{C}, \mathcal{O}$.
```

▷ Claim 17. If $|\mathcal{P}_i^f| > \frac{\varepsilon}{17}|\mathcal{P}_i|$, then $\Delta(\mathcal{P}_i, c_i) < (1+\frac{\varepsilon}{5})\Delta(\mathcal{P}_i) + \frac{3\varepsilon}{10k}\Delta(\mathbb{P})$.

Observe that

$$\{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1} \uplus \{c_i\}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} \subseteq \{p \in \mathcal{P} : \Delta(p, \mathcal{C}_{i-1}) > \frac{\varepsilon\Delta(\mathbb{P})}{8k|\mathcal{P}_i|}\} = \mathcal{P} \setminus \mathcal{B}_{i-1} \subseteq \mathcal{P}^{\ddagger},$$

where the last step is due to Lemma 14. Combining this with Claim 17, we know that Lemma 16 is true. ◀

Lemma 12 and Lemma 16 suggest that for each $i \in \{2, \cdots, k\}$, $\tau(i)$ holds if $\tau(i-1)$ is true. Combining this with the initial condition $\tau(1)$ being true, we establish the validity of $\tau(i)$ for each $i \in [k]$. The proof of Lemma 10 effortlessly follows from the statement of $\tau(k)$.

4 Applications

In this section we show the applications of Algorithm 1. We first address the Euclidean k-MeansOut problem, and then show how to extend our approach to constrained cases.

4.1 The Algorithm for Euclidean k-MeansOut

Our approach for solving Euclidean k-MeansOut is presented in Algorithm 2, which takes as inputs a constant $\varepsilon \in (0,1)$ and an instance $\mathcal{I} = (\mathcal{P}, k, m)$ with $\mathcal{P} \subset \mathbb{R}^d$ and $|\mathcal{P}| = n$. The algorithm iteratively invokes Algorithm 1 to construct a collection of center sets and returns the one, along with the corresponding outlier set, that minimizes the cost for $\mathcal{I}$. By analyzing the performance of Algorithm 2, we complete the proof of Theorem 2.

Proof (of Theorem 2). Let $(\mathcal{C}^*, \mathcal{O}^*)$ be an optimal solution to $\mathcal{I}$, which opens a set $\mathcal{C}^* = \{c_1^*, \cdots, c_k^*\}$ of k centers from $\mathbb{R}^d$ and removes a set $\mathcal{O}^*$ of m outliers from $\mathcal{P}$. For each $i \in [k]$, denote by $\mathcal{P}_i^*$ the subset of the points in $\mathcal{P} \setminus \mathcal{O}^*$ whose closest center in $\mathcal{C}^*$ is c_i^*. Define $\Delta(\mathbb{P}) = \sum_{i=1}^{k} \Delta(\mathcal{P}_i^*, c_i^*) = \sum_{p \in \mathcal{P} \setminus \mathcal{O}^*} \Delta(p, \mathcal{C}^*)$ as the cost of $(\mathcal{C}^*, \mathcal{O}^*)$. Moreover, let $\mathbb{C}$ be the collection of center sets constructed by Algorithm 2, and let $(\mathcal{C}, \mathcal{O})$ be the solution to $\mathcal{I}$ returned by the algorithm. Observe that the statement in Lemma 10 holds with probability no less than 15^{-k}. Given that Algorithm 2 invokes **Sampling** 15^k times to construct $\mathbb{C}$, the probability of the statement in Lemma 10 being true in at least one of these invocations can be lower-bounded by $1 - (1 - 15^{-k})^{15^k} > 1 - e^{-1}$. Consequently, inequality

$$\sum_{p\in\mathcal{P}\setminus\mathcal{O}} \Delta(p,\mathcal{C}) \le \sum_{i=1}^{k} \min_{c\in\mathcal{C}} \Delta(\mathcal{P}_i^*, c) \le (1+\varepsilon)\Delta(\mathbb{P}) \tag{8}$$

holds with probability at least $1-e^{-1}$, where the first step is due to the operation in step 9 of Algorithm 2, and the second step follows from Lemma 10. Inequality (8) implies that the approximation ratio of Algorithm 2 is $1+\varepsilon$.

It remains to analyze the running time of Algorithm 2. Lemma 9 implies that invoking **Sampling** 15^k times takes $nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time and adds $n((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ center sets to $\mathbb{C}$. For each center set from $\mathbb{C}$, the algorithm takes $O(ndk)$ time to compute the corresponding cost for $\mathcal{I}$. Consequently, we know that Algorithm 2 runs in $n^2d((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time. This completes the proof of Theorem 2. ◀

4.2 A Coreset-Based Accelerated Algorithm

In a recent study, Huang et al. [29] showed that a subset of $\text{poly}(m,k,\varepsilon^{-1})$ weighted points, referred to as a coreset, can be identified in linear time for the given instance of Euclidean k-MeansOut, such that the outlier-removal clustering costs, induced by any set of k centers, are similar between the coreset and the entire point set. Such a method for coreset construction can serve as a means to accelerate our algorithm. Specifically, when selecting a well-performing center set from the collection $\mathbb{C}$, we can compare the clustering costs induced by the center sets on the coreset rather than the entire point set. This reduces the running time of the algorithm in Theorem 2 to $nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$, with an arbitrarily small loss in the approximation ratio.

▶ **Lemma 18** ([29]). *Given a constant $\varepsilon\in(0,1)$, an instance $\mathcal{I}=(\mathcal{P},k,m)$ of Euclidean k-MeansOut satisfying $\mathcal{P}\subset\mathbb{R}^d$ and $|\mathcal{P}|=n$, and an algorithm that has the guarantee of yielding a solution $(\mathcal{C},\mathcal{O})$ satisfying $\sum_{p\in\mathcal{P}\setminus\mathcal{O}}\Delta(p,\mathcal{C})\le\alpha\cdot opt$, $|\mathcal{C}|\le\beta k$, and $|\mathcal{O}|\le\gamma m$ for three real numbers $\alpha,\beta,\gamma\ge1$ in $T(n,d,k,m)$ time (opt is the cost of an optimal solution to $\mathcal{I}$), a weighted subset $\mathcal{S}\subseteq\mathcal{P}$ satisfying $|\mathcal{S}|\le\gamma m+\beta(k\varepsilon^{-1})^{O(1)}$ with weight function $w:\mathcal{S}\to[0,+\infty)$ can be constructed in $T(n,d,k,m)+O(ndk)$ time, such that*

$$\min_{\mathcal{O}'\subseteq\mathcal{S}\wedge\sum_{p\in\mathcal{O}'}w(p)\le m}\ \sum_{p\in\mathcal{S}\setminus\mathcal{O}'} w(p)\Delta(p,\mathcal{C}') \in (1\pm\alpha\varepsilon) \min_{\mathcal{O}'\subseteq\mathcal{P}\wedge|\mathcal{O}'|\le m}\ \sum_{p\in\mathcal{P}\setminus\mathcal{O}'} \Delta(p,\mathcal{C}')$$

for each $\mathcal{C}'\subset\mathbb{R}^d$ with $|\mathcal{C}'|=k$.

To leverage Lemma 18, we give a straightforward and fast bi-criteria approximation algorithm for Euclidean k-MeansOut, based on the D^2-sampling method given by Arthur and Vassilvitskii [2].

▶ **Lemma 19.** *Given an instance $\mathcal{I}=(\mathcal{P},k,m)$ of Euclidean k-MeansOut with $\mathcal{P}\subset\mathbb{R}^d$ and $|\mathcal{P}|=n$, a center set $\mathcal{C}\subset\mathbb{R}^d$ satisfying $|\mathcal{C}|=O(k+m)$ and $\sum_{p\in\mathcal{P}}\Delta(p,\mathcal{C})\le O(opt)$ can be identified in $O(nd(k+m))$ time, where opt is the cost of an optimal solution to $\mathcal{I}$.*

Taking as an input the algorithm given in Lemma 19, Lemma 18 yields a coreset of size $((k+m)\varepsilon^{-1})^{O(1)}$ in $O(nd(k+m))$ time. Leveraging this coreset allows us to reduce the running time of our algorithm for Euclidean k-MeansOut to be linear in n, as described in the following theorem.

▶ **Theorem 20.** *Given a constant $\varepsilon \in (0,1)$ and an instance $(\mathcal{P}, k, m)$ of Euclidean k-*MeansOut *satisfying $\mathcal{P} \subset \mathbb{R}^d$ and $|\mathcal{P}| = n$, there is a $(1+O(\varepsilon))$-approximation algorithm running in $nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time.*

Proof. Let $\mathcal{S} \subseteq \mathcal{P}$ be the coreset constructed by Lemma 18 when taking as inputs constant ε, instance $(\mathcal{P}, k, m)$, and the algorithm in Lemma 19, and let $w : \mathcal{S} \to [0, +\infty)$ be the corresponding weight function. We have $|\mathcal{S}| \le ((k+m)\varepsilon^{-1})^{O(1)}$ due to Lemma 18 and Lemma 19. Our accelerated algorithm for Euclidean k-MeansOut is identical to Algorithm 2, differing only in steps 6 and 8, where we calculate the outlier-removal clustering costs induced by the candidate center sets and select the one with minimum cost. We now define

$$\texttt{wcost}(\mathcal{C}') = \min_{\mathcal{O}' \subseteq \mathcal{S} \wedge \sum_{p \in \mathcal{O}'} w(p) \le m} \sum_{p \in \mathcal{S} \setminus \mathcal{O}'} w(p) \Delta(p, \mathcal{C}')$$

for each $\mathcal{C}' \in \mathbb{C}$ in step 6 of the algorithm, and select the center set $\mathcal{C} \in \mathbb{C}$ with minimum value of $\texttt{wcost}(\mathcal{C})$ in step 8.

Observe that calculating $\texttt{wcost}(\mathcal{C}')$ for each $\mathcal{C}' \in \mathbb{C}$ takes $O(|\mathcal{S}||\mathcal{C}'|d) \le d((k+m)\varepsilon^{-1})^{O(1)}$ time. Combining this with the fact that we take $nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time to construct a collection $\mathbb{C}$ of $n((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ candidate center sets (as argued in the proof of Theorem 2), we know that the accelerated algorithm runs in $nd((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time.

The analysis of the approximation ratio of our accelerated algorithm follows that of Algorithm 2 (given in the proof of Theorem 2). Denote by $(\tilde{\mathcal{C}}, \tilde{\mathcal{O}})$ the solution to $\mathcal{I}$ constructed by the accelerated algorithm, and let $(\mathcal{C}, \mathcal{O})$ be a solution to $\mathcal{I}$ satisfying inequality (8). We have

$$\sum_{p \in \mathcal{P} \setminus \tilde{\mathcal{O}}} \Delta(p, \tilde{\mathcal{C}}) \le \frac{\texttt{wcost}(\tilde{\mathcal{C}})}{1 - O(\varepsilon)} \le \frac{\texttt{wcost}(\mathcal{C})}{1 - O(\varepsilon)} \le \frac{1 + O(\varepsilon)}{1 - O(\varepsilon)} \sum_{p \in \mathcal{P} \setminus \mathcal{O}} \Delta(p, \mathcal{C}), \tag{9}$$

where the first and third steps follow from the approximation guarantee of the coreset given by Lemma 18 and Lemma 19, and the second step is due to the fact that the center set $\tilde{\mathcal{C}}$ selected by our accelerated algorithm satisfies $\texttt{wcost}(\tilde{\mathcal{C}}) = \min_{\mathcal{C}' \in \mathbb{C}} \texttt{wcost}(\mathcal{C}')$. Combining inequality (9) with inequality (8), we know that the approximation ratio of the accelerated algorithm is $1 + O(\varepsilon)$. ◀

5 Extensions to Constrained Cases

Constrained k-MeansOut problems generalize k-MeansOut by introducing additional constraints on the feasibilities of solutions. For example, in the capacitated generalization, there is an upper bound on the size of each cluster. Similarly, in the lower-bounded generalization, each cluster must contain at least a specified number of points. There are known frameworks for reducing constrained k-MeansOut problems to their outlier-free counterparts with small losses in the approximation ratios [32, 13]. Combined with the approximation schemes applicable in outlier-free cases, such as the ones given in [8, 16, 4], these frameworks enable the development of $(1+\varepsilon)$-approximation algorithms for constrained k-MeansOut problems. However, similar to the unconstrained case, the reductions given in [32, 13] require time exponentially dependent on m.

An apparent distinction between constrained k-MeansOut problems and the unconstrained counterpart lies in the locality property described in Section 1.1: In the former, clusters in an optimal solution can be quite different from the ones with minimum clustering cost, as additional constraints are imposed on their feasibilities. Fortunately, Lemma 10 implies that a near-optimal set of centers corresponding to any outlier-removal k-clustering result can be found by Algorithm 1, including those imposed with additional constraints. Given k good-enough centers, it is sufficient to remove the outliers and assign each point to a center under the additional constraints. Indeed, combining Algorithm 1 with problem-specific methods for identifying outliers and assigning points, we obtain the first PTASs for constrained k-MeansOut problems in Euclidean spaces for constant k and super-constant m, including capacitated and fair k-MeansOut.

