44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2024, December 16–18, 2024, Gandhinagar, Gujarat, India

Edited by

Siddharth Barman
Sławomir Lasota

LIPICS

LIPIcs – Vol. 323 – FSTTCS 2024 www.dagstuhl.de/lipics

Editors

Siddharth Barman
Indian Institute of Science, Bangalore, India
barman@iisc.ac.in

Sławomir Lasota
University of Warsaw, Poland
s.lasota@uw.edu.pl

ACM Classification 2012
Theory of computation; Computing methodologies; Software and its engineering

ISBN 978-3-95977-355-3

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/978-3-95977-355-3.

Publication date
December, 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.FSTTCS.2024.0

ISBN 978-3-95977-355-3 **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 Papers

Regular Papers

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024). The conference was held as an in-person event during the period December 16–18, 2024, on the campus of IIT Gandhinagar in Gujarat, India.

The conference has two tracks: Track A focusing on algorithms, complexity and related issues, and Track B focusing on logic, automata and other formal method aspects of computer science. Each track had its own Program Committee (PC) and chair (Siddharth Barman for Track A and Sławomir Lasota for Track B). This volume constitutes the joint proceedings of the two tracks, published in the LIPIcs series under a Creative Common license, with free online access for all.

The conference comprises of 6 invited talks, 21 contributed talks in Track A, and 11 contributed talks in Track B. This volume contains all the contributed papers from the two tracks, and two articles accompanying two of the invited talks. We thank all the authors who submitted their papers to FSTTCS 2024. We are especially grateful to the PC members for their tireless work, and all the external reviewers for their expert opinion in the form of timely reviews.

We thank all the invited speakers for accepting our invitation: Pankaj K. Agarwal (Duke University), Suguman Bansal (Georgia Institute of Technology), Ioannis Caragiannis (Aarhus University), Dmitry Chistikov (University of Warwick), Uriel Feige (Weizmann Institute), Sebastian Siebertz (Universität Bremen).

The main conference was co-located with four workshops: Milestones and Motifs in the Theory of Proofs, Algebraic Computation, and Lower Bounds (organised by S. Akshay, Olaf Beyersdorff, Nutan Limaye, and Prajakta Nimbhorkar), Research Highlights in Programming Languages (organised by Madhukar Kumar, Divyesh Unadkat, and Abhisekh Sankaran), Automata and Games for Synthesis (organised by Sougata Bose and K.S. Thejaswini), and Workshop on Algorithmic Mechanism Design (organised by Umang Bhaskar and Meghana Nasre).

We are indebted to the organizing committee for making all the necessary arrangements for the conference: the chair Neeldhara Misra (IIT Gandhinagar), Abhishek Bichhawat (IIT Gandhinagar), Bireswar Das (IIT Gandhinagar), Anirban Dasgupta (IIT Gandhinagar), Manoj Gupta (IIT Gandhinagar), Balagopal Komarath (IIT Gandhinagar), and Manisha Padala (IIT Gandhinagar); and to past organizers for knowledge transfer.

We thank S.P. Suresh (CMI, Chennai) for maintaining the conference web page. We are grateful to the friendly staff at Dagstuhl LIPIcs for helping us put together the proceedings, in particular to Michael Didas and Michael Wagner. Finally, we thank the members of the Steering Committee, especially Amit Kumar, and the PC Chairs from FSTTCS 2023 (Patricia Bouyer and Srikanth Srinivasan), for providing pertinent information and valuable advice about various aspects of the conference.

Siddharth Barman and Sławomir Lasota
October 2024

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Programme Committee

Track A

- Aritra Banik (National Institute of Science Education and Research)
- Siddharth Barman (Indian Institute of Science) – Track A Chair
- Umang Bhaskar (Tata Institute of Fundamental Research)
- Diptarka Chakraborty (National University of Singapore)
- Debarati Das (Pennsylvania State University)
- Klim Efremenko (Ben-Gurion University)
- Sushmita Gupta (The Institute of Mathematical Sciences)
- Rohit Gurjar (Indian Institute of Technology Bombay)
- Arindam Khan (Indian Institute of Science)
- Neeldhara Misra (Indian Institute of Technology, Gandhinagar)
- Rajat Mittal (Indian Institute of Technology, Kanpur)
- Kamesh Munagala (Duke University)
- Prajakta Nimbhorkar (Chennai Mathematical Institute)
- Manisha Padala (Indian Institute of Technology, Gandhinagar)
- Jaikumar Radhakrishnan (Tata Institute of Fundamental Research)
- Nidhi Rathi (Max Planck Institute for Informatics, University of Saarland)
- Ramprasad Saptharishi (Tata Institute of Fundamental Research)
- Sahil Singla (Georgia Tech)
- Makrand Sinha (University of Illinois Urbana-Champaign)
- Seeun William Umboh (The University of Melbourne)
- Santhoshini Velusamy (Toyota Technological Institute at Chicago)
- Michał Włodarczyk (University of Warsaw)
- Meirav Zehavi (Ben-Gurion University)

Track B

- Parosh Aziz Abdulla (Uppsala University)
- C. Aiswarya (Chennai Mathematical Institute)
- S. Akshay (Indian Institute of Technology Bombay)
- Christel Baier (Technical University Dresden)
- Laure Daviaud (University of East Anglia)
- Sibylle Fröschle (Technical University Hamburg)
- Blaise Genest (CNRS)
- Stefan Haar (INRIA, France)
- Christoph Haase (University of Oxford)
- Petr Jančar (Palacky University, Olomouc)
- Ismaël Jecker (University of Franche-Comté)
- Edon Kelmendi (Queen Mary University of London)
- Sławomir Lasota (University of Warsaw) – Track B Chair
- Ranko Lazic (University of Warwick)
- Christof Löding (RWTH Aachen)
- Meena Mahajan (The Institute of Mathematical Sciences)
- Jean-Francois Raskin (Université Libre de Bruxelles)
- Nathalie Sznajder (Sorbonne University)

List of External Reviewers

Sheikh Shakil Akhtar
Tatsuya Akutsu
Yan Hong Yao Alvin
Ashwani Anand
Nikhil Balaji
Sayan Bandyapadhyay
Ioana Bercea
C. S. Bhargav
Vishwas Bhargava
Sujoy Bhore
Václav Blažej
Sougata Bose
Joshua Brody
Márcia Cappelle
Philip Cervenjak
Harish Chandramouleeswaran
Abhranil Chatterjee
Prerona Chatterjee
Juhi Chaudhary
Vera Chekan
Lijie Chen
Yu Chen
Andrew Childs
Yogesh Dahiya
Bireswar Das
Quentin Deschamps
Palash Dey
Sanjana Dey
Shyam Dhamapurkar
Prateek Dwivedi
Henning Fernau
Johannes K. Fichte
Simon Forest
Hadar Frenkel
Arnab Ganguly
Mohit Garg
Pratik Ghosal
Prantar Ghosh
Sumanta Ghosh
Christian Glasser
Mayank Goswami
R. Govind
Tomer Grossman
Nikhil Gupta
Waldo Gálvez
Léo Henry
Chien-Chung Huang
Tanmay Inamdar
Dmitry Itsykson
Pallavi Jain
Rahul Jain
Shweta Jain
Satyabrata Jana
Stacey Jeffery
Agastya Vibhuti Jha
John Kallaugher
Lawqueen Kanesh
Debajyoti Kar
Jun Kawahara
Chandrima Kayal
Dominik Kempa
Sudeshna Kolay
Annamaria Kovacs
László Kozma
Vaibhav Krishan
Pooja Kulkarni
Rucha Kulkarni
Gunjan Kumar
Rajendra Kumar
Srijita Kundu
Abhiruk Lahiri
Michael Levet
Ramanujan M. Sridharan
Gopinath Mishra
Parth Mittal
Sounak Modak
Anish Mukherjee
Saraswati Nanoti
Daniel Neuen
Jakob Nogler
Manaswi Paraashar
Pan Peng
Geevarghese Philip
Milind Prabhu
Aditya Prakash
Yuanyuan Qi
Tim Quatmann
C. Ramya
Arka Ray
Aravind Reddy

Alexis Reynouard
Neha Rino
Mohammad Roghani
Bodhayan Roy
Aravinda Kanchana Ruwanpathirana
Abhishek Sahu
Mohammad Saneian
Arnaud Sangnier
Mathieu Sassolas
Srinivasa Rao Satti
Saket Saurabh
Aleksy Schubert
Sayantan Sen
Olivier Serre
Aditi Sethia
Roohani Sharma
Suhail Sherif
Mihaela Sighireanu
Sandeep Silwal
Kirill Simonov
Benjamin Smith
Joachim Spoerhase
Léo Stefanesco
Rafał Stefański
Madhu Sudan
S. P. Suresh
Prafullkumar Tale
Anamay Tengse
Raghunath Tewari
Neekon Vafa
Rohit Vaish
Leo van Iersel
Erik Jan van Leeuwen
Pierre Vandenhove
Nithin Varma
Manolis Vasilakis
Patrick Wienhöft
Jie Xue
Chuanqi Zhang

List of Authors

Étienne André (3)
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France; Institut Universitaire de France (IUF), Paris, France

Johan Arcile (3)
IBISC, Univ Evry, Université Paris-Saclay, 91025 Evry, France

V. Arvind (4)
The Institute of Mathematical Sciences (HBNI), Chennai, India;
Chennai Mathematical Institue, India

Ali Asadi (5)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Nikhil Ayyadevara (6)
University of Michigan, Ann Arbor, MI, USA

Aritra Banik (7)
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, India

Benoît Barbot (8)
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Dylan Bellier (9)
Univ Rennes, IRISA, CNRS, France

Massimo Benerecetti (9)
Università degli Studi di Napoli Federico II, Italy

Vishwas Bhargava (10)
Department of Computing and Mathematical Sciences, Caltech, Pasadena, CA, USA

Sudatta Bhattacharya (11)
Charles University, Prague, Czech Republic

Sujoy Bhore (12)
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Markus Bläser (13)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Alberto Bombardelli (14)
Fondazione Bruno Kessler, Trento, Italy

Patricia Bouyer (8)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190 Gif-sur-Yvette, France

Laura Bozzelli (14)
University of Napoli “Federico II”, Italy

Damien Busatto-Gaston (15)
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Deeparnab Chakrabarty (16)
Dartmouth College, Hanover, NH, USA

L. Sunil Chandran (17)
Indian Institute of Science, Bengaluru, India

Krishnendu Chatterjee (5)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Sravanthi Chede (18)
Indian Institute of Technology Ropar, Rupnagar, India

Leroy Chew (18)
TU Wien, Austria

Ashish Chiplunkar (6)
Indian Institute of Technology, New Delhi, India

Dmitry Chistikov (1)
Centre for Discrete Mathematics and its Applications (DIMAP) &, Department of Computer Science, University of Warwick, UK

Arjan Cornelissen (19)
Simons Institute for the Theory of Computing, University of California, Berkeley, CA, USA;
IRIF – CNRS, Paris, France

Bireswar Das (20)
Indian Institute of Technology Gandhinagar, India

Sayani Das (7)
Theoretical Computer Science, The Institute of Mathematical Sciences, Chennai, India

Samir Datta (4)
Chennai Mathematical Institute and UMI ReLaX, India

Sanjana Dey (11)
National University of Singapore, Singapore

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Kyveli Doveri (21)
University of Warsaw, Poland

Julian Dörfler (13)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Sarfaraz Equbal (12)
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Bernd Finkbeiner (22)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Rishikesh Gajjala (17)
Indian Institute of Science, Bengaluru, India

Karthik Gajulapalli (23)
Georgetown University, Washington, DC, USA

Pierre Ganty (21)
IMDEA Software Institute, Pozuelo de Alarcón, Madrid, Spain

Jinia Ghosh (20)
Indian Institute of Technology Gandhinagar, India

Elazar Goldenberg (11)
The Academic College of Tel-Aviv-Yaffo, Israel

Sushmita Gupta (24)
The Institute of Mathematical Sciences, HBNI, Chennai, India

Rohit Gurjar (12)
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Serge Haddad (8)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190 Gif-sur-Yvette, France

Nathaniel Harms (25)
EPFL, Lausanne, Switzerland

Furio Honsell (26)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Tanmay Inamdar (24)
Indian Institute of Technology Jodhpur, Jodhpur, India

Felix Jahn (22)
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Pallavi Jain (24)
Indian Institute of Technology Jodhpur, Jodhpur, India

Gorav Jindal (13)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Arindam Khan (27)
Indian Institute of Science, Bengaluru, India

Asif Khan (4)
Chennai Mathematical Institute, India

Christian Komusiewicz (28)
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Christian Konrad (29)
University of Bristol, UK

Michal Koucký (11)
Charles University, Prague, Czech Republic

Anant Kumar (20)
Indian Institute of Technology Gandhinagar, India

Engel Lefaucheux (3)
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Marina Lenisa (26)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Zeyong Li (23)
National University of Singapore, Singapore

Hang Liao (16)
Dartmouth College, Hanover, NH, USA

Daniel Lokshtanov (24)
University of California Santa Barbara, CA, USA

Anand Louis (30)
Indian Institute of Science, Bengaluru, India

Anil Maheshwari (7)
School of Computer Science, Carleton University, Ottawa, Canada

Nikhil S. Mande (19)
University of Liverpool, UK

Bubai Manna (7)
Department of Mathematics, Indian Institute of Technology Kharagpur, India

Rogers Mathew (31)
Department of Computer Science and Engineering, IIT Hyderabad, India

Andrew McGregor (29)
University of Massachusetts Amherst, MA, USA

Shravan Mehra (17)
Indian Institute of Science, Bengaluru, India;
University of Birmingham, UK