Our algorithms for constrained k-MeansOut problems closely resemble Algorithm 2, with the only difference being in the construction of the solution based on $\mathbb{C}$ (as shown in steps 5-10). Given a set $\mathcal{P} \subset \mathbb{R}^d$ with $|\mathcal{P}| = n$, a real number $\varepsilon \in (0, 1]$, and two positive integers k and m, let $\mathbb{C}$ be the collection of center sets constructed by Algorithm 2. By Lemma 10 and the fact that $\mathbb{C}$ is constructed by invoking **Sampling** 15^k times, we know that the following event happens with probability no less than $1 - (1 - 15^{-k})^{15^k} > 1 - e^{-1}$: For any k disjoint subsets $\mathcal{P}_1, \cdots, \mathcal{P}_k$ of $\mathcal{P}$ with $\sum_{i=1}^{k} |\mathcal{P}_i| = n - m$, there is a center set $\mathcal{C} \in \mathbb{C}$ satisfying

$$\sum_{i=1}^{k} \min_{c \in \mathcal{C}} \Delta(\mathcal{P}_i, c) \leq (1 + \varepsilon) \sum_{i=1}^{k} \Delta(\mathcal{P}_i). \tag{10}$$

This implies a good chance of finding a near-optimal set of centers corresponding to any outlier-removal k-clustering result in $\mathbb{C}$. The next step involves developing problem-specific methods to remove m outliers and partition the remaining points into k clusters based on the given center set, with the objective of minimizing the clustering cost (defined as the sum of the squared distances from the points to the corresponding centers) while satisfying the specified additional constraints. If we can remove the outliers and partition the points in an optimal way, then the guarantee of the center set given in inequality (10) suggests the achievement of a $(1 + \varepsilon)$-approximation solution.

5.1 The Algorithm for Capacitated k-MeansOut

Capacitated clustering is one of the most extensively studied generalizations of the standard clustering formulation, which imposes an upper-bound constraint on the size of each cluster. An Euclidean instance of capacitated k-MeansOut is specified by a set $\mathcal{P} \subset \mathbb{R}^d$ of n points, two positive integers k and m, and a *capacity* $u \geq 1$. A feasible solution to the instance selects a set $\mathcal{C} \subset \mathbb{R}^d$ of centers and a set $\mathcal{O} \subseteq \mathcal{P}$ of outliers, and assigns each point $p \in \mathcal{P} \backslash \mathcal{O}$ to a center $\varphi(p) \in \mathcal{C}$, such that $|\mathcal{C}| \leq k$, $|\mathcal{O}| \leq m$, and $|\varphi^{-1}(c)| \leq u$ for each $c \in \mathcal{C}$. The cost of such a solution is $\sum_{p \in \mathcal{P} \backslash \mathcal{O}} ||p - \varphi(p)||^2$. The goal of the problem is to find a feasible solution with minimum cost.

Motivated by the method for solving outlier-free constrained problems given in [16], we reduce the task of identifying outliers and assigning points for capacitated k-MeansOut to the well-known *minimum-cost circulation* problem [40], which can be defined as follows.

▶ **Definition 21** (minimum-cost circulation). *An instance of the minimum-cost circulation problem is specified by a directed graph $G(\mathcal{V}, \mathcal{A})$ with a set $\mathcal{V}$ of vertices and a set $\mathcal{A}$ of arcs, where each $(v, w) \in \mathcal{A}$ has a cost $\Delta(v, w) \geq 0$, a capacity $u(v, w) \geq 0$, and a demand $l(v, w) \in [0, u(v, w)]$. A feasible solution to the instance associates each $(v, w) \in \mathcal{A}$ with a flow $f(v, w) \in [l(v, w), u(v, w)]$ such that $\sum_{w:(v,w) \in \mathcal{A}} f(v, w) = \sum_{w:(w,v) \in \mathcal{A}} f(w, v)$ for each $v \in \mathcal{V}$. The cost of this solution is $\sum_{(v,w) \in \mathcal{A}} \Delta(v, w) f(v, w)$. The problem aims to find a feasible solution with minimum cost.*

Let $\mathcal{I} = (\mathcal{P}, k, m, u)$ be an Euclidean instance of capacitated k-MeansOut. As previously discussed, Algorithm 2 can construct a collection $\mathbb{C}$ of center sets, including a near-optimal one for $\mathcal{I}$, with high probability. For each $\mathcal{C} \in \mathbb{C}$, we construct an instance of minimum-cost circulation as follows.

- The vertex set $\mathcal{V}$ consists of the points in $\mathcal{P}$, the centers in $\mathcal{C}$, and three additional vertices v_1, v_2, and v_3.
- There is an arc $(v_3, v_1) \in \mathcal{A}$ with $\Delta(v_3, v_1) = 0$ and $u(v_3, v_1) = l(v_3, v_1) = |\mathcal{P}|$.
- For each $p \in \mathcal{P}$ and $c \in \mathcal{C}$, there is an arc $(p, c) \in \mathcal{A}$ with $\Delta(p, c) = ||p - c||^2$, $u(p, c) = 1$, and $l(p, c) = 0$. Here, $f(p, c) = 1$ signifies the assignment of point p to center c (i.e., $\varphi(c) = p$).
- For each $p \in \mathcal{P}$, there is an arc $(p, v_2) \in \mathcal{A}$ with $\Delta(p, v_2) = 0$, $u(p, v_2) = 1$, and $l(p, v_2) = 0$. A point p with $f(p, v_2) = 1$ is identified as an outlier. To ensure that the outliers are no more than m (more formally, $\sum_{p \in \mathcal{P}} f(p, v_2) \leq m$), we add to $\mathcal{A}$ an arc (v_2, v_3) with $\Delta(v_2, v_3) = 0$, $u(v_2, v_3) = m$, and $l(v_2, v_3) = 0$.
- To ensure that each $p \in \mathcal{P}$ is assigned to a center in $\mathcal{C}$ or identified as an outlier (i.e., $\sum_{c \in \mathcal{C} \cup \{v_2\}} f(p, c) = 1$), we add to $\mathcal{A}$ an arc (v_1, p) with $\Delta(v_1, p) = 0$ and $u(v_1, p) = l(v_1, p) = 1$.
- To satisfy the capacity constraint imposed on each $c \in \mathcal{C}$ (i.e., $\sum_{p \in \mathcal{P}} f(p, c) \leq u$), we add to $\mathcal{A}$ an arc (c, v_3) with $\Delta(c, v_3) = 0$, $u(c, v_3) = \lfloor u \rfloor$, and $l(c, v_3) = 0$.

Given that all the capacities and demands in the instance of minimum-cost circulation described above are integers, its optimal integral solutions can be found in $(nk)^{O(1)}$ time [40]. It can be seen that these solutions correspond to optimal ways of identifying outliers and assigning points for the given center set.

Let $(\mathcal{C}^*, \mathcal{O}^*, \varphi^*)$ be an optimal solution to $\mathcal{I}$, where $\mathcal{C}^* = \{c_1^*, \cdots, c_k^*\}$. For each $i \in [k]$, define $\mathcal{P}_i^* = \{p \in \mathcal{P} \backslash \mathcal{O}^* : \varphi^*(p) = c_i^*\}$. Inequality (10) implies that there is a center set $\mathcal{C} \in \mathbb{C}$ satisfying

$$\sum_{i=1}^{k} \min_{c \in \mathcal{C}} \Delta(\mathcal{P}_i^*, c) \leq (1 + \varepsilon) \sum_{i=1}^{k} \Delta(\mathcal{P}_i^*) = (1 + \varepsilon) \sum_{p \in \mathcal{P} \backslash \mathcal{O}^*} ||p - \varphi^*(p)||^2 \tag{11}$$

with constant probability. Based on an optimal integral solution to the instance of minimum-cost circulation corresponding to such a center set $\mathcal{C}$, we can construct a solution $(\mathcal{C}, \mathcal{O}, \varphi)$ to $\mathcal{I}$ satisfying

$$\sum_{p \in \mathcal{P} \backslash \mathcal{O}} ||p - \varphi(p)||^2 \leq \sum_{i=1}^{k} \min_{c \in \mathcal{C}} \Delta(\mathcal{P}_i^*, c). \tag{12}$$

Inequalities (11) and (12) imply that a $(1 + \varepsilon)$-approximation solution to $\mathcal{I}$ has been constructed. Recall that $\mathbb{C}$ is of size $n((k + m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ and can be constructed in $nd((k + m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time (due to Lemma 9). Combining this with the fact that the instance of minimum-cost circulation corresponding to each center set in $\mathbb{C}$ can be solved in $(nk)^{O(1)}$ time, we know that constructing the $(1 + \varepsilon)$-approximation solution takes $n^{O(1)} d((k + m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time.

▶ **Theorem 22.** *Given a constant $\varepsilon \in (0, 1)$ and an Euclidean instance $(\mathcal{P}, k, m, u)$ of capacitated k-MeansOut satisfying $\mathcal{P} \subset \mathbb{R}^d$ and $|\mathcal{P}| = n$, there is a $(1 + \varepsilon)$-approximation algorithm running in $n^{O(1)} d((k + m)\varepsilon^{-1})^{O(k\varepsilon^{-1})}$ time.*

5.2 The Algorithm for Fair k-MeansOut

Motivated by applications related to fair data representation [5], fair clustering problems, which impose constraints on the proportions of each type of points within the clusters, have garnered significant attention. We consider the fair k-MeansOut problem defined in [13]. An instance of the problem is specified by a collection $\mathbb{P} = \{\mathcal{P}^1, \cdots, \mathcal{P}^\ell\}$ of ℓ disjoint sets of points in $\mathbb{R}^d$, a positive integer k, two fairness vectors $\vec{\alpha}, \vec{\beta} \in [0,1]^\ell$, and an outlier vector $\vec{m} \in \mathbb{N}^\ell$. A feasible solution to this instance selects a set $\mathcal{C} \subset \mathbb{R}^d$ of no more than k centers and a set $\mathcal{O} \subseteq \bigcup_{i=1}^{\ell} \mathcal{P}^i$ of outliers, and assigns each point $p \in \bigcup_{i=1}^{\ell} \mathcal{P}^i \backslash \mathcal{O}$ to a center $\varphi(p) \in \mathcal{C}$, such that $|\mathcal{P}^i \cap \varphi^{-1}(c)||\varphi^{-1}(c)|^{-1} \in [\alpha_i, \beta_i]$ and $|\mathcal{P}^i \cap \mathcal{O}| \leq m_i$ for each $i \in [\ell]$ and $c \in \mathcal{C}$. The cost of the solution is $\sum_{i=1}^{\ell} \sum_{p \in \mathcal{P}^i} ||p - \varphi(p)||^2$.

Similar to our previous strategy, we address the fair k-MeansOut problem based on a collection of center sets constructed by repeatedly invoking Algorithm 1. Unfortunately, identifying outliers and assigning points for a given set of centers in an optimal way within polynomial time is no longer feasible. Specifically, a reduction from the 3*D-matching* problem suggests that this task is NP-hard [5]. Using the mixed-integer linear programming-based algorithm given in [13], this task can be completed in an exponential time.

▶ **Lemma 23** ([13]). *Given a set $\mathcal{C} \subset \mathbb{R}^d$ of no more than k centers and an instance $\mathcal{I} = (\mathbb{P}, k, \vec{\alpha}, \vec{\beta}, \vec{m})$ of fair k-MeansOut satisfying $|\mathbb{P}| = \ell$, $\sum_{\mathcal{P} \in \mathbb{P}} |\mathcal{P}| = n$, $\sum_{i=1}^{\ell} m_i = m$, and $\mathcal{P} \subset \mathbb{R}^d$ for each $\mathcal{P} \in \mathbb{P}$, a feasible solution to $\mathcal{I}$ with minimum cost among the ones taking $\mathcal{C}$ as the center set can be constructed in $(k\ell)^{O(k\ell)} n^{O(1)} dL$ time, where L is the bit-size of $\mathcal{I}$.*

Using our sampling-based algorithm for constructing center sets (m is replaced with $\sum_{i=1}^{\ell} m_i$) and the algorithm for constructing solutions given in Lemma 23, we obtain the following approximation scheme for fair k-MeansOut.

▶ **Theorem 24.** *Given a constant $\varepsilon \in (0,1)$ and an instance $\mathcal{I} = (\mathbb{P}, k, \vec{\alpha}, \vec{\beta}, \vec{m})$ of fair k-MeansOut satisfying $|\mathbb{P}| = \ell$, $\sum_{\mathcal{P} \in \mathbb{P}} |\mathcal{P}| = n$, $\sum_{i=1}^{\ell} m_i = m$, and $\mathcal{P} \subset \mathbb{R}^d$ for each $\mathcal{P} \in \mathbb{P}$, there is a $(1+\varepsilon)$-approximation algorithm running in $n^{O(1)} d((k+m)\varepsilon^{-1})^{O(k\varepsilon^{-1})} (k\ell)^{O(k\ell)} L$ time, where L is the bit-size of $\mathcal{I}$.*

6 Conclusions

In this paper, we present $(1+\varepsilon)$-approximation algorithms with running times exponential in k for Euclidean k-MeansOut and constrained generalizations of the problem, including capacitated and fair k-MeansOut. For each considered problem, our proposed algorithm stands for the first PTAS for constant k.

Considering the APX-hardness of Euclidean k-MeansOut [3], it is unlikely to design a $(1+\varepsilon)$-approximation algorithm without exponential dependence on k in high dimensions. Nonetheless, exploring ways to improve the running time of our algorithm for Euclidean k-MeansOut remains an interesting direction for future research. Especially, one can see whether it is possible to develop a $(1+\varepsilon)$-approximation algorithm running in $(ndm)^{O(1)} f(k, \varepsilon)$ time for some positive function f.

References

1 Akanksha Agrawal, Tanmay Inamdar, Saket Saurabh, and Jie Xue. Clustering what matters: Optimal approximation for clustering with outliers. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 6666–6674, 2023.

2 David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

3 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*, volume 34, pages 754–767, 2015.

4 Sayan Bandyapadhyay, Fedor V. Fomin, and Kirill Simonov. On coresets for fair clustering in metric and Euclidean spaces and their applications. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming*, volume 198, pages 23:1–23:15, 2021.

5 Suman Kalyan Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems*, pages 4955–4966, 2019.

6 Aditya Bhaskara, Sharvaree Vadgama, and Hong Xu. Greedy sampling for approximate clustering in the presence of outliers. In *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems*, pages 11146–11155, 2019.

7 Anup Bhattacharya, Dishant Goyal, Ragesh Jaiswal, and Amit Kumar. On sampling based algorithms for k-means. In *Proceedings of the 40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 182, pages 13:1–13:17, 2020.

8 Anup Bhattacharya, Ragesh Jaiswal, and Amit Kumar. Faster algorithms for the constrained k-means problem. *Theory Comput. Syst.*, 62(1):93–115, 2018.

9 Sanjay Chawla and Aristides Gionis. k-means--: A unified approach to clustering and outlier detection. In *Proceedings of the 13th SIAM International Conference on Data Mining*, pages 189–197, 2013.

10 Jiecao Chen, Erfan Sadeqi Azer, and Qin Zhang. A practical algorithm for distributed clustering and outlier detection. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems*, pages 2253–2262, 2018.

11 Ke Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM J. Comput.*, 39(3):923–947, 2009.

12 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. *J. ACM*, 68(6):44:1–44:34, 2021.

13 Rajni Dabas, Neelima Gupta, and Tanmay Inamdar. FPT approximations for capacitated/fair clustering with outliers. *CoRR*, abs/2305.01471, 2023.

14 Wenceslas Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation schemes for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 50–58. ACM, 2003.

15 Amit Deshpande, Praneeth Kacham, and Rameshwar Pratap. Robust k-means++. In *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence*, volume 124, pages 799–808, 2020.

16 Hu Ding and Jinhui Xu. A unified framework for clustering constrained data without locality property. *Algorithmica*, 82(4):808–852, 2020.

17 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 569–578, 2011.

18 Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A PTAS for k-means clustering based on weak coresets. In *Proceedings of the 23rd ACM Symposium on Computational Geometry*, pages 11–18, 2007.

19 Dan Feldman and Leonard J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1343–1354, 2012.

20 Zachary Friggstad, Kamyar Khodamoradi, Mohsen Rezapour, and Mohammad R. Salavatipour. Approximation schemes for clustering with outliers. *ACM Trans. Algorithms*, 15(2):26:1–26:26, 2019.

21 Luis Ángel García-Escudero and Alfonso Gordaliza. Robustness properties of k-means and trimmed k-means. *J. Am. Stat. Assoc.*, 94(447):956–969, 1999.

22 Alexandros Georgogiannis. Robust k-means: A theoretical revisit. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems*, pages 2883–2891, 2016.

23 Christoph Grunau and Václav Rozhon. Adapting k-means algorithms for outliers. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 7845–7886, 2022.

24 Sudipto Guha, Yi Li, and Qin Zhang. Distributed partial clustering. *ACM Trans. Parallel Comput.*, 6(3):11:1–11:20, 2019.

25 Shalmoli Gupta, Ravi Kumar, Kefu Lu, Benjamin Moseley, and Sergei Vassilvitskii. Local search methods for k-means with outliers. *Proc. VLDB Endow.*, 10(7):757–768, 2017.

26 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.*, 58:13–30, 1963.

27 Junyu Huang, Qilong Feng, Ziyun Huang, Jinhui Xu, and Jianxin Wang. Fast algorithms for distributed k-clustering with outliers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 13845–13868, 2023.

28 Lingxiao Huang, Shaofeng H.-C. Jiang, Jian Li, and Xuan Wu. ϵ-coresets for clustering (with outliers) in doubling metrics. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 814–825, 2018.

29 Lingxiao Huang, Shaofeng H.-C. Jiang, Jianing Lou, and Xuan Wu. Near-optimal coresets for robust clustering. In *Proceedings of the 11th International Conference on Learning Representations*, 2023.

30 Sungjin Im, Mahshid Montazer Qaem, Benjamin Moseley, Xiaorui Sun, and Rudy Zhou. Fast noise removal for k-means clustering. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, volume 108, pages 456–466, 2020.

31 Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering (extended abstract). In *Proceedings of the 10th Annual Symposium on Computational Geometry*, pages 332–339, 1994.

32 Ragesh Jaiswal and Amit Kumar. Clustering what matters in constrained settings: Improved outlier to outlier-free reductions. In *Proceedings of the 34th International Symposium on Algorithms and Computation*, volume 283, pages 41:1–41:16, 2023.

33 Ragesh Jaiswal, Amit Kumar, and Sandeep Sen. A simple D^2-sampling based PTAS for k-means and other clustering problems. *Algorithmica*, 70(1):22–46, 2014.

34 Ragesh Jaiswal, Mehul Kumar, and Pulkit Yadav. Improved analysis of D^2-sampling based PTAS for k-means and other clustering problems. *Inf. Process. Lett.*, 115(2):100–103, 2015.

35 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004.

36 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 646–659, 2018.

37 Amit Kumar, Yogish Sabharwal, and Sandeep Sen. Linear-time approximation schemes for clustering problems in any dimensions. *J. ACM*, 57(2):5:1–5:32, 2010.

38 Euiwoong Lee, Melanie Schmidt, and John Wright. Improved and simplified inapproximability for k-means. *Inf. Process. Lett.*, 120:40–43, 2017.

39 Shi Li and Xiangyu Guo. Distributed k-clustering for data with heavy noise. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems*, pages 7849–7857, 2018.

40 Alexander Schrijver. *Combinatorial optimization: Polyhedra and efficiency.* Springer, Berlin, 2003.

41 Zhen Zhang, Qilong Feng, Junyu Huang, Yutian Guo, Jinhui Xu, and Jianxin Wang. A local search algorithm for k-means with outliers. *Neurocomputing*, 450:230–241, 2021.

Approximate Suffix-Prefix Dictionary Queries

Wiktor Zuba
CWI, Amsterdam, The Netherlands

Grigorios Loukides
King's College London, UK

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Sharma V. Thankachan
North Carolina State University, Raleigh, NC, USA

Abstract

In the *all-pairs suffix-prefix* (APSP) problem [Gusfield et al., *Inf. Process. Lett.* 1992], we are given a dictionary R of r strings, $S_1, \dots, S_r$, of total length n, and we are asked to find the length $\mathsf{SPL}_{i,j}$ of the *longest* string that is both a suffix of S_i and a prefix of S_j, for all $i, j \in [1\mathinner{.\,.} r]$. APSP is a classic problem in string algorithms with applications in bioinformatics, especially in sequence assembly. Since $r = |R|$ is typically very large in real-world applications, considering all r^2 pairs of strings explicitly is prohibitive. This is when the data structure variant of APSP makes sense; in the same spirit as distance oracles computing shortest paths between any two vertices given online.

We show how to quickly locate *k-approximate matches* (under the Hamming or the edit distance) in R using a version of the k-errata tree [Cole et al., STOC 2004] that we introduce. Let $\mathsf{SPL}^k_{i,j}$ be the length of the longest suffix of S_i that is at distance at most k from a prefix of S_j. In particular, for any $k = \mathcal{O}(1)$, we show an $\mathcal{O}(n \log^k n)$-sized data structure to support the following queries:

- $\mathsf{One\text{-}to\text{-}One}^k(i, j)$: output $\mathsf{SPL}^k_{i,j}$ in $\mathcal{O}(\log^k n \log\log n)$ time.
- $\mathsf{Report}^k(i, d)$: output all $j \in [1\mathinner{.\,.} r]$, such that $\mathsf{SPL}^k_{i,j} \geq d$, in $\mathcal{O}(\log^k n(\log n/\log\log n + \mathsf{output}))$ time, where output denotes the size of the output.

In fact, our algorithms work for any value of k not just for $k = \mathcal{O}(1)$, but the formulas bounding the complexities get much more complicated for larger values of k.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases all-pairs suffix-prefix, suffix-prefix queries, suffix tree, k-errata tree

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.85

Funding *Wiktor Zuba*: Received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement Grant Agreement No 101034253.
Solon P. Pissis: Supported in part by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.
Sharma V. Thankachan: Supported by the U.S. National Science Foundation (NSF) grant CCF-2316691.

1 Introduction

Given a dictionary R of r strings, $S_1, \dots, S_r$, of total length n, the *all-pairs suffix-prefix* (APSP) problem asks us to find, for each string S_i, $i \in [1\mathinner{.\,.} r]$, its longest suffix that is a prefix of string S_j, for all $j \neq i$, $j \in [1\mathinner{.\,.} r]$. APSP is a classic problem in string algorithms [18] with numerous applications in sequence assembly [18, 26, 31, 7, 10]. Gusfield et al. [19] presented

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 85; pp. 85:1–85:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an algorithm running in the optimal $\mathcal{O}(n + r^2)$ time for solving APSP, assuming all strings in R are over an integer alphabet of size $\sigma = n^{\mathcal{O}(1)}$. Many other optimal algorithms exist for APSP aiming at a smaller memory footprint [27, 36, 24].

DNA sequences (fragments) read by sequencing machines are typically assembled by first computing the maximal overlap between every pair of sequences. The size r of dictionary R in this case is very large. In particular, r^2 usually dominates n, and thus the r^2 factor is the bottleneck both in the time and the space complexity of assembly applications. For instance, in typical benchmark datasets for genome assembly using short DNA fragments, r is in the order of 10^6 to 10^8 and n is in the order of 10^8 to 10^{10}. Hence r^2 dominates n significantly.

In [25], we initiated a study on several data structure variants of APSP. Let $\mathsf{SPL}_{i,j}$, for any $i, j \in [1\mathinner{.\,.} r]$, denote the length of the *longest* suffix of S_i that is a prefix of S_j. We proposed $\mathcal{O}(n)$-sized data structures to support the following queries:

- One-to-One(i, j): output $\mathsf{SPL}_{i,j}$ in $\mathcal{O}(\log \log r)$ time.
- Report(i, d): output all distinct $j \in [1\mathinner{.\,.} r]$ such that $\mathsf{SPL}_{i,j} \geq d$, for some $d \in \mathbb{N}$, in $\mathcal{O}(\log n/ \log \log n + \mathsf{output})$ time, where output denotes the size of the output.

We also proposed other $\mathcal{O}(n)$-sized data structures for answering One-to-All(i), Count(i, ℓ), and Top(i, K) queries efficiently (see [25] for the precise definitions of these queries).

This framework is interesting both from a practical and from a theoretical perspective. By being able to answer different types of such queries efficiently, practitioners may be able to design alternative algorithms that *avoid the r^2 factor* in their time or space complexity; see the recent work of Talera et al. [33], which investigates practical aspects of our framework. Furthermore, the underlying data structure problems are also appealing from a theoretical perspective: (i) they are analogous to *distance oracles* for networks [35, 28, 14, 13, 11, 17]; and (ii) they are special types of *internal pattern matching* data structures [21, 20, 2, 1, 12, 4].

In this work, we make the next natural yet *challenging* step, that is, extend the framework we introduced in [25] to support *k-approximate suffix-prefix* queries. Indeed, these are the most useful queries in real-world applications. Approximate variants of APSP, under the Hamming or edit distance, have been considered in the literature. Barton et al. [5] showed how to solve the problem for two strings ($r = 2$), under Hamming or edit distance k, in $\mathcal{O}(nk)$ time. Thankachan et al. [34] showed how to solve the problem in $\mathcal{O}((n+r^2) \log^k n)$ time under Hamming or edit distance k, for any r and $k = \mathcal{O}(1)$. Many other proposed methods aimed at practical efficiency and are thus based on filtering strategies [29, 37, 22]. They typically use a two-step approach: candidate regions are identified that potentially correspond to sought matches; and those candidates are checked to actually verify the desired matching condition. These approaches do not yield strong theoretical time bounds but they are very efficient in practice. In any case, none of the proposed methods addresses the data structure variant.

Problems Statements. Let us start with the following basic definition.

▶ **Definition 1.** *For any given set $R = \{S_1, \ldots, S_r\}$ of r strings, we define the* longest suffix-prefix of S_i and S_j with k-errors *as the longest suffix U_i of S_i that is at Hamming/edit distance at most k from some prefix V_j of S_j. We denote the length of U_i by $\mathsf{SPL}^k_{i,j}$.*

By fixing k *apriori*, we want to efficiently compute the values $\mathsf{SPL}^k_{i,j}$ for certain (S_i, S_j) pairs given in an online fashion. In particular, we consider the following queries:

- One-to-One$^k(i, j)$: output $\mathsf{SPL}^k_{i,j}$.
- Report$^k(i, d)$: output all distinct $j \in [1\mathinner{.\,.} r]$ such that $\mathsf{SPL}^k_{i,j} \geq d$, for some $d \in \mathbb{N}$.