Fabio Mogavero (9)
Università degli Studi di Napoli Federico II, Italy

Subhas C. Nandy (7)
Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India

Alantha Newman (30)
Université Grenoble Alpes, France

Youssouf Oualhadj (15)
Univ Paris Est Creteil, LACL, F-94010 Creteil, France; CNRS, ReLaX, IRL 2000, Siruseri, India

Fahad Panolan (24, 31)
School of Computer Science,
University of Leeds, UK

Subhasree Patro (19)
Technische Universiteit Eindhoven, The Netherlands; Centrum Wiskunde en Informatica (QuSoft), Amsterdam, The Netherlands

Sophie Pinchinat (9)
Univ Rennes, IRISA, CNRS, France

Pierre Popoli (32)
Department of Mathematics,
University of Liège, Belgium

Krishna Priya K. M. (7)
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, India

Saladi Rahul (17)
Indian Institute of Science, Bengaluru, India

Shanthanu S. Rai (33)
Tata Institute of Fundamental Research, Mumbai, India

Arka Ray (30)
Indian Institute of Science, Bengaluru, India

Artur Riazanov (25)
EPFL, Lausanne, Switzerland

Bodhayan Roy (7)
Department of Mathematics, Indian Institute of Technology Kharagpur, India

Sasanka Roy (7)
Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India

Abhishek Sahu (7)
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, India

Raimundo Saona (5)
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Saket Saurabh (24)
The Institute of Mathematical Sciences, HBNI, Chennai, India;
University of Bergen, Norway

Ivan Scagnetto (26)
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Jannik Schestag (28)
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Rik Sengupta (29)
IBM Research, Cambridge, MA, USA;
University of Massachusetts Amherst, MA, USA

Seshikanth (31)
Department of Computer Science and Engineering, IIT Hyderabad, India

Jeffrey Shallit (32)
School of Computer Science,
University of Waterloo, Canada

Amatya Sharma (6)
University of Michigan, Ann Arbor, MI, USA

Shivdutt Sharma (4)
Indian Institute of Information Technology, Una, India

Anil Shukla (18)
Indian Institute of Technology Ropar, Rupnagar, India

Julian Siber (22)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Sebastian Siebertz (2)
University of Bremen, Germany

B. Srivathsan (21)
Chennai Mathematical Institute, India;
CNRS IRL 2000, ReLaX, Chennai, India

Manon Stipulanti (32)
Department of Mathematics,
University of Liège, Belgium

Aditya Subramanian (27)
Indian Institute of Science, Bengaluru, India

Jakub Svoboda (5)
Institute of Science and Technology Austria
(ISTA), Klosterneuburg, Austria

César Sánchez (14)
IMDEA Software Institute, Madrid, Spain

Anamay Tengse (10)
School of Computer Sciences, NISER,
Bhubaneswar, India

Cuong Than (29)
University of Massachusetts Amherst, MA, USA

Léo Tible (15)
Univ Paris Est Creteil, LACL, F-94010 Creteil,
France

Stefano Tonetta (14)
Fondazione Bruno Kessler, Trento, Italy

Daniele Varacca (15)
Univ Paris Est Creteil, LACL, F-94010 Creteil,
France

Yadu Vasudev (4)
Indian Institute of Technology Madras,
Chennai, India

Shankar Ram Vasudevan (4)
Chennai Mathematical Institute, India

Alexandre Vigny (2)
University Clermont Auvergne, France

Ilya Volkovich (23)
Boston College, MA, USA

Tobias Widmann (27)
Technical University of Munich, Germany

Andreas Wiese (27)
Technical University of Munich, Germany

Yinfeng Zhu (34)
Institute of Natural Sciences and Mathematics,
Ural Federal University, Ekaterinburg, Russia

An Introduction to the Theory of Linear Integer Arithmetic

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, UK

Abstract

Presburger arithmetic, or linear integer arithmetic (LIA), is a logic that allows one to express linear constraints on integers: equalities, inequalities, and divisibility by nonzero $n \in \mathbb{Z}$. More formally, it is the first-order theory of integers with addition and ordering. This paper offers a short introduction: what can be expressed in this logical theory, decision problems, and automated reasoning methods.

We begin with an elementary introduction, explaining the language of linear arithmetic constraints by examples. We adopt a theoretical perspective, focusing on the decision problem: determining the truth value of a logical sentence. The following three views on Presburger arithmetic give us three effective methods for decision procedures: a view from geometry (using semi-linear sets), from automata theory (using finite automata and recognizable sets), and from symbolic computation (using quantifier elimination).

The decision problem for existential formulas of Presburger arithmetic is essentially the feasibility problem of integer linear programming. By a fundamental result due to Borosh and Treybig [*Proc. Am. Math. Soc.* 55(2), 1976] and Papadimitriou [*J. ACM* 28(4), 1981], it belongs to the complexity class NP. Echoing the three views discussed above, we sketch three proofs of this result and discuss how these ideas have been used and developed in the recent research literature.

This is a companion paper for a conference talk focused on the three views on Presburger arithmetic and their applications. The reader will require background knowledge at the level of undergraduate computer science curricula. The discussion of complexity aspects is more advanced.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Logic

Keywords and phrases Logical theories of arithmetic, decision procedures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.1

Category Invited Paper

Funding Supported in part by the Engineering and Physical Sciences Research Council [EP/X03027X/1].

Acknowledgements I would like to thank Christoph Haase for introducing me to arithmetic theories and all my co-authors for sharing inspiration, ideas, and kindness. I am also grateful to Ranko Lazić and Shrisha Rao for useful discussions of the Steinitz lemma and its applications. Last but not least, I would like to thank everyone who gave feedback on early drafts of this paper.

1 What is linear integer arithmetic?

Linear integer arithmetic, also known as *Presburger arithmetic,* combines linear Diophantine equations with logic (Boolean connectives and quantifiers). Intuitively, a logic is a language in which we can express things that are true or false. In the language of Presburger arithmetic we can:

- talk about integers (referred to as variables x, y, $\dots$),
- assert linear inequalities involving these integers,

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 1; pp. 1:1–1:36
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

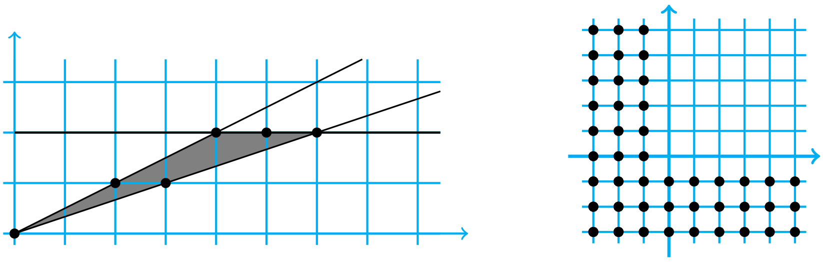


Figure 1 Black dots show integer points (assignments to x, y) that make the formulas in Eqs. (1) and (2) true. These assignments form sets T (left) and U (right), respectively. On the left, the set of solutions in $\mathbb{R}^2$ to the system of 3 inequalities in Eq. (1) is the intersection of 3 half-planes, shown in grey.