The offline version (that is, the k-approximate variant of APSP), which we denote by APSP^k, is defined as follows: output $\mathsf{SPL}^k_{i,j}$, for all $1 \le i, j \le r$. Recall that APSP^k has already been solved by Thankachan et al. [34] in $\mathcal{O}((n + r^2)\log^k n)$ time for any $k = \mathcal{O}(1)$.

▶ **Observation 2.** *For Hamming distance we always have that $|U_i| = SPL^k_{i,j} = |V_i|$. However, for edit distance it does not need to be the case: we cannot always maximize the lengths of both suffix and prefix simultaneously. For $S_i = \mathtt{xx}\cdots\mathtt{xabcd}, S_j = \mathtt{bcdeyy}\cdots\mathtt{y}$, we have $SPL^1_{i,j} = 4$ as $\mathtt{abcd}$ is at edit distance 1 from $\mathtt{bcd}$, but $\mathtt{bcd}$ is at edit distance 1 from $\mathtt{bcde}$, and $\mathtt{abcd}$ is at edit distance 2 from $\mathtt{bcde}$. In this sense, edit distance is a bit more complicated.*

Results and Paper Organization. We assume the standard word-RAM model of computations with word size $w = \Omega(\log n)$, where n is the input size. We present our results below for any $k = \mathcal{O}(1)$; our algorithms work for any arbitrary value of k but the formulas bounding the complexities get much more complicated for larger values of k. For the exact time complexities we defer the reader to the actual theorem statements (Theorems 13 and 14).

Query	Space (words)	Query time	Note
$\mathsf{One\text{-}to\text{-}One}^k(i,j)$	$\mathcal{O}(n\log^k n)$	$\mathcal{O}(\log^k n \log\log n)$	Theorem 13
$\mathsf{Report}^k(i,d)$	$\mathcal{O}(n\log^k n)$	$\mathcal{O}(\log^k n(\log n/\log\log n + \mathsf{output}))$	Theorem 14

Let us introduce the central notion of *extension-prefix pair* for any two strings X and Y.

▶ **Definition 3.** *A pair of strings (Y, X) is an* extension-prefix pair*, if X is a prefix of Y; and hence Y is called an* extension *of X. A pair of strings (Y, X) is a* k-approximate extension-prefix pair*, if X is at (Hamming or edit) distance at most k from some prefix of Y.*

Section 2 introduces the necessary definitions and notation as well as a few observations. In Section 3, we introduce a version of the k-errata tree [15] and show its efficient construction as well as some of its properties that are crucial to arrive at our main results. Specifically in Section 3.1, we show the construction for edit distance, and then also state the changes for the (easier) case of the Hamming distance. In Section 3.2, we show the key combinatorial property of the extension-prefix pairs (Lemma 5) in our k-errata tree. Section 3.3 shows an upper bound on the size of the k-errata tree introduced here for the interesting case where $k = \mathcal{O}(\frac{\log n}{\log\log n})$, which is then used to bound the running time and space of the algorithms – we are not interested in $k = \omega(\frac{\log n}{\log\log n})$ as in this case we have $\text{poly}(n) = \mathcal{O}(\log^k n)$, and hence trivial algorithms become (at least) as efficient as our approach. Section 3.4 refines the combinatorial Lemma 5 to its algorithmic applications (Corollaries 10–12). For the sake of an application, consider that we have two dictionaries D_1 and D_2. We concentrate not only on pairs of strings which are at distance at most k, but globally, for any string $Y \in D_2$, we show how to compute its longest prefix which is at distance at most k from any other string $X \in D_1$, and further show how to compute the set of lengths of all such prefixes.

Finally, in Sections 4.1 and 4.2, we apply our techniques to construct the data structures for answering $\mathsf{One\text{-}to\text{-}One}^k$ and Report^k queries, respectively. Full detailed proofs omitted from the main text are included in Appendix A.

2 Preliminaries

We consider strings over an integer alphabet $\Sigma = [1\mathinner{.\,.}\sigma]$ of size σ. The elements of Σ are called *letters*. A *string* $X = X[1\mathinner{.\,.}n]$ is a sequence of letters from Σ; we denote by $|X| = n$ the *length* of X. The fragment $X[i\mathinner{.\,.}j]$ of X is an *occurrence* of the underlying *substring* $S = X[i]\cdots X[j]$ occurring at *position* i in X. A *prefix* of X is a substring of X of the form

$X[1\,.\,.\,j]$ and a *suffix* of X is a substring of X of the form $X[i\,.\,.\,n]$. The *reverse* of X is the string $X^R = X[n]\cdots X[1]$. By $\mathsf{LCP}(X,Y)$, we denote the length of the longest common prefix of strings X and Y. The *Hamming distance* of two equal-length strings is the number of positions where the strings differ. The *edit distance* (or *Levenshtein distance*) of two strings X and Y is the minimum number of edit operations required to transform X into Y. Here, by *edit operation* we mean an insertion, a substitution, or a deletion of a single letter. A set D of strings is called a *dictionary.*

The following edit-distance property (Lemma 4) underlies most string algorithms on edit distance; e.g., [23]. We provide a proof of this property in Appendix A for completeness.

▶ **Lemma 4.** *For any two strings X and Y, there exists a smallest sequence of edit operations changing X to Y satisfying recursively that the first operation occurs at position $\mathsf{LCP}(X,Y)+1$. By recursively, we mean that after applying the first operation on X to obtain X', the leftmost operation to get from X' to Y occurs at position $\mathsf{LCP}(X',Y)+1 \geq \mathsf{LCP}(X,Y)+1$.*

Let D be a dictionary of $n = |D|$ strings. A node in the trie of D is called *branching* if it has at least two children and *terminal* if it represents a string in D. The *compacted trie* T of D is obtained from the underlying trie by removing all nodes except the root node, the branching and the terminal nodes. The removed nodes are called *implicit* while the remaining ones are called *explicit.* Each terminal node corresponding to a string X from D is labeled with the identifier of X: a pointer to X in D. Edge labels are stored as pointers to fragments of strings in D and so the compacted trie takes $|T| = \mathcal{O}(n)$ extra space.

Our algorithms use a version of the famous k-errata tree of Cole et al. [15]. The tree nodes store information about strings and a representation of edit operations applied on them. In order to efficiently operate on those edit operations we define their representation here as an abstract structure and describe the operations that are performed on such structures later.

For convenience we represent multisets of edit operations by lists sorted non-decreasingly. A Hamming distance list consists of up to k elements, where every element is denoted by l_S, for $l \in [1\,.\,.\,n]$. An edit distance list consists of up to k elements, where every element is denoted by l_E for $l \in [1\,.\,.\,n]$, $E \in \{S, I, D\}$; elements are first sorted by l, then by $D < S < I$, and l_D elements may repeat (denoting that several letter deletions occur at the same position l). A single list does not contain both l_S and l_I, or multiple such elements, for the same l. By $|\ell|$ we denote the number of elements in list ℓ (counting the multiplicities). For any two lists ℓ_1, ℓ_2, by $\ell_1 \subseteq \ell_2$, we denote that every element of ℓ_1 appears in ℓ_2 and the multiplicity of any element in ℓ_1 does not exceed its multiplicity in ℓ_2. By $\ell_1 \cup \ell_2$ we denote the union of the two lists; for each element of ℓ_1 or ℓ_2 its multiplicity is equal to its maximum multiplicity in ℓ_1 and ℓ_2. By $\max^+(\ell)$ we denote the smallest element that can be inserted at the end of list ℓ without breaking the specified condition of the list being non-decreasing; that is, the maximum element of the list if this element may appear multiple times and the smallest larger element if the currently maximal element cannot appear multiple times ($\max^+(\ell) = l_D$ when the last element of ℓ is equal to $l_D, (l-1)_S$ or $(l-1)_I$, $\max^+(\emptyset) = 1_D$). By ℓ^d we denote the prefix of list ℓ containing only the elements l_E such that $l \leq d$, and by $\mathsf{surpl}(\ell)$ we denote the surplus of l_D operations over l_I operations in ℓ, that is $|\{l_D \in \ell\}| - |\{l_I \in \ell\}|$; i.e., by how many letters the length of the string decreases after applying the operations from ℓ.

3 Finding Approximate Extension-Prefix Pairs using the k-Errata Tree

For any dictionary D, extension-prefix pairs can be found by using a trie of D. Every such pair corresponds to a descendant-ancestor node pair in the trie. By using this property, all such pairs can be listed in the optimal time. In this section, we show that for *k-approximate* extension-prefix pairs, a similar property holds for a version of the k-errata tree of the trie of D. We start by showing the tree construction, then we show the counterpart of this property, and we end by showing how to use the tree for finding k-approximate extension-prefix pairs.

The Hamming distance can be treated as a special case of edit distance, where only substitution operations are allowed; we thus focus on the edit distance construction, and then comment on the differences that should be made to obtain the tree for Hamming distance. Let D be a dictionary of $n = |D|$ strings and T be the compacted trie of D. In the complexity analysis of our construction, we assume that $k \leq \frac{\log n}{2 \log \log n}$ [15]. (The construction itself and the combinatorial lemmas *do not assume any bound on* k.)

We compute the *heavy-light decomposition* of T in $\mathcal{O}(n)$ time [32]. Let $s(w)$ denote the number of node labels in the subtree of T rooted at w (these labels identify the full strings from D). We call an edge (u, v) of T *heavy* if $s(v)$ is maximal among every edge originating from u (breaking ties arbitrarily). All other edges are called *light*. We call a node that is reached from its parent through a heavy edge *heavy*; otherwise, it is called *light*. The heavy path of T is the path that starts at the root of T and at each node on the path descends to the heavy child as defined above. The heavy path decomposition is then defined recursively: it is a union of the heavy path of T and the heavy path decompositions of the off-path subtrees of the heavy path. The crucial well-known property of this decomposition is that every root to node path in T passes through $\mathcal{O}(\log n)$ light nodes. Our k-errata tree construction actually refers to the heavy-light decomposition of the uncompacted trie of D – in particular to the direct children of the branching nodes even if those are implicit. The decompositions of both the uncompacted and the compacted versions of T are in a bijection: a child of a branching node of the uncompacted version is light if and only if its closest explicit descendant is light in the compacted version. Therefore we refer to the nodes of the uncompacted version of T, and access all children of the branching nodes in time proportional to the size of its compacted version, obtaining the compacted version of the errata tree in the process.

3.1 The k-Errata Tree Construction

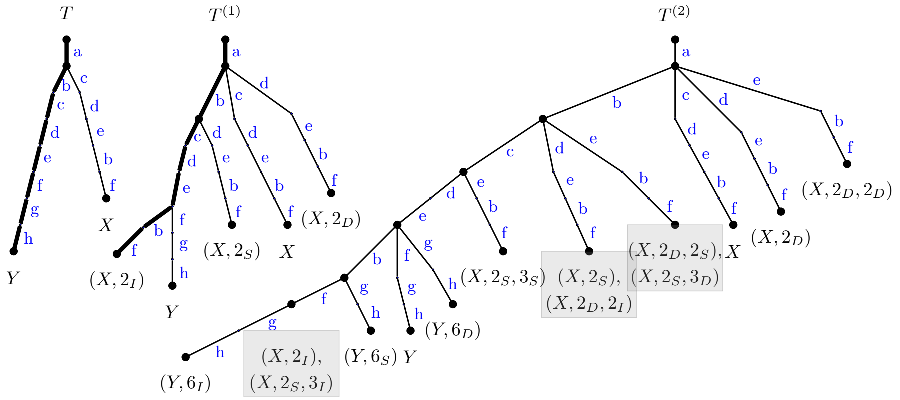


Figure 1 Example of an edit distance 1- and 2-errata tree for trie T of strings $\{Y = \texttt{abcdefgh}, X = \texttt{acdebf}\}$. The rectangles mark the nodes with multiple distinct labels. Notice that a node with label $(Y, 6_I)$ has an ancestor with two labels (X, ℓ), since $|\{2_I\} \cup \{6_I\}| \leq 2$, X is at edit distance at most 2 from some prefix of Y. Label $(X, 2_S, 3_I)$ does not provide such result as $|\{2_S, 3_I\} \cup \{6_I\}| = 3 > 2$.

We denote the k-*errata tree* of T by $T^{(k)}$ and construct it as a 1-errata of $T^{(k-1)}$, where $T^{(0)} = T$. Before constructing the 1-errata tree of T, we compute the heavy-light decomposition of T. For every *light node* v at string depth $l = d(v)$ in T that is reached from its parent through an edge labeled with $a \in \Sigma$, we create three copies of its subtree (inspect Figure 1):

- In one copy, corresponding to a substitution, we change the labels of the subtree nodes from X to (X, l_S) and merge it with the subtree of its heavy sibling.
- In one copy, corresponding to a deletion, we change the labels of the subtree nodes from X to (X, l_D) and merge it with the subtree of its parent.
- In one copy, corresponding to an insertion, we change the labels of the subtree nodes from X to (X, l_I) and merge it with the child of its heavy sibling reached through the edge labeled with letter a (or create such a child if it does not exist).

A node of 1-errata with label: (X, l_S) can be reached by replacing the l-th position of X by the heavy letter (the letter used to reach the heavy child from the node reached by spelling $X[1\mathinner{.\,.} l-1]$ in T); (X, l_D) can be reached by deleting the letter at the l-th position in X; and (X, l_I) can be reached by inserting that heavy letter before the l-th position of X. It should be clear that in the Hamming distance case only the first copy is required.