- form Boolean combinations (logical AND, OR, NOT, denoted by $\wedge$, $\vee$, $\neg$, respectively) of these assertions, and
- quantify over (all) integers, using "for all" ($\forall$) and "there exists" ($\exists$).

We develop the intuition first, deferring rigorous definitions to Section 2.

A *formula* in linear integer arithmetic is a syntactic object. A formula expresses (*defines*) a set of points with integer coordinates, that is, a subset of $\mathbb{Z}^d$ for some d. For example, the formula

$$(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2) \tag{1}$$

defines the set $T = \{(0,0),(2,1),(3,1),(4,2),(5,2),(6,2)\}$ of integer points in a triangle. Indeed, $\{(x,y) \in \mathbb{Z}^2 : (x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)\} = T$, see Fig. 1 (left). Informally, the formula talks about x and y, and thus defines a set in 2D. A set may be equivalently expressed by more than one logical formula. For example, the two formulas

$$(x < 0) \vee (y < 0) \qquad \text{and} \qquad \neg((x \geqslant 0) \wedge (y \geqslant 0)) \tag{2}$$

define the same set, call it U, depicted in Fig. 1 (right).

Let us consider some formulas with quantifiers. As an elementary example, the formula

$$\forall x\ [(\exists y\ (x = 2y)) \vee (\exists z\ (x = 2z + 1))] \tag{3}$$

can be interpreted as saying that every integer is either even or odd (possibly both). Notice that we could equivalently write $\forall x\ [(\exists y\ (x = 2y)) \vee (\exists y\ (x = 2y + 1))]$, because the choice of name (y or z) for the auxiliary variable does not change the meaning of the formula, as long as there is no "name clash": usage of a name that is already in use at this point in the formula.

Formally, variables that a formula φ "talks about" are called *free variables* of φ. For example, the formula

$$E(x){:} \quad \exists y\ (x = 2y) \tag{4}$$

talks about a single variable, x, and asserts that x is even. Note that we denoted this formula $E(x)$. More generally, we write, for instance, $\varphi(x, y, z)$ implying that φ has no free variables except x, y, and z. It is *not* implied that x, y, and z are mentioned by φ: some or even all of them may not appear in it. In the formula from Eq. (3), all variables are "quantified away", that is, these formulas have no free variables. Each of these formulas evaluates to just true or false. Formulas without free variables are called *sentences*.

▶ **Example 1.** Fix two integers $a, b > 0$. Consider the sentence

$$C_{a,b}\colon \quad \forall s\ \exists x_1\ \exists x_2\ \ s = ax_1 - bx_2.$$

We will see in Section 3 that $C_{a,b}$ is true if and only if a and b are coprime. Intuitively, $C_{a,b}$ is implied by, and in fact equivalent to, a similar formula $\forall s_1\ \forall s_2\ \exists x_1\ \exists x_2\ \ s_2 - s_1 = ax_1 - bx_2$. Rearranging the terms, we can rewrite the equation as $s_1 + ax_1 = s_2 + bx_2$. So the formula in fact asserts that the two arithmetic progressions $s_1, s_1+a, s_1+2a, \ldots$ and $s_2, s_2+b, s_2+2b, \ldots$ always have a number appearing in both, no matter how the initial terms s_1 and s_2 are chosen. ⌟

▶ Remark. In Example 1, a and b are fixed parameters. If they were, in fact, variables, then the given formula would not be a sentence and would talk about a and b. Denote it $C(a, b)$. Assuming $a, b > 0$, the formula $C(a, b)$ would still assert that a and b are co-prime, but would no longer be a formula of linear integer arithmetic, because of the multiplication of two variables a and x_1. Logicians would say that $C(a, b)$ is a formula in the language of rings.

All variables are always quantified over the same set, namely $\mathbb{Z}$. That is, the syntax of Presburger arithmetic disallows formulas such as $\forall x \in S\ \varphi(x)$. Can we still express such an assertion in our logic? Assuming that the set S itself can be defined in Presburger arithmetic, namely as $S = \{a \in \mathbb{Z} : \psi(a) \text{ is true}\}$ for some formula ψ with one free variable, we can write $\forall x\ (\neg\psi(x) \lor \varphi(x))$ or, equivalently, $\forall x\ (\psi(x) \to \varphi(x))$, where $\to$ denotes logical implication. Presburger arithmetic can quantify over individual numbers (*first-order* quantification), but not over sets, relations, etc.

Some assertions cannot be written as a finite Boolean combination of linear inequalities without the help of quantifiers. An example is the formula $E(x)$ from Eq. (4). It is thus often convenient to extend the syntax of the logic by assertions such as "x is even", "$x - 3y + 5$ is divisible by 7", etc. We can write them as divisibility constraints $k \mid \ldots$, reading "k divides $\ldots$", where k is a fixed nonzero integer (that is, k cannot be a variable or an expression involving variables). Yet another alternative syntax is congruences $t_1 \equiv t_2 \pmod{k}$, with t_1, t_2 linear functions. Such constraints, also known as modulo constraints, can be used alongside linear inequalities as basic building blocks of linear integer arithmetic.

▶ **Example 2.** By the Chinese remainder theorem, formula

$$(x \equiv 2 \,(\mathrm{mod}\ 3)) \land (x \equiv 1 \,(\mathrm{mod}\ 5)) \land (x \equiv 3 \,(\mathrm{mod}\ 7)) \land (x \geqslant 0)$$

defines the set of natural numbers congruent to 101 modulo $105 = 3 \cdot 5 \cdot 7$. An equivalent formula avoids modulo constraints at the cost of extra quantified variables: $\exists u\, \exists v\, \exists w\ (x - 2 = 3u) \land (x - 1 = 5v) \land (x - 3 = 7w) \land (x \geqslant 0)$. ⌟

Linear integer arithmetic, and logical theories of arithmetic more generally (not necessarily linear), provide a common framework for expressing problems from various domains: for example, many classical combinatorial optimisation problems can be encoded directly.

▶ **Example 3.** The subset sum problem asks, given natural numbers $a_1, \ldots, a_n$, whether there exists a subset that sums up to a given target, t:

$$\exists x_1 \ldots \exists x_n \bigwedge_{i=1}^{k} [(x_i = 0) \lor (x_i = 1)] \land \sum_{i=1}^{n} a_i x_i = t.$$

Notice that, since all quantification is over integers, the subformula in square brackets can be rewritten as $[(0 \leqslant x_i) \land (x_i \leqslant 1)]$, making the entire sentence an existentially quantified conjunction of equalities and inequalities. In terms of computational complexity, the subset sum problem is NP-complete [136, Section 7.5]. ⌟

▶ **Example 4.** The Frobenius coin problem asks for the largest whole amount that *cannot* be formed using coins with denominations $a_1, \dots, a_n$, all in unbounded supply [144, 5]. If the greatest common divisor of $a_1, \dots, a_n$ is 1, then such a number exists and is referred to as the Frobenius number of $a_1, \dots, a_n$, denoted $F(a_1, \dots, a_n)$. For every $x \in \mathbb{Z}$, we have $F(a_1, \dots, a_n) \leqslant x$ if and only if the following formula of Presburger arithmetic is true:

$$\Phi(x): \quad \forall y \, \exists x_1 \dots \exists x_n \, \left[(y \leqslant x) \vee \left(\bigwedge_{i=1}^{k} (x_i \geqslant 0) \wedge \left(\sum_{i=1}^{n} a_i x_i = y \right) \right) \right].$$

Thus, $F(a_1, \dots, a_n)$ is the number that *satisfies* (makes true) the formula $\Phi(x) \wedge \neg\Phi(x-1)$ or, equivalently, the formula

$$\Phi(x) \wedge \forall z_1 \dots \forall z_n \, \left[\bigvee_{i=1}^{k} (z_i < 0) \vee \left(\sum_{i=1}^{n} a_i z_i < x \right) \vee \left(\sum_{i=1}^{n} a_i z_i > x \right) \right].$$

Deciding whether $F(a_1, \dots, a_n) \leqslant t$ is NP-hard [4] and belongs to the complexity class $\mathrm{coNP}^{\mathrm{NP}} = \Pi_2\mathrm{P}$ (see, e.g., the definition of the polynomial(-time) hierarchy [136, Section 10.3]). ⌟

Decision problems and decision procedures

Is it possible to determine the truth value of a given sentence of Presburger arithmetic? This problem is traditionally referred to as the *decision problem*:

Input: Sentence φ.
Output: Is φ true or false?

There exists an algorithm (a *decision procedure*) that solves the decision problem for Presburger arithmetic; see Section 3. Thus, one says that (the theory of) Presburger arithmetic is *decidable*. This statement requires proof, because quantifiers in φ range over an infinite set, $\mathbb{Z}$.

The existence of algorithms that solve the decision problem for this and for other logics enables the field of *automated reasoning*: we can outsource to a computer the determination of whether a formal mathematical statement (even if expressed in a relatively simple language) is true or false! At the same time, sentences of Presburger arithmetic are not deep mathematical truths: this logic is rather restrictive. Extending the syntax by allowing not just linear but arbitrary polynomial constraints makes the problem undecidable. (This follows from Gödel's incompleteness theorems and Tarski's undefinability theorem. The argument is shown in, e.g., [23, Chapter 17] and [135, Chapter 10].) In fact, even for Presburger arithmetic the worst-case computational complexity of the problem is high, meaning that big input sentences φ may require prohibitively large computation time. Still, Presburger arithmetic offers a useful language for expressing assertions that arise in applications and can be checked in an automated fashion, striking a balance between expressiveness and decidability.

Nowadays powerful software tools are available that implement decision procedures for many logical theories (involving arithmetic or not). A big class of such tools is *satisfiability modulo theories (SMT) solvers*, developed since the early 2000s (see, e.g., [2, 12, 11]). SMT solvers build on the earlier boom of Boolean satisfiability (SAT) solvers, adding to SAT more powerful logics; hence the "modulo [logical] theories" in the name.

The present paper adopts a theoretical perspective, focusing on the pure decision problem as defined above. In practice, the success of software tools depends on many other features taking theoretical ideas further (see, e.g., [26, Chapter 1]). For example, solvers can be asked to produce explanations: proofs that a given sentence is true or false. Also key is incremental solving: when a new constraint is appended to an input sentence, the solver can benefit from information gained in its previous run, instead of restarting from scratch. SMT solvers implement not only algorithms (in the strict sense of the word) for decision problems but also heuristic approaches (for decidable as well as undecidable theories). This way the tools can successfully handle formulas coming from applications with thousands of variables, avoiding the worst-case computational complexity or even undecidability.

2 Syntax and semantics (formal definitions)

We give formal definitions of syntax and semantics of linear integer arithmetic. A description of the syntax of a logic specifies which syntactic expressions are admissible ("belong" to the logic), and the semantics prescribes their meaning. The goal here is not to re-define or revise fundamental mathematical notions, but rather to determine unambiguously:

(syntax) what kind of formulas a decision procedure must be able to handle (as its input);

(semantics) what the correct output (truth value) for each possible input formula is.

Let $\mathcal{V}$ be the set (alphabet) of variables that a formula may use. A small example might have $\mathcal{V} = \{x, y\}$, but in general $\mathcal{V}$ may well be infinite.

Syntax. We first define *terms*. Intuitively, a term in our logic is an expression that can evaluate (under an assignment of values to variables) to an integer. Formally, a term is a formal expression of the form $a_0 + a_1 x_1 + \ldots + a_n x_n$, where $a_0, \ldots, a_n \in \mathbb{Z}$, $x_1, \ldots, x_n \in \mathcal{V}$, and $n \geqslant 0$.

We now define formulas of the logic. Formulas, like terms, are syntactic objects. Unlike terms, the intuition is that a formula should evaluate (again under an assignment of values to variables) to true or false. The definition is in two "stages".

An *atomic formula*, or an *atom*, is either a comparison of the form $t_1 < t_2$, $t_1 \leqslant t_2$, or $t_1 = t_2$, where t_1 and t_2 are terms, or a congruence constraint of the form $t_1 \equiv t_2 \pmod{m}$, where t_1 and t_2 are terms and $m \in \mathbb{Z} \setminus \{0\}$.

▶ Remark. Although this definition excludes comparisons of the form $t_1 > t_2$ and $t_1 \geqslant t_2$, we can always rewrite such comparisons backwards: $t_2 < t_1$ and $t_2 \leqslant t_1$, respectively. Therefore, we can regard symbols $>$ and $\geqslant$ as "syntactic sugar". Similarly, a divisibility constraint of the form $k \mid t$, where $k \in \mathbb{Z} \setminus \{0\}$ and t is a term, can be rewritten as $t \equiv 0 \pmod{k}$.

The main definition is inductive. A *formula* is:

- either an atomic formula,
- or an expression of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\neg\varphi$, where φ and ψ are formulas,
- or an expression of the form $\exists x\ \varphi$ or $\forall x\ \varphi$, where φ is a formula.

Only expressions of the above three kinds are considered formulas. We use brackets to disambiguate the composition of formulas from its sub-formulas. The reader can verify that all formulas from Section 1 are indeed formulas according to this definition.

Note that the third case does not restrict φ, and in particular we do not specify whether or not the variable x bound by the quantifier even appears in φ.

▶ Remark. A reader unfamiliar with mathematical logic may find it convenient to assume **no variable reuse**. That is, whenever a formula Φ contains a subformula $\exists x\ \varphi$ or $\forall x\ \varphi$, we can assume that *all* occurrences of x in Φ are within subformulas of such two forms, and moreover these subformulas cannot contain one another. This convention forbids, e.g., nested quantifiers that bind the same variable.

Semantics. We now show how to "assign meaning" to formulas, which have been defined above as purely syntactic objects.