For convenience, we represent all the added parts of labels collectively with lists of edit operations (see Section 2). To keep the data structure simpler, we avoid introducing redundant labels that would later make proofs and implementations more complicated (in particular an insertion or deletion would change the positions of the edits to the right). Specifically, we produce a copy of a label (X, ℓ) only if the added element l_E is greater or equal to $\max^+(\ell)$. This means that the lists are sorted non-decreasingly. We do not miss any significant sequence of edit operations due to Lemma 4 as shown in the proof of Lemma 5.

Since we may possibly delete a few next letters from the same position (after removing a single letter the next one takes its position) we have to allow elements l_D to appear in a list multiple times. In case of insertions or substitutions this is not needed (and we can prioritize deletions over other operations as there is no reason to first modify a letter and then delete it), hence we do not allow two elements l_I or l_S (or l_I and l_S) for the same $l \in [1\mathinner{.\,.} n]$. As such, the order of elements l_E, $l \in [1\mathinner{.\,.} n], E \in \{S, D, I\}$, is defined by first comparing l, and then $D < S < I$, thus the definition of $\max^+(\ell)$ (see Section 2).

In our construction of trie T, unlike the one used in the classic construction [15] where D is made prefix-free, two terminal nodes can lie on the same heavy path (for an extension-prefix pair (Y, X) for $X, Y \in D$ the node representing X may in fact be an internal labeled node that lies on a heavy path). This does not allow for checking if Y is a k-approximate prefix of X in the same way since it is already an extension of X (in case of edit distance one can always check if the difference of lengths is at most k, and in case of Hamming distance simply respond negatively if $|X| < |Y|$); for example, when $D = \{X = \texttt{abc}, Y = \texttt{abcde}\}$, $T^{(2)} = T$, there is a single heavy path since T has no light nodes. To mitigate this problem and still have the nice properties of our construction (ancestor-descendant relations of the labeled nodes), for a labeled node u with label (X, ℓ) that has (non-trivial) descendants, we also create a copy with label $(X, \ell \cup (d(u)+1)_I)$ and add it to its heavy child (insertion after the last position of X). Substitution and deletion after the end of a string clearly do not make sense, thus an insertion operation is the only one allowed. This change does not influence our bounds on the k-errata tree size as the total size of all such subtrees is the number of the labeled nodes; we change this from $\mathcal{O}(|T| \log^k |T|)$ to $\mathcal{O}(|T|(\log |T| + 1)^k)$.

We remark, that our structure differs substantially from the original k-errata from [15]. In particular, we do not use extra letters outside of the alphabet to mark the edges transitioning from the original tree to a copied part, and instead actually merge the nodes corresponding to the same string. Thanks to this change we do not need to use the extra copies called group trees, which makes the search algorithm simpler in the case of our structure (the first place where the search can diverge from the heavy path is the place where the pattern actually diverges from the heavy path) and running in asymptotically the same time (up to an $f(k)$

factor for some f and $k = \mathcal{O}(\frac{\log n}{\log \log n})$). We do not describe the algorithm in detail however, since in this paper we are only interested in internal queries (the data structure is constructed for the fixed dictionary D and no other string is ever queried). All those simplifications pose the cost of dealing with the lists of edit operations, which are not present in the data structure from [15], but we show how to do that efficiently in the remainder of this section.

3.2 Extension-Prefix Pairs in a k-Errata Tree

We now prove the key combinatorial property of the described k-errata tree of dictionary D.

▶ **Lemma 5.** *For any two elements $X, Y \in D$, X is at edit (resp. Hamming) distance at most k from some prefix of Y if and only if there exist two nodes, u and its descendant v, in $T^{(k)}$ with labels (X, ℓ_1) and (Y, ℓ_2) respectively, such that $|\ell_1 \cup \ell_2| \leq k$, where $T^{(k)}$ is the edit (resp. Hamming) distance k-errata tree of D.*

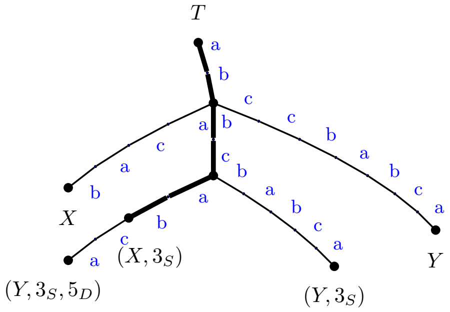


Figure 2 Illustration of a part of the edit distance 2-errata tree T for dictionary D containing $X =$ `abacab` and $Y =$ `abccbabca` with the choice of heavy paths depicted with thicker lines. X is at edit distance 2 from $Y' =$ `abccbab`, which is a prefix of Y; the operations used to transform X into Y' are *substitution* at position 3 and *insertion* at position 5. The lowest common ancestor of nodes with labels X and Y is at string depth 2 (`ab`); as shown in the proof of Lemma 5 there exists a child of this node that is a common ancestor of nodes with labels (X, ℓ'_1) and (Y, ℓ'_2) such that $\ell'_1 \cup \ell'_2 = \{3_S\}$, and a descendant of this child with label (X, ℓ_1) that is an ancestor of a node with label (Y, ℓ_2), such that $|\ell_1 \cup \ell_2| = 2$. Conversely, since there exists such an ancestor-descendant pair of nodes one can compute the list of at most 2 edit operations that transform X into Y'.

Sketch of Proof. The proof of the forward implication follows a path down the k-errata tree – one starts with empty lists ℓ_1 and ℓ_2. Each time an edit operation is applied to X one adds an element to ℓ_1 or ℓ_2 or both (depending on the edit operation and the location of the heavy child) and finds a descendant of the current node that is a common ancestor of nodes with labels (X, ℓ_1) and (Y, ℓ_2). The backward implication can be obtained by reversing that approach and producing a list of at most k edit operations that transform X into a prefix of Y (see Appendix A for the full proof and Figure 2 for an example of the construction). ◀

The full proof shows that, if a prefix Y' of Y is at distance at most k from X, then there exists a pair of nodes u and v with labels (X, ℓ_1) and (Y, ℓ_2), respectively, in an ancestor-descendant relationship, such that $f(d(u)) - k + |\ell_1 \cup \ell_2| \leq |Y'| \leq f(d(u))$, where $f(l) = l + \mathsf{surpl}(\ell_2^l)$ is the actual length of the prefix of Y represented by the ancestor of v at depth l. In particular, the longest prefix is represented by either such a pair for $u = v$ (the whole Y is at edit distance at most k from X), or a pair where $|\ell_1 \cup \ell_2| = k$, and hence $\ell_2 = \ell_2^{d(u)}$.

This property and the monotonicity of $f(l)$ for a single label (Y, ℓ_2) of node v imply that to find the longest prefix of Y at distance at most k from X it is enough to focus on the lowest ancestor u of v with label (X, ℓ_1) such that $|\ell_1 \cup \ell_2| \leq k$. In fact, if $\ell_2 \neq \ell_2^{d(u)}$, then both $u \neq v$ and $|\ell_1 \cup \ell_2^{d(u)}| < |\ell_1 \cup \ell_2| \leq k$, hence there must exist a pair u', v' with labels (X, ℓ'_1) and (Y, ℓ'_2) respectively, for which $d(u') + \mathsf{surpl}(\ell_2'^{d(u')}) > d(u) + \mathsf{surpl}(\ell_2^{d(u)})$ (we can extend the prefix Y' by inserting the next letter of Y after its last position), hence we get the following corollary. (The algorithm can easily use $\ell_2^{d(u)}$ for the comparison between different ℓ_2, but it is neater this way.)

▶ **Corollary 6.** *The length of the longest prefix of Y that is at edit distance at most k from X is equal to the maximum over all node pairs (u, v) of value $d(u) + \mathsf{surpl}(\ell_2)$, where u has label (X, ℓ_1), v has label (Y, ℓ_2), u is a (potentially trivial) ancestor of v, and $|\ell_1 \cup \ell_2| \leq k$.*

This is also the case for the Hamming distance but then $\mathsf{surpl}(\ell_2) = 0$.

When we are interested not only in the longest prefix of Y at edit distance at most k from X, but in the set of all the prefixes with this property, it is enough to compute the union of the intervals $d(u) + \mathsf{surpl}(\ell_2) + [-k + |\ell_1 \cup \ell_2| \mathinner{..} 0]$ over all such pairs u and v.

▶ **Corollary 7.** *The set of lengths of prefixes of Y that are at edit distance at most k from X is equal to the union of intervals $d(u) + \mathsf{surpl}(\ell_2) + [-k + |\ell_1 \cup \ell_2| \mathinner{..} 0]$ over all node pairs (u, v), where u has label (X, ℓ_1), v has label (Y, ℓ_2), u is a (potentially trivial) ancestor of v, and $|\ell_1 \cup \ell_2| \leq k$.*

For Hamming distance the same property is simpler as the intervals are singletons $\{d(u)\}$.

3.3 Size of the k-Errata Tree

For completeness of the k-errata tree construction, we show a bound on the size of the data structure for a compacted trie T with at most n explicit nodes and n distinct node labels. We assume $k \leq \frac{\log n}{2 \log \log n}$ (for all further complexity considerations), but the properties and the correctness of the algorithms shown further do not require this assumption on k. For such a value of k, the size of our data structure is asymptotically the same as of the k-errata tree from [15] up to an $f(k)$ factor (the bound is not necessarily tight – optimizing the $f(k)$ value is not a focus of this paper). The proof of Lemma 8 can be found in Appendix A.

▶ **Lemma 8.** *For any compacted trie T with at most n explicit nodes and n distinct node labels, and for any $k \leq \frac{\log n}{2 \log \log n}$, the k-errata tree $T^{(k)}$ has $\mathcal{O}(nk!(c_\delta \log n)^k)$ explicit nodes and labels, where $c_\delta = 1$ for Hamming distance and $c_\delta = 3$ for edit distance.*

In particular, a single node label of T has $\mathcal{O}(k!(c_\delta \log n)^k)$ copies in $T^{(k)}$.

▶ **Corollary 9.** *For any $k \leq \frac{\log n}{2 \log \log n}$, we have $\log |T^{(k)}| = \mathcal{O}(k \log n)$.*

In what follows, we give the complexity of our algorithms including the proven bound on the $f(k)$ factor, and additionally the formulas in the most interesting case $k = \mathcal{O}(1)$.

3.4 Finding Approximate Prefixes

In Section 3.2, we were interested in a single extension-prefix pair (Y, X), such that $X, Y \in D$. Here, say that we are given two subdictionaries D_1 and D_2 such that $D_1, D_2 \subseteq D$, and want to find for every element of D_2 its longest prefix that is at distance at most k from some element of D_1 using Corollary 6, or the lengths of all such prefixes using Corollary 7. Let $n = |D|, n_1 = |D_1|, n_2 = |D_2|$, and let m be the length of the longest string in D.

By running a DFS on $T^{(k)}$ and storing the labels from all the ancestors of a given node labeled (Y, ℓ_2), $Y \in D_2$, we can find all the pairs where this node is the descendant. We need, however, to take care of the lists of edit operations: for the label (Y, ℓ_2), we have to look for the elements (X, ℓ_1), $X \in D_1$, stored in the set of ancestor labels, such that $|\ell_1 \cup \ell_2| \leq k$. We do not want to check all the elements from this set separately however (potentially there are $\Omega(n)$ such elements). We can group the elements based on their lists ℓ_1; still for a single ℓ_2 there can be $\binom{m}{k}$ such fitting lists ℓ_1 for Hamming distance (and even more for edit distance).

Corollary 6 tells us that as long as $|\ell_1 \cup \ell_2| \leq k$ it does not matter what is the content of ℓ_1, nor what is the X in the label as long as $X \in D_1$. In order to avoid looking for each list ℓ_1 separately, we can store the elements (string depths of nodes with labels (X, ℓ_1) for any $X \in D_1$ and ℓ_1) grouped by the size of ℓ_1 and its intersection with every possible ℓ_2.

More formally we can store the information about the ancestors of the current node of the DFS traversal in stacks indexed with pairs (ℓ, x), where ℓ is a valid list of elements and $x \in [0\,..\,k - |\ell|]$. For a single label (X, ℓ_1) of node u we store $d(u)$ in stacks (ℓ, x), such that $\ell \subseteq \ell_1, |\ell_1| - |\ell| \leq x \leq k - |\ell|$ (ℓ_1 contains all elements of ℓ plus at most x other elements) upon reaching u (and remove it when returning to its parent). Since a list can have up to k elements, every single label (X, ℓ_1) is responsible for $2^{|\ell_1|} \cdot (k - |\ell_1| + 1) \leq 2^k(k+1)$ elements in stacks, hence in total the DFS traversal will perform $\mathcal{O}(2^k(k+1) \cdot |T^{(k)}|) = \mathcal{O}(n(k+1)!(2 \cdot c_\delta \log n)^k)$ operations on stacks. Notice that we use a sparse representation (hash table) to store only the non-empty stacks explicitly – the universe of all possible stacks is of size at least m^k. We can access a stack in $\mathcal{O}(1)$ worst-case time by using perfect hashing [8].