An *assignment* is a map from $\mathcal{V}$, our alphabet of variables, to $\mathbb{Z}$. Let $\nu\colon \mathcal{V} \to \mathbb{Z}$ be an assignment and take some integer $a \in \mathbb{Z}$ and variable $x \in \mathcal{V}$. By $\nu[a/x]$ we denote another assignment, ν', that agrees with ν on all variables from $\mathcal{V}$ except x and sets $\nu(x)$ to a:

$$\nu'(u) = \begin{cases} \nu(u) & \text{if } u \in \mathcal{V} \setminus \{x\}, \\ a & \text{if } u \text{ is } x. \end{cases}$$

For example, if $\mathcal{V} = \{x, y\}$ and $\nu(x) = \nu(y) = 6$, then for $\nu' = \nu[3/y]$ we have $\nu'(x) = 6$ and $\nu'(y) = 3$.

If t is a term and ν an assignment, then by $\nu(t)$ we denote the value of t under ν. Formally, if t is $a_0+a_1x_1+\ldots+a_nx_n$ where $x_1, \ldots, x_n \in \mathcal{V}$, then $\nu(t) = a_0+a_1\nu(x_1)+\ldots+a_n\nu(x_n)$. Note that $\nu(t)$ is a (specific) integer. In the example from the previous paragraph, $\nu(x-3y+5) = -7$ and $\nu[3/y](x - 3y + 5) = 2$.

We are now ready to assign truth values to formulas, given an assignment. This definition is also inductive, following the inductive definition of a formula. Let ν be an assignment. For every formula we determine whether it *holds under* ν (also: φ *is true on* ν; ν *satisfies* φ):

- Take an atomic formula $t_1 < t_2$, $t_1 \leqslant t_2$, $t_1 = t_2$, or $t_1 \equiv t_2 \pmod{m}$, where t_1 and t_2 are terms and $m \in \mathbb{Z} \setminus \{0\}$. Then $t_1 < t_2$ holds under ν if and only if $\nu(t_1) < \nu(t_2)$. Here $\nu(t_1) < \nu(t_2)$ is just a comparison between two specific numbers from $\mathbb{Z}$. The definition for the cases of $t_1 \leqslant t_2$ and $t_1 = t_2$ is analogous. For the case of $t_1 \equiv t_2 \pmod{m}$, the condition is that $\nu(t_1) - \nu(t_2)$ is a multiple of m.
- A formula $\varphi \wedge \psi$ holds under ν if both φ and ψ hold under ν. For $\varphi \vee \psi$, the condition is that at least one of φ and ψ holds. For $\neg\varphi$, the condition is that φ does not hold.
- A formula $\exists x\ \varphi$ holds under ν if for some $a \in \mathbb{Z}$ the formula φ is true under the assignment $\nu[a/x]$. A formula $\forall x\ \varphi$ holds under ν if for every $a \in \mathbb{Z}$ the formula φ is true under the assignment $\nu[a/x]$.

This definition may appear tautological (and even unnecessary), but it ensures that the meaning of formulas is determined unambiguously.

▶ Remark. In logic, the standard notation for "φ holds under ν" would be "$(\mathbb{Z}, +, \leqslant), \nu \models \varphi$", as the satisfaction relation is really a ternary relation. Here $(\mathbb{Z}, +, \leqslant)$ is the structure that we have fixed throughout: $\mathbb{Z}$ is the domain of discourse (universe), $+\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is binary addition on $\mathbb{Z}$, and $\leqslant$ is the standard "less than or equal to" relation (predicate) on $\mathbb{Z}$. Intuitively, a structure "realises" the syntax of a logic, giving a "valuation" for each symbol.

An occurrence of a variable $x \in \mathcal{V}$ in a formula Φ is *free* if it lies outside all subformulas $\exists x\ \varphi$ and $\forall x\ \varphi$, and *bound* otherwise. Now, $x \in \mathcal{V}$ is a *free variable* of Φ if it has a free occurrence, and a *bound variable* otherwise. A bound variable might have no occurrences whatsoever. *Sentences* are exactly formulas with no free variables.

▶ **Proposition 5.** *The truth value of a sentence does not depend on the choice of assignment.*

Proof idea. Reduce to the following special case. Let x be a bound variable of Φ and assume $\nu_2 = \nu_1[a/x]$ for some $a \in \mathbb{Z}$. Then Φ holds under ν_1 if and only if it holds under ν_2. (In fact, Φ may be assumed to be an arbitrary formula, not necessarily a sentence.) ◀

▶ Remark. In logic, the *theory of a structure* is the set of all sentences true in this structure. Presburger arithmetic is, formally, the (first-order) theory of the structure $(\mathbb{Z}, +, \leqslant)$. Consider the sentence $\Phi\colon\ \forall x\, \exists y\ (x = 2y)$, which is false. According to this definition, Φ is therefore not a sentence of Presburger arithmetic. It is, however, a sentence (written) in the syntax of Presburger arithmetic.

In the present paper, whenever we refer to sentences of Presburger arithmetic (linear integer arithmetic), as well as of other logics, the meaning is purely syntactic. We will not use the concept of theory as such, but we have decided to mention the distinction so that the reader does not get confused when consulting literature.

Taking a formula $\varphi(x_1, \dots, x_n)$, that is, one where all free variables are among $x_1, \dots, x_n$, it is occasionally convenient to write $\varphi(a_1, \dots, a_n)$ for the truth value of φ on any assignment ν such that $\nu(x_i) = a_i$ for all i. For a sentence φ, it should be clear from the context whether, when writing φ, we are referring to the syntactic object or to its truth value.

The set $\{\boldsymbol{a} \in \mathbb{Z}^n : \varphi(\boldsymbol{a}) \text{ is true}\}$ is the set *defined* by formula φ; sets for which a suitable formula exists are *definable* sets. Here and elsewhere, we use boldface letters to denote elements of $\mathbb{Z}^n$. We do this for tuples of variables too, writing $\boldsymbol{x} = (x_1, \dots, x_n)$. Two formulas $\varphi(\boldsymbol{x})$ and $\psi(\boldsymbol{x})$ are *equivalent* if they define the same set; we write $\varphi(\boldsymbol{x}) \iff \psi(\boldsymbol{x})$.

3 Three views on linear integer arithmetic

Sets $S \subseteq \mathbb{Z}^d$ definable in linear integer arithmetic (also: *Presburger-definable* sets) can also be represented with the help of geometry, or with the help of strings. The following three representations are available.

Semi-linear sets. A set can be specified by referring to its geometric features. In an analogy, a triangle in $\mathbb{R}^2$ can be specified by referring to its vertices. In the case of linear integer arithmetic, we also need to describe periodic patterns to specify a set.

Finite automata. To represent a subset of natural numbers, we can use a finite automaton over $\{0, 1\}$ that recognises (accepts) the set of binary expansions of these numbers. This idea extends to negative integers, as well as to tuples of integers.

Logical formulas. To represent a set, we can use a formula that is true exactly for the elements of the set. We already saw this representation in Section 1, following the syntax given in Section 2.

Thus, Presburger-definable sets can be viewed in three different ways: from the perspective of geometry, automata theory, and symbolic computation, respectively. Each of the three representations can be employed to solve the decision problem for Presburger arithmetic. The three resulting *methods*, generalised appropriately, can also be used for other logics.