Now, for a single label (Y, ℓ_2) of a node v, $Y \in D_2$, to find the deepest node u with label (X, ℓ_1) over all $X \in D_1$, such that $|\ell_1 \cup \ell_2| \leq k$, one only needs to take the maximum of the top elements over at most $2^{|\ell_2|} = \mathcal{O}(2^k)$ stacks $(\ell, k - |\ell_2|)$ for $\ell \subseteq \ell_2$. Through the whole DFS, we need to perform a total of $\mathcal{O}(n_2 \cdot 2^k k!(c_\delta \log n)^k)$ such operations for all $Y \in D_2$ as there are $\mathcal{O}(k!(c_\delta \log n)^k)$ copies of label Y in $T^{(k)}$ by Lemma 8.

▶ **Corollary 10.** *For any two dictionaries D_1 and D_2 such that $D_1, D_2 \subseteq D$, we can find the longest prefix of Y, for every $Y \in D_2$ at (Hamming or edit) distance at most k from some string $X \in D_1$, in $\mathcal{O}(n(k+1)!(2 \cdot c_\delta \log n)^k)$ total time; for any $k = \mathcal{O}(1)$, this is $\mathcal{O}(n \log^k n)$.*

If we rather want to find the list of *all* the prefixes of Y which are at distance at most k from any $X \in D_1$, we need to look through the whole stacks, not only at the top elements, and also pay attention to the size of $\ell_1 \cup \ell_2$, for labels (Y, ℓ_2).

For a single label (Y, ℓ_2) of a node v, if a node u with label (X, ℓ_1), such that $|\ell_1 \cup \ell_2| = x \leq k$, is an ancestor of v, then upon visiting v, $d(u)$ will be stored on stack (ℓ, y), where $\ell = \ell_1 \cap \ell_2$, and $y = |\ell_1| - |\ell| = |\ell_1 \cup \ell_2| - |\ell_2|$ (as $|\ell_1 \cup \ell_2| = |\ell_1| + |\ell_2| - |\ell|$). For such an element on the stack, we produce an interval $d(u) + \mathsf{surpl}(\ell_2) + [-k + |\ell_1 \cup \ell_2|\,..\,0] = d(u) + \mathsf{surpl}(\ell_2) + [-k + y + |\ell_2|\,..\,0]$ of lengths of prefixes of Y that are at edit distance at most k from X (by Corollary 7), or simply a length $d(u)$ of a prefix of Y at Hamming distance at most k from X.

The element (X, ℓ_1) can be also represented on other stacks (ℓ', y') for $\ell' \subseteq \ell_1 \cap \ell_2$ and $y' \geq |\ell_1| - |\ell'|$, but then $y' + |\ell_2| \geq |\ell_1| + |\ell_2| - |\ell'| \geq |\ell_1| + |\ell_2| - |\ell| = |\ell_1 \cup \ell_2| = y + |\ell_2|$; hence the interval generated this way is a subset of the interval represented by (ℓ, y) and thus does not change the union of such intervals (still we do not know what is the right ℓ or y, so we need to check all valid pairs).

This way we can compute the union of those at most $m \cdot 2^k(k+1)$ elements of stacks for Hamming distance or of at most $(m+k) \cdot 2^k(k+1)$ intervals for edit distance for a single label (Y, ℓ_2) of node v in time proportional to those values (in case of edit distance using the sweep line technique).

▶ **Corollary 11.** *For any two dictionaries D_1 and D_2 such that $D_1, D_2 \subseteq D$, we can find the lengths of all the prefixes of Y at (Hamming or edit) distance at most k from some string $X \in D_1$ for every $Y \in D_2$ in $\mathcal{O}((n + n_2(m+k))(k+1)!(2 \cdot c_\delta \log n)^k)$ total time; for any $k = \mathcal{O}(1)$, this is $\mathcal{O}((n + n_2 m)\log^k n)$, where m is the length of the longest string in D.*

If the focus of the output is oriented towards D_1 instead (we want to know what strings from D_1 are at distance at most k from some prefix of a string in D_2), then we can store in the stacks for each label (X, ℓ_1) the identifier of X instead. In this case if we run a DFS, and for each label (Y, ℓ_2) we collect the union of sets represented by the stacks (ℓ, y), then the union of those results over all $Y \in D_2$ and all ℓ_2 would be equal to exactly the elements of D_1 at distance at most k from some prefix of some $Y \in D_2$.

To avoid handling unnecessary duplicates of strings X for a single (ℓ, y) (with different lists $\ell_1 \supseteq \ell$ and $\ell_1' \supseteq \ell$), or for many different Y, instead of stacks, we store the information in a multiset data structure: a hash table storing as keys the elements of D_1 and as satellite data the multiplicity of every element. When an element $X \in D_1$ is read (and hence also returned), we delete it from every hash table and store it separately, so that it is never added again. The removal of those copies from every multiset is easy since we additionally store pointers to all of them partitioned by the elements of D_1. For the label (Y, ℓ_2), when reading this multiset, we iterate over the elements in the hash table in time proportional to their number (the deleted elements are stored separately). We implement multisets as dynamic perfect hash tables with $\mathcal{O}(1)$-time worst-case operations [8].

▶ **Corollary 12.** *For any two dictionaries D_1 and D_2 such that $D_1, D_2 \subseteq D$, we can find all the elements of D_1 at (Hamming or edit) distance at most k from a prefix of some $Y \in D_2$ in $\mathcal{O}(n(k+1)!(2 \cdot c_\delta \log n)^k)$ total time; for any $k = \mathcal{O}(1)$, this is $\mathcal{O}(n \log^k n)$.*

4 Application to Approximate Suffix-Prefix Dictionary Queries

Recall that we are given a dictionary $R = \{S_1, \ldots, S_r\}$ of r strings whose total length is n, and we want to find, for a given set of pairs i, j, the value $\mathsf{SPL}^k_{i,j}$ equal to the length of the longest suffix of S_i that is at (Hamming or edit) distance at most k from some prefix of S_j.

We focus on finding the longest prefix of S_j that is at distance at most k from some suffix of S_i, as our k-errata structures are more focused on the prefix lengths than suffix lengths; by Observation 2, those values are not necessarily maximized simultaneously in the edit distance case. Still the two problems reduce to one another by reversing all the strings in R. Henceforth, $\mathsf{SPL}^k_{i,j}$ denotes the length of the longest prefix of S_j that is at (Hamming or edit) distance at most k from some suffix of S_i.

In preprocessing, we construct the generalized suffix tree ST_R of R (without the commonly used $\$ \notin \Sigma$ separators) [16]: ST_R is the compacted trie of the suffixes of all strings in R. A node u in ST_R is labeled by i if and only if u represents a suffix of string S_i from R. Provided that we are given an integer $k > 0$, we also construct the k-errata tree $ST_R^{(k)}$ of ST_R. We also distinguish the labels corresponding to the full strings S_i (and not their non-trivial suffixes) – we mark all their copies and store links to them for $\mathcal{O}(1)$-time access.

4.1 Approximate One-to-One Queries

We first describe the full data structure for answering $\mathsf{One\text{-}to\text{-}One}^k$ queries; then we give the querying algorithm; and, finally, we analyze the data structure size and the query time. Recall that $\mathsf{One\text{-}to\text{-}One}^k(i, j)$ returns $\mathsf{SPL}^k_{i,j}$, for two $i, j \in [1\,..\,r]$.

Data Structure. After preprocessing $ST_R^{(k)}$, we construct the data structure for answering $\mathsf{One\text{-}to\text{-}One}^k$ queries. It consists of $\mathcal{O}((k+1)2^k)$ trees, in total, of three types:

- the main tree $ST_R^{(k)}$;
- trees $ST_{i,\ell}$, which contain nodes of $ST_R^{(k)}$ with labels (i,ℓ_1), such that $\ell \subseteq \ell_1$, for $i \in [1\mathinner{.\,.} r]$;
- trees $ST_{i,\ell,x}$, for $x \in [0\mathinner{.\,.} k-|\ell|]$, which contain nodes with labels (i,ℓ_1), such that $\ell \subseteq \ell_1$ and $x \geq |\ell_1| - |\ell|$.

Note that $|\ell| \leq k$ and that we do not actually construct the empty trees (i.e., trees that contain no labels in the instance of the problem). Even though $ST_{i,\emptyset}$ is basically the counterpart of the tree ST_i from the One-to-One without errors solution [25], it is not necessarily equal to the k-errata tree of the suffix tree of S_i, since the heavy-light decomposition of ST_i and of the generalized suffix tree (and hence the errata trees) may differ. That is, to obtain a smaller tree, we make a copy of a larger tree, remove the labels that do not fit, remove the nodes that do not have any labels in their subtree, and finally compactify the tree.

The trees of the first two types are enhanced with rank-select (RS) [6] and lowest common ancestor (LCA) data structures [9]; the trees of the third type are enhanced with a weighted ancestor (WA) data structure [3] plus a pointer from each node to its closest ancestor with a label. (The latter information can be trivially computed by using a DFS tree traversal.)

While the LCA and WA data structures are by default defined on (rooted) trees, we need to specify how the RS data structures are defined on trees (as they are usually defined on arrays). In our case, the RS data structures span over all the node labels (in one of the constructions a single label will actually be given a few next positions). In particular, the positions are first ordered by the position of the node (left to right DFS first visit order), and next its labels (i,ℓ) are ordered lexicographically. In case of $ST_R^{(k)}$, the RS data structure is over the alphabet $[1\mathinner{.\,.} r]$, where the letter representing the node label (i,ℓ) is i. Each label of $ST_R^{(k)}$ is linked with its counterpart from $ST_{i,\emptyset}$. The tree $ST_{i,\ell}$ is connected to trees $ST_{i,\ell\cup\{l_E\}}$, where $l_E \geq \max^+(\ell)$ and trees $ST_{i,\ell,x}$, where $x \leq k-|\ell|$, and hence it contains two separate RS data structures. The first one is over the alphabet of valid elements: $[\max^+(\ell)\mathinner{.\,.} n_S]$ for Hamming distance; $[\max^+(\ell)\mathinner{.\,.}(n+k)_I]$ for edit distance. Each label (i,ℓ_1) is given a position for every element $l_E \in \ell_1$ greater or equal to $\max^+(\ell)$ and this element l_E is its representative. This position in the RS array is linked with the corresponding terminal node of tree $ST_{i,\ell\cup\{l_E\}}$. The second one is over alphabet $[0\mathinner{.\,.} k-|\ell|]$. Each node label (i,ℓ_1) is given $k-|\ell_1|+1$ positions that contain values $x \in [|\ell_1|-|\ell|\mathinner{.\,.} k-|\ell|]$. Each such position is connected to the corresponding node label of the tree $ST_{i,\ell,x}$.

Querying. We want to find $\mathsf{SPL}_{i,j}^k$, i.e., the longest prefix of S_j that is at (Hamming or edit) distance at most k from some suffix of S_i (if we store some additional information with the labels we can also find the length of this suffix). By Lemma 8, S_j corresponds to $\mathcal{O}(k!(c_\delta \log n)^k)$ different node labels (j,ℓ_2) in $ST_R^{(k)}$; we will consider each such label independently, and then choose the optimal result.

For a single node v with label (j,ℓ_2), we want to find its lowest ancestor with label (i,ℓ_1) such that $|\ell_1 \cup \ell_2| \leq k$. We know that the ancestor is contained in one of the trees $ST_{i,\ell,k-|\ell_2|}$ for some $\ell \subseteq \ell_2$. If we know a node v' with a label that is stored in that tree, and is the closest to v in the DFS order of $ST_R^{(k)}$ (they have the deepest LCA among all such nodes v'), then we know that any ancestor of v that belongs to $ST_{i,\ell,k-|\ell_2|}$ is also an ancestor of v', and that every ancestor of v' at depth at most $\mathsf{LCA}(v,v')$ is also an ancestor of v. Thus for knowing the location of v' in $ST_{i,\ell,k-|\ell_2|}$ and the value of $\mathsf{LCA}(v,v')$, it suffices to ask a WA query in $ST_{i,\ell,k-|\ell_2|}$ for node v' and depth $\mathsf{LCA}(v,v')$ and then ask for the lowest ancestor of the node with a label (this information is stored in the node as a pointer) - the answer is

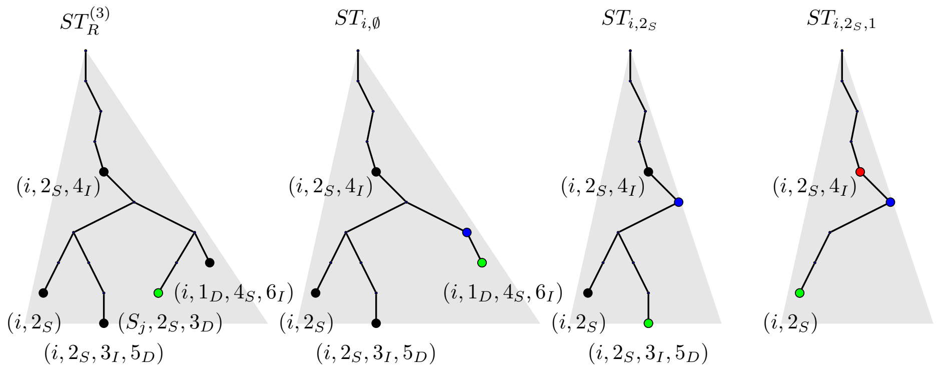


Figure 3 An example traversal through the trees to reach the lowest ancestor (i,ℓ_1) of label $(S_j,2_S,3_D)$, such that $2_S \in \ell_1$ and $|\ell_1 \cup \{2_S,3_D\}| \leq 3$. The green node depicts the label that is closest to $(S_j,2_S,3_D)$ in $ST_R^{(3)}$, out of the ones existing in the tree, while the blue one depicts the lowest common ancestor of the two nodes (equal to green in $ST_R^{(3)}$). The next green node is obtained via a rank-select query, while the blue nodes depth is obtained via an LCA query. After reaching a terminating tree, we ask for the depth of the lowest ancestor of the blue node that contains a label.

the sought ancestor of v. To obtain the location of v' and the value of $\mathsf{LCA}(v,v')$, we jump through the trees in between, always storing the two pieces of information: the labeled node v'' closest to v that appears in the tree; and the value $\mathsf{LCA}(v,v'')$.

In $ST_R^{(k)}$ we want to find the closest label (i,ℓ_1) for any ℓ_1; we simply ask an RS query to find the closest such node to the left and to the right of v. We compare those two nodes by the string depth of their LCA with v, choose the one for which this depth is larger (breaking ties arbitrarily), and then we proceed to the node with this label in $ST_{i,\emptyset}$.

Now, in tree $ST_{i,\ell}$, for $\ell \subseteq \ell_2$, we start with the labeled node v' closest to v in $ST_R^{(r)}$, and the string depth of their LCA. For each element $l_E \in \ell_2$, such that $l_E \geq \max^+(\ell)$, we find the closest node with label (i,ℓ_3) such that $\ell \cup \{l_E\} \subseteq \ell_3$ in $ST_{i,\ell}$ by asking the RS data structure (the one over $[\max^+(\ell)\,..\,(n+k)_I]$) for l_E – both left and right. We compare those two nodes based on the LCA string depth with v' to get the node v'', then compute $\mathsf{LCA}(v,v'') = \min\{\mathsf{LCA}(v,v'),\mathsf{LCA}(v',v'')\}$, and jump to the corresponding terminal node of $ST_{i,\ell\cup\{l_E\}}$ with the new LCA value. Additionally we ask the other RS data structure for the closest label represented by $k-|\ell_2|$, and jump to the labeled node of $ST_{i,\ell,k-|\ell_2|}$. There we find its lowest ancestor at depth at most the value of stored LCA string depth, and ask this ancestor for the string depth of its lowest ancestor storing a label. If this value is larger than the value of the currently stored candidate, we replace the candidate with the new one.

After the whole recursive procedure ends for each labeled node v corresponding to the full string S_j, the stored candidate is the final answer. Inspect Figure 3, for an example.

Data Structure Size. Each node of $ST_R^{(k)}$ belongs to $\mathcal{O}((k+1)\cdot 2^k)$ different trees. In total, the trees have size $N = \mathcal{O}(n(k+1)!(2c_\delta \log n)^k)$. The RS [6], LCA [9], and WA [3] data structures occupy $\mathcal{O}(N)$ space and can be constructed in $\mathcal{O}(N \log\log N)$ time.

Query Time. We have $\mathcal{O}(k!(c_\delta \log n)^k)$ node labels (S_j,ℓ_2) for a full string S_j. For each of those we reach at most 2^k further trees. Each operation of reaching the next tree (from the previous one) costs a constant number of RS, LCA and WA queries – that is, the cost of reaching each tree is $\mathcal{O}(\log\log n)$; hence the total cost is $\mathcal{O}((2c_\delta \log n)^k \log\log n)$. In particular, the RS [6] and WA [3] queries take $\mathcal{O}(\log\log n)$ time when $\mathcal{O}(n)$ space is used.

▶ **Theorem 13.** *For any dictionary of strings of total length n and any $k \leq \frac{\log n}{2\log\log n}$, we can construct a data structure of $\mathcal{O}(n(k+1)!(2c_\delta \log n)^k)$ words of space answering* $\mathsf{One\text{-}to\text{-}One}^k$ *queries in $\mathcal{O}((k+1)!(2c_\delta \log n)^k \log\log n)$ time. The data structure can be constructed in $\mathcal{O}(n(k+1)!(2c_\delta \log n)^k \log\log n)$ time.*

For any $k = \mathcal{O}(1)$, the data structure size is $\mathcal{O}(n \log^k n)$, the query time is $\mathcal{O}(\log^k n \log\log n)$ and the construction time is $\mathcal{O}(n \log^k n \log\log n)$.

4.2 Approximate Report Queries

Once again we use an inverted version of the problem that is equivalent to the original one; i.e., we want to answer $\mathsf{Report}^k(j,d)$ queries that return all the values of $i \in [1\,..\,r]$ such that there exits a prefix of S_j at distance at most k from some suffix of S_i of length at least d. Recall that by Observation 2, for edit distance, this is not necessarily equivalent to finding all j such there exists a prefix of S_j of length at least d at distance at most k from some suffix of S_i – this can be done using the idea of Corollary 12 within the same time complexity.

By Corollary 6, to find whether a prefix of length at least d of S_j is at distance at most k from a suffix of S_i, it suffices to check, for every label (j,ℓ_2) of a node v representing the full string S_j, if it has an ancestor u with label (i,ℓ_1), such that $|\ell_1 \cup \ell_2| \leq k$ and $d(u) \geq d - \mathsf{surpl}(\ell_2)$. We first describe the data structure for answering Report^k queries and analyze its size; then we give the querying algorithm and analyze the query time.

Data Structure. We start by constructing $ST_R^{(k)}$. For each pair (ℓ, x), such that $x \leq k - |\ell|$ and there exists a label with list ℓ in $ST_R^{(k)}$, we create a linear-space 2D rectangle stabbing data structure [30]. For every node u with label (i,ℓ_1), we create a rectangle $[L(u)\,..\,R(u)] \times [0\,..\,d(u)]$ with label i, where $L(u)$ and $R(u)$ are the numbers of the leftmost and the rightmost label in the subtree of u in $ST_R^{(k)}$ when ordered in the left-to-right DFS traversal order, and insert it to data structures (ℓ, x) for each $\ell \subseteq \ell_1$ and $|\ell_1| - |\ell| \leq x \leq k - |\ell|$. Next in every structure we make the rectangles of the same type disjoint.

We have in total $|ST_R^{(k)}| = \mathcal{O}(nk!(c_\delta \log n)^k)$ labels, hence the total number of rectangles in all the data structures is $|ST_R^{(k)}| \cdot (k+1)2^k = \mathcal{O}(n(k+1)!(2c_\delta \log n)^k)$, and the dimensions of the rectangles are in $[0\,..\,|ST_R^{(k)}|] \times [0\,..\,n]$. Hence the total size is $\mathcal{O}(n(k+1)!(2c_\delta \log n)^k)$.

Querying. For each label (j,ℓ_2) of a node v representing the full string S_j, and for each $\ell \subseteq \ell_2$, we ask a query $(L(v), d - \mathsf{surpl}(\ell_2))$ to the 2D rectangle stabbing data structure $(\ell, k - |\ell_2|)$. Over all labels (j,ℓ_2), we have in total $\mathcal{O}(k!(2c_\delta \log n)^k)$ lists of total length $\mathcal{O}(|Q| \cdot k!(2c_\delta \log n)^k)$, where $|Q|$ is the size of the output, and we want to output their union. By using the 2D rectangle stabbing data structure from [30] we obtain the following result.

▶ **Theorem 14.** *For any dictionary of strings of total length n and any $k \leq \frac{\log n}{2\log\log n}$, we can construct a data structure of $\mathcal{O}(nk!(2c_\delta \log n)^k)$ words of space answering* Report^k *queries in $\mathcal{O}((k+1)!(2c_\delta \log n)^k(\log n/\log\log n + |Q|))$ time. For any $k = \mathcal{O}(1)$, the data structure size is $\mathcal{O}(n \log^k n)$ and the query time is $\mathcal{O}(\log^k n(\log n/\log\log n + |Q|))$.*

References

1 Paniz Abedin, Arnab Ganguly, Solon P. Pissis, and Sharma V. Thankachan. Efficient data structures for range shortest unique substring queries. *Algorithms*, 13(11):276, 2020. `doi:10.3390/A13110276`.

2 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. `doi:10.1007/s00453-020-00744-0`.

3 Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.

4 Golnaz Badkobeh, Panagiotis Charalampopoulos, Dmitry Kosolobov, and Solon P. Pissis. Internal shortest absent word queries in constant time and linear space. *Theor. Comput. Sci.*, 922:271–282, 2022. doi:10.1016/j.tcs.2022.04.029.

5 Carl Barton, Costas S. Iliopoulos, Solon P. Pissis, and William F. Smyth. Fast and simple computations using prefix tables under hamming and edit distance. In Jan Kratochvíl, Mirka Miller, and Dalibor Froncek, editors, *Combinatorial Algorithms - 25th International Workshop, IWOCA 2014, Duluth, MN, USA, October 15-17, 2014, Revised Selected Papers*, volume 8986 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014. doi:10.1007/978-3-319-19315-1_5.

6 Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015. doi:10.1145/2629339.

7 Ilan Ben-Bassat and Benny Chor. String graph construction using incremental hashing. *Bioinform.*, 30(24):3515–3523, 2014. doi:10.1093/bioinformatics/btu578.

8 Michael A. Bender, Alex Conway, Martin Farach-Colton, William Kuszmaul, and Guido Tagliavini. Iceberg hashing: Optimizing many hash-table criteria at once. *J. ACM*, 70(6):40:1–40:51, 2023. doi:10.1145/3625817.

9 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

10 Paola Bonizzoni, Gianluca Della Vedova, Yuri Pirola, Marco Previtali, and Raffaella Rizzi. FSG: fast string graph construction for de novo assembly. *J. Comput. Biol.*, 24(10):953–968, 2017. doi:10.1089/cmb.2017.0089.

11 Panagiotis Charalampopoulos, Pawel Gawrychowski, Yaowei Long, Shay Mozes, Seth Pettie, Oren Weimann, and Christian Wulff-Nilsen. Almost optimal exact distance oracles for planar graphs. *J. ACM*, 70(2):12:1–12:50, 2023. doi:10.1145/3580474.

12 Panagiotis Charalampopoulos, Tomasz Kociumaka, Manal Mohamed, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal dictionary matching. *Algorithmica*, 83(7):2142–2169, 2021. doi:10.1007/s00453-021-00821-y.

13 Shiri Chechik. Approximate distance oracles with constant query time. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 654–663. ACM, 2014. doi:10.1145/2591796.2591801.

14 Shiri Chechik. Approximate distance oracles with improved bounds. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 1–10. ACM, 2015. doi:10.1145/2746539.2746562.

15 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100. ACM, 2004. doi:10.1145/1007352.1007374.

16 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.

17 Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping dynamic distance oracles. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 50:1–50:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.50.

18 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

19 Dan Gusfield, Gad M. Landau, and Baruch Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Inf. Process. Lett.*, 41(4):181–185, 1992. doi:10.1016/0020-0190(92)90176-V.

20 Tomasz Kociumaka. Efficient data structures for internal queries in texts. *PhD thesis, University of Warsaw, October 2018.*, 2018. URL: https://https://www.mimuw.edu.pl/~kociumaka/files/phd.pdf.

21 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Internal pattern matching queries in a text and applications. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 532–551. SIAM, 2015. doi:10.1137/1.9781611973730.36.

22 Gregory Kucherov and Dekel Tsur. Improved filters for the approximate suffix-prefix overlap problem. In Edleno Silva de Moura and Maxime Crochemore, editors, *String Processing and Information Retrieval - 21st International Symposium, SPIRE 2014, Ouro Preto, Brazil, October 20-22, 2014. Proceedings*, volume 8799 of *Lecture Notes in Computer Science*, pages 139–148. Springer, 2014. doi:10.1007/978-3-319-11918-2_14.

23 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *J. Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.

24 Grigorios Loukides and Solon P. Pissis. All-pairs suffix/prefix in optimal time using Aho-Corasick space. *Inf. Process. Lett.*, 178:106275, 2022. doi:10.1016/j.ipl.2022.106275.

25 Grigorios Loukides, Solon P. Pissis, Sharma V. Thankachan, and Wiktor Zuba. Suffix-prefix queries on a dictionary. In Laurent Bulteau and Zsuzsanna Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 21:1–21:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.CPM.2023.21.

26 Eugene W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, September 2005. doi:10.1093/bioinformatics/bti1114.

27 Enno Ohlebusch and Simon Gog. Efficient algorithms for the all-pairs suffix-prefix problem and the all-pairs substring-prefix problem. *Inf. Process. Lett.*, 110(3):123–128, 2010. doi:10.1016/j.ipl.2009.10.015.

28 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014. doi:10.1137/11084128X.

29 Kim R. Rasmussen, Jens Stoye, and Eugene W. Myers. Efficient q-gram filters for finding all *epsilon*-matches over a given length. *J. Comput. Biol.*, 13(2):296–308, 2006. doi:10.1089/cmb.2006.13.296.

30 Qingmin Shi and Joseph F. JáJá. Novel transformation techniques using q-heaps with applications to computational geometry. *SIAM J. Comput.*, 34(6):1474–1492, 2005. doi:10.1137/S0097539703435728.

31 Jared T. Simpson and Richard Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinform.*, 26(12):367–373, 2010. doi:10.1093/bioinformatics/btq217.

32 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

33 Saumya Talera, Parth Bansal, Shabnam Khan, and Shahbaz Khan. Practical algorithms for hierarchical overlap graphs. *CoRR*, abs/2402.13920, 2024. doi:10.48550/arXiv.2402.13920.