In this section, we describe these three views on linear integer arithmetic in more detail. Actual *algorithms* are not given in this short introduction, only the ideas behind them.

3.1 A view from geometry: semi-linear sets

Semi-linear sets are a generalisation to $\mathbb{Z}^d$ of ultimately periodic sets of natural numbers. Let $S \subseteq \mathbb{N}$.[1] The set S is called:

- *periodic* if there is $p > 0$ such that, for all $x \in \mathbb{N}$, we have $x \in S$ if and only if $x + p \in S$;
- *ultimately periodic* if there are N and $p > 0$ such that, for all $x \geqslant N$, we have $x \in S$ if and only if $x + p \in S$.

We call the numbers p and N the *period* and the *offset*, respectively. A multiple of a period is also a period, and any number exceeding an offset is also an offset.

▶ **Example 6.** In Fig. 2, the top set is periodic with period 3, and the other two sets are not periodic. The top two sets are ultimately periodic, also with period 3. ⌟

[1] Whether $0 \in \mathbb{N}$ or not is a matter of convention. In logic, it is more common to include zero in the set of natural numbers. We follow this convention.

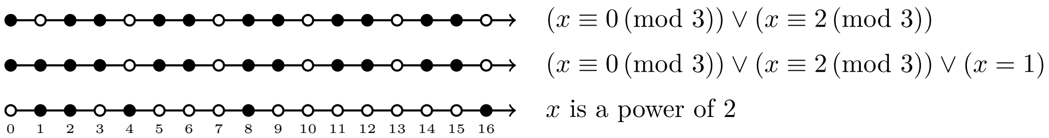

$(x \equiv 0 \,(\mathrm{mod}\ 3)) \vee (x \equiv 2 \,(\mathrm{mod}\ 3))$
$(x \equiv 0 \,(\mathrm{mod}\ 3)) \vee (x \equiv 2 \,(\mathrm{mod}\ 3)) \vee (x = 1)$
x is a power of 2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Figure 2 Three sets of natural numbers as sets of points on the number line with coordinate $x \in \mathbb{N}$. Black and white dots show numbers included in and excluded from the set, respectively. The top two sets are definable in linear integer arithmetic and the bottom set is not.

For the following Proposition, our definition of an arithmetic progression is a set

$$\{a, a+m, a+2m, \ldots\} = \{x \in \mathbb{Z} : \exists t \in \mathbb{Z}\ (x = a + tm) \wedge (t \geqslant 0)\}, \qquad a, m \in \mathbb{N}. \tag{5}$$

A special case is a singleton, if $m = 0$. From Eq. (5), or by the formula $(x \equiv a \,(\mathrm{mod}\ m)) \wedge (x \geqslant a)$, every arithmetic progression is definable in linear integer arithmetic.

▶ **Proposition 7.** *A set $S \subseteq \mathbb{N}$ is ultimately periodic if and only if it is a union of finitely many arithmetic progressions.*

Ultimately periodic sets are closed under the following operations: complement (because p and N stay unchanged); intersection (the least common multiple of the two periods is a new period, and the maximum of the two offsets is a new offset); and union (e.g., by De Morgan's law).

There is more than one way to generalise ultimately periodic sets to higher dimensions. For example, should the two sets shown in Fig. 1 be considered ultimately periodic or not? For us, the most useful analogue will be the following definition due to Parikh [114], which is probably the most inclusive.

We first define an analogue of arithmetic progressions. A set $S \subseteq \mathbb{Z}^d$ is called *linear* if there is $\boldsymbol{b} \in \mathbb{Z}^d$ and a finite set $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\} \subseteq \mathbb{Z}^d$ such that

$$S = L(\boldsymbol{b}, P) \overset{\text{def}}{=} \left\{ \boldsymbol{b} + \sum_{i=1}^{k} \lambda_i \boldsymbol{p}_i : \lambda_1, \ldots, \lambda_k \in \mathbb{N} \right\}. \tag{6}$$

The term "linear" is traditional; the integer programming community uses the term "integer cone" (if $\boldsymbol{b}$ is $\boldsymbol{0} \overset{\text{def}}{=} (0, \ldots, 0)$) instead (see, e.g., [48, 80]). Notice that k may be different from d. If $P = \varnothing$, then $L(\boldsymbol{b}, P) = \{\boldsymbol{b}\}$.

A set is *semi-linear* if it is a union of finitely many linear sets. Points $\boldsymbol{b}$ are referred to as *base points*, *offsets*, or *constants*. Vectors $\boldsymbol{p} \in P$ are *period vectors*, or *periods*. Base points and periods can be collectively referred to as *generators*, and the representation $S = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$ as *generator representation* of S. Generator representation is not unique, unless S is finite.

▶ **Example 8.** Linear set $L(\boldsymbol{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ with $\boldsymbol{p}_1 = (2, 1)$ and $\boldsymbol{p}_2 = (1, 2)$ is shown in Fig. 3 (left). Both sets in Fig. 1 are semi-linear. Indeed, $T = \bigcup_{\boldsymbol{t} \in T} L(\{\boldsymbol{t}\}, \varnothing)$ and $U = L(\{-\boldsymbol{e}_1\}, \{-\boldsymbol{e}_1, \boldsymbol{e}_2, -\boldsymbol{e}_2\}) \cup L(\{-\boldsymbol{e}_2\}, \{-\boldsymbol{e}_2, \boldsymbol{e}_1, -\boldsymbol{e}_1\})$, where $\boldsymbol{e}_1 = (1, 0)$ and $\boldsymbol{e}_2 = (0, 1)$. ⌟

The following theorem shows that nice closure properties of ultimately periodic sets in $\mathbb{N}$ extend to semi-linear sets in $\mathbb{Z}^d$. As it turns out, in dimension 1, ultimately periodic sets are exactly subsets of $\mathbb{N}$ definable in linear integer arithmetic. In higher dimension, definable sets are exactly semi-linear sets.

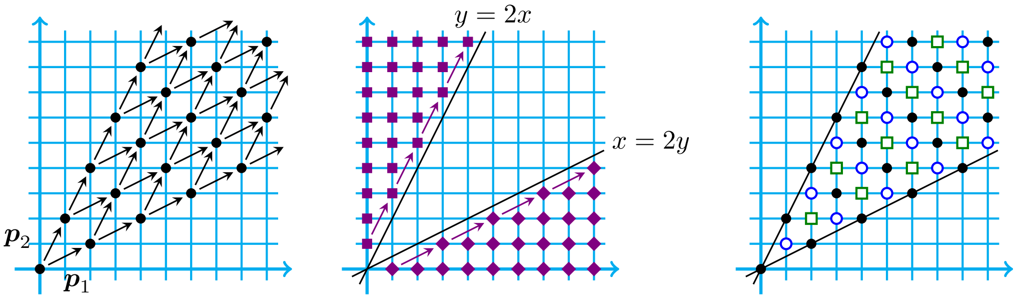


Figure 3 Five sets forming a partition of $\mathbb{N}^2$, for Example 11. Left: linear set $L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ with $\boldsymbol{p}_1 = (2, 1)$ and $\boldsymbol{p}_2 = (1, 2)$. Middle: linear sets V_1 (in diamond shape) and V_2 (in square shape). Period vectors not shown are unit vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, respectively. Right: linear sets W_1 (in empty blue circles) and W_2 (in empty green squares), which are shifts of $L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ by $(1, 1)$ and $(2, 2)$, respectively.