34 Sharma V. Thankachan, Chaitanya Aluru, Sriram P. Chockalingam, and Srinivas Aluru. Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis. In Benjamin J. Raphael, editor, *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, April 21-24, 2018, Proceedings*, volume 10812 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2018. doi:10.1007/978-3-319-89929-9_14.

35 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. doi:10.1145/1044731.1044732.

36 William H. A. Tustumi, Simon Gog, Guilherme P. Telles, and Felipe A. Louza. An improved algorithm for the all-pairs suffix-prefix problem. *J. Discrete Algorithms*, 37:34–43, 2016. doi:10.1016/j.jda.2016.04.002.

37 Niko Välimäki, Susana Ladra, and Veli Mäkinen. Approximate all-pairs suffix/prefix overlaps. *Inf. Comput.*, 213:49–58, 2012. doi:10.1016/j.ic.2012.02.002.

A Omitted Proofs

Proof of Lemma 4. Say, that the leftmost operation in an optimal sequence of edit operations between strings X and Y happens at position $l_1 \leq \mathsf{LCP}(X, Y)$, that is, in particular $X[l_1] = Y[l_1]$.

The positions $1, \ldots, l_1 - 1$ of strings X and Y are matched respectively, and we need to match the letter $Y[l_1]$ with something in X. Therefore, the leftmost operation (subset of subsequent operations) is either an insertion of letter $Y[l_1]$ or a possibly empty sequence of deletions followed by an identity or a substitution on the first not deleted position (if the first operation after deletions was an insertion, then we can pull this insertion left).

- If the first operation is insertion, then we can instead match letter $Y[l_1]$ with the letter $X[l_1]$, and insert the letter one position further - since the inserted letter and $X[l_1]$ are equal this does not change the number of operations nor the result.
- If a sequence of deletions is followed by an identity, then we can instead match letter $Y[l_1]$ with $X[l_1]$ and delete the matched letter $X[l_2]$ for $l_2 > l_1$ (if the sequence was empty then the position of the first operation was not l_1) shifting the left-most first operation to the right.
- If a sequence of deletions was followed by a substitution of letter $X[l_2]$, then we can instead match letter $Y[l_1]$ with $X[l_1]$ and delete letter $X[l_2]$ obtaining a set of operations of smaller size - a contradiction with the minimality of the set.

For the recursive part the same proof follows the same way after applying the first operation. ◀

Full proof of Lemma 5. (⇒) Let Y' be a prefix of Y at edit distance at most k from X. In particular there exists a sequence of at most k operations that transform X into Y'. Using this sequence of edit operations, we find two nodes u and v in $T^{(k)}$ with labels (X, ℓ_1) and (Y, ℓ_2) respectively, satisfying the conditions of the statement.

Recall that T denotes the compacted trie of D. We start our search at the root of T with two empty lists ℓ_1 and ℓ_2. By Lemma 4 we can safely assume that the position of the first edit operation used to change X into Y' is $l = \mathsf{LCP}(X, Y') + 1 = \mathsf{LCP}(X, Y) + 1$. The letter at position l in X is changed: it is deleted; substituted with the l-th letter of Y'; or the l-th letter of Y' is inserted right before this letter.

Let us first assume that neither X is a prefix of Y nor Y' is a prefix of X (or that we do not have any more edit operations to apply).

We know that in T the nodes with labels X and Y have their lowest common ancestor at depth $l - 1$. Now, assume, that the heavy child of this common ancestor is reached from its parent through an edge labeled with letter a. Now depending on the operation performed:

- If the operation is a substitution, then if $X[l] \neq a$, add l_S to ℓ_1, and if $Y[l] \neq a$, add l_S to ℓ_2 (possibly both lists gain the same element).
- If the operation is a deletion, then if $X[l] \neq a$, add l_D to ℓ_1, otherwise ($Y[l] \neq a$), add l_I to ℓ_2.
- If the operation is an insertion, then, if $Y[l] = a$ ($X[l] \neq a$), add l_I to ℓ_1, otherwise ($Y[l] \neq a$), add l_D to ℓ_2.

Now the nodes with labels (X, ℓ_1) and (Y, ℓ_2) belong to the subtree of a node at depth $l-1$ in $T^{(1)}$ (and hence also in $T^{(k)}$), and actually if the added label was not l_D also of the node at depth l (the heavy child of the one at depth $l-1$), this ensures that the next element will be no smaller than $\max^+(\ell_1 \cup \ell_2)$. Furthermore, we know that the edit distance between X and Y', with the operations in lists ℓ_1 and ℓ_2 applied, is smaller by 1 than the distance between X and Y'.

We iterate this technique at most k times walking down the tree. At step $x \leq k$ we start our work at a node of $T^{(x-1)}$ (a node of $T^{(k)}$ that was there before the x-th step of the k-errata construction) that is an ancestor of nodes with labels (X, ℓ_1) and (Y, ℓ_2). After at most k steps we get to a node in $T^{(k)}$, such that its descendant with label (X, ℓ_1) is an ancestor of the node with label (Y, ℓ_2) as no further edit operations remain. Y' turns out to the prefix of Y with the last $d(v) - d(u)$ letters removed.

Now consider the case where, after applying $x-1$ operations, the nodes with labels (X, ℓ_1) and (Y, ℓ_2), respectively, are already in an ancestor descendant relationship, and there are still operations changing X into Y' not processed yet.

If a node with label (X, ℓ_1) is a (non-trivial) descendant of node with label (Y, ℓ_2) the only possible next operation is deletion (we need to remove the suffix of X). If the first edge used to reach the node with label (X, ℓ_1) from the node with label (Y, ℓ_2) is heavy in $T^{(x-1)}$ we can still add l_I to ℓ_2 (due to the extra copy of labeled nodes), otherwise a node with label $(X, \ell_1 \cup \{l_D\})$ is a descendant of the node with label (Y, ℓ_2) in $T^{(x)}$ of smaller depth. We apply this until we reach a single node with both labels (X, ℓ_1) and (Y, ℓ_2).

If a node with label (X, ℓ_1) is a (non-trivial) ancestor of a node with label (Y, ℓ_2), we already know that the claim is true (for some prefix Y'' of Y, that is at smaller edit distance from X than Y'). We may still want to pair X with a longer (or a shorter) prefix of Y, in particular the longest possible. If the next applied operation is an insertion (we want to make the prefix longer) of letter $Y[l]$, then this is symmetric to the case considered in the previous paragraph. We do not provide any modification that allows applying deletion in this case since, in our main application, we care for the longest prefixes only. Still, we know, that the shortest prefix of Y we could obtain this way would be $k - x + 1$ letters shorter than Y'', and hence every such prefix is actually represented in $T^{(k)}$.

In case of Hamming distance the set of operations consists solely of substitutions, and hence the proof reduces naturally (only elements l_S are inserted to the lists, hence only copies of labels with such lists are considered).

($\Leftarrow$) This is the reversal of the forward ($\Rightarrow$) proof construction, that is, we change ℓ_1 and ℓ_2 into a single list of edit operations.

In the Hamming distance case, the set of mismatches between X and $Y[1\mathinner{.\,.}|X|]$ is contained in the set $\{l : l_S \in \ell_1 \cup \ell_2\}$, hence X and a prefix of Y are at Hamming distance at most $|\ell_1 \cup \ell_2| \leq k$ ($|\ell_1 \cup \ell_2|$ may be greater than the actual Hamming distance if a suboptimal pair of labeled nodes is chosen, but this is not a problem). In the edit distance case, the construction is more complicated due to the shifts made by insertions and deletions.

We have two nodes u and v with labels (X, ℓ_1) and (Y, ℓ_2), respectively, such that u is an ancestor of v. Let Y' be the prefix of Y of length $|Y| - d(v) + d(u) = d(u) + \mathsf{surpl}(\ell_2)$ assuming that $\ell_2 = \ell_2^{d(u)}$ ($l \leq d(u)$ for $l_E \in \ell_2$) – otherwise we can remove those larger elements as those only affect the part of Y that does not play a role in the transformation of X into Y' (there exists a descendant of u with label $(Y, \ell_2^{d(u)})$ which we can choose as our v). We know that after applying operations from ℓ_1 to X and operations from ℓ_2 to Y' we obtain the same string: the one obtained by reading the path from root to u in $T^{(k)}$.

Let $\ell = \ell_1 \cup \ell_2$, and for each element of ℓ we mark whether it comes from ℓ_1, from ℓ_2, or from both. For ℓ it may actually be the case that both l_S and l_I appear (coming from two different lists ℓ_1, ℓ_2); this will not be that much of a problem however since this is counted as two edit operations in $|\ell|$. We continue by showing that in fact ℓ encodes a transformation of X into Y' with the use of at most $|\ell|$ edit operations.

We read ℓ and modify X step by step to reach Y' keeping an invariant that when element l_E is about to be processed the first $l-1+i$ positions of the modified string X are equal to the first $l-1+i$ positions of Y, where the value i represents the imbalance between deletion and insertion operations from ℓ_2 already applied, and hence it starts from 0. We have that $i = \mathsf{surpl}(\ell_2')$, where ℓ_2' is the prefix of ℓ_2 storing operations already applied - $\ell_2' = \ell_2^l$ when we move to the node at depth l for the first time.

- If the element is l_S, we substitute the letter at position $l+i$ of the current version of X with $Y[l+i]$.
- If the element l_D comes from both lists ℓ_1 and ℓ_2 (multiple l_D in both lists are matched into pairs), we replace the letter at position $l+i$ by $Y[l+i]$ and increase i by 1 (deletion of the corresponding letters of both strings is equivalent to a substitution, but the depth of the node decreases by one).
- If the element l_D comes from list ℓ_1 (a surplus in pairing), we remove the letter at position $l+i$ in the current version of X.
- If the element l_D comes from list ℓ_2 (a surplus in pairing), we add letter $Y[l+i]$ between letters at positions $l+i-1$ and $l+i$ in X and increase i by 1.
- If the element l_I comes from both lists we do not modify X, and instead decrease i by 1 (inserting the same letter at corresponding positions in X and Y does not change the edit distance).
- If the element l_I comes from list ℓ_1, we insert letter $Y[l+i]$ between letters at positions $l+i-1$ and $l+i$ in X.
- If the element l_I comes from list ℓ_2, we delete the letter at position $l+i$ in X and decrease i by 1.

Notice that when i decreases, the value l of the next l_E must be strictly greater, hence the sum $l+i$ never decreases between operations; and when $l+i$ increases, $X[l+i]$ and $Y[l+i]$ (for X after the changes applied and the old value of $l+i$) must actually be equal.

Further notice that, as claimed, the prefix Y' obtained from X by applying those edit operations (obtained from ℓ_1 and $\ell_2^{d(u)}$) has length equal to $d(u) + \mathsf{surpl}(\ell_2^{d(u)})$. ◀

Proof of Lemma 8. Let $|T^{(k)}|$ denote the size of the k-errata of T. The proof proceeds by induction. $|T^{(0)}| = |T| \leq 3n$.

Assume, that $|T^{(k-1)}| \leq c_{n,k-1} \cdot n(k-1)!(c_\delta \log n)^{k-1}$, for a constant $3 \leq c_{n,k-1} \leq 16$, depending only on n and k, to be specified later.

A single node or label can be copied at most $c_\delta \log |T^{(k-1)}| + 1$ times, since the heavy-light decomposition of the $(k-1)$-errata tree is weighted by the number of explicit nodes and labels. Note that c_δ comes from the number of copies made, and $+1$ comes from the extra copy of a labeled node that is not a leaf.

Now $c_\delta \log |T^{(k-1)}| + 1 \leq c_\delta \log[c_{n,k-1} \cdot n(k-1)!(c_\delta \log n)^{k-1}] + 1 \leq c_\delta[\log c_{n,k-1} + \log n + (k-1)\log k + (k-1)[\log c_\delta + \log n]] + 1 \leq c_\delta[k(\log n + \log k + 7)] = c_\delta k \log n[1 + \frac{\log k + 7}{\log n}]$.

Hence, $|T^{(k)}| \leq |T^{(k-1)}| \cdot (c_\delta \log |T^{(k-1)}| + 1) \leq c_{n,k-1} \cdot n(k-1)!(c_\delta \log n)^{k-1} \cdot c_\delta k \log n[1 + \frac{\log k + 7}{\log n}] = c_{n,k-1} \cdot [1 + \frac{\log k + 7}{\log n}] \cdot [nk!(c_\delta \log n)^k]$.

Hence for $c_{n,k} = c_{n,k-1} \cdot [1 + \frac{\log k + 7}{\log n}]$ the property $|T^{(k)}| \leq c_{n,k} \cdot nk!(c_\delta \log n)^k$ is satisfied, and it remains to show, that $c_{n,k} \leq 16$ if $k \leq \frac{\log n}{2 \log \log n}$. $c_{n,0} = 3, c_{n,k} \leq 3 \cdot [1 + \frac{\log k + 7}{\log n}]^k$.

Recall, that $(1 + \frac{1}{x})^x \to e$, and that for $k \leq \frac{\log n}{2 \log \log n}$ we have $\frac{\log k + 7}{\log n} \leq \frac{2 \log \log n}{\log n}$ (for large enough n), hence $c_{n,k} \leq 3e < 16$.

The bound on the number of copies of a single label follows analogously. ◀