▶ **Theorem 9** (Ginsburg and Spanier [57, 58]). *For a set $S \subseteq \mathbb{Z}^d$, the following are equivalent:*

(a) *$S = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$ for some finite set I, offsets $\boldsymbol{b}_i \in \mathbb{Z}^d$ and finite sets of periods $P_i \subseteq \mathbb{Z}^d$;*

(b) *$S = \{ (a_1, \ldots, a_d) \in \mathbb{Z}^d : \varphi(a_1, \ldots, a_d)$ is true$\}$ for some formula φ of linear integer arithmetic.*

Moreover, this equivalence is effective: there exist algorithms for conversion between the generator representation in (a) *and the formula in* (b)*.*

Proof idea.

(a) ⇒ (b). For given $\boldsymbol{b}$ and $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\}$, the condition $\boldsymbol{x} \in L(\boldsymbol{b}, P)$ can be expressed using a formula with k existentially quantified variables for $\lambda_1, \ldots, \lambda_k$ in Eq. (6). The quantifier-free part of the formula is a conjunction of d equalities, one per coordinate, and d inequalities $\lambda_i \geqslant 0$. A union of $|I|$ sets corresponds to a disjunction of k formulas.

(b) ⇒ (a). The proof is by induction on the structure of the formula φ, following the definition of the syntax from Section 2.

- For an atomic formula (inequality, equality, or congruence) we construct the generator representation directly. (It suffices to prove this for a non-strict inequality, because other atoms can be avoided. Indeed: $t_1 < t_2$ if and only if $\neg(t_1 \geqslant t_2)$; $t_1 = t_2$ if and only if $(t_1 \leqslant t_2) \wedge (t_2 \leqslant t_1)$; and $t_1 \equiv t_2 \pmod{m}$ if and only if $\exists x\ (t_1 - t_2 = mx)$.)
- For formulas of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\neg\varphi$, we prove the closure of the family of semi-linear sets under Boolean operations.
- For an existential quantifier, let $\boldsymbol{y} = (y_1, \ldots, y_d)$ be the vector of free variables. Observe that the set $\{\boldsymbol{b} \in \mathbb{Z}^d$: formula $\exists x\ \varphi(x, \boldsymbol{y})$ holds on $\boldsymbol{b}\}$ can be obtained from the set $\{(a, \boldsymbol{b}) \in \mathbb{Z} \times \mathbb{Z}^d$: formula $\varphi(x, \boldsymbol{y})$ holds on $(a, \boldsymbol{b})\}$ by orthogonal projection: namely by removing the x-component (coordinate) from each element. In the generator representation, we simply cross out this component in all generators. Universal quantifiers can be handled by observing that a formula $\forall x\ \varphi$ is equivalent to $\neg\exists x\ \neg\varphi$.

In truth, this plan requires a tweak: we need to handle *systems* of inequalities as a primitive. (We sketch the construction in Section 5.1.) This is because proofs of the closure of semi-linear sets under intersection use semi-linearity of sets of solutions to such systems. ◀

The algorithm (b) ⇒ (a) of Theorem 9, run on sentences, is a decision procedure for linear integer arithmetic.

▶ **Example 10.** Let us follow the sketch above for the co-primality formula $C_{a,b}$ from Example 1, Section 1. To make the argument concrete, fix $a = 4$ and $b = 6$.

The first step is to find a generator representation of the set of solutions in x_1, x_2, s to $s = 4x_1 - 6x_2$. A suitable one is $L(\mathbf{0}, \{-\boldsymbol{v}_1, \boldsymbol{v}_1, -\boldsymbol{v}_2, \boldsymbol{v}_2\})$ with $\boldsymbol{v}_1 = (1, 0, 4)$ and $\boldsymbol{v}_2 = (0, 1, -6)$, where the coordinates are written in the order x_1, x_2, s. Indeed, it is clear that all vectors of this linear set are solutions; conversely, for a solution (x_1, x_2, s) we have $x_1 \cdot \boldsymbol{v}_1 + x_2 \cdot \boldsymbol{v}_2 = (x_1, x_2, 4x_1 - 6x_2) = (x_1, x_2, s)$, and thus $(x_1, x_2, s) \in L(\mathbf{0}, \{-\boldsymbol{v}_1, \boldsymbol{v}_1, -\boldsymbol{v}_2, \boldsymbol{v}_2\})$.

In the second step, we handle the existential quantifiers that bind x_1 and x_2. We cross out the corresponding coordinates, which leads to the set $S_1 \overset{\text{def}}{=} L(0, \{-4, 4, -6, 6\}) = \{4z_1 + 6z_2 : z_1, z_2 \in \mathbb{Z}\}$. To handle the universal quantifier, we rely on the "equivalence" $\forall = \neg\exists\neg$: indeed, the formula $C_{4,6}$ is true if and only if the set $\mathbb{Z} \setminus S_1$ is empty. To complement S_1, observe that $S_1 \cap \mathbb{N}$ is periodic with period 4 as well as with period 6. It is therefore periodic with period $\gcd(4, 6) = 2$. The same is true for $S_1 \cap \{-n : n \in \mathbb{N}\}$. Since $0 \in S_1$ and $\pm 1 \notin S_1$, we have $S_1 = L(0, \{-2, 2\})$ and $\mathbb{Z} \setminus S_1 = L(1, \{-2, 2\}) \neq \varnothing$. Thus, $C_{4,6}$ is false. ⌟

The construction of a semi-linear set for an inequality $a_0 + a_1x_1 + \ldots + a_dx_d \leqslant 0$, where all $a_i \in \mathbb{Z}$, generalises what we saw for equality $s = 4x_1 - 6x_2$ in Example 10.

The following example illustrates the complementation of a semi-linear set.

▶ **Example 11.** Consider $L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ with $\boldsymbol{p}_1 = (2, 1)$ and $\boldsymbol{p}_2 = (1, 2)$, in Fig. 3. Its complement can be decomposed as $\mathbb{Z}^2 \setminus L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\}) = U \cup V_1 \cup V_2 \cup W_1 \cup W_2$, where:

$$\begin{aligned} U &= \{(x, y) \in \mathbb{Z}^2 : (x < 0) \vee (y < 0)\} \qquad \text{(Example 8)}, \\ V_1 &= \{(x, y) \in \mathbb{Z}^2 : (y \geqslant 0) \wedge (2y < x)\} = L(\boldsymbol{e}_1, \{\boldsymbol{e}_1, \boldsymbol{p}_1\}), \\ V_2 &= \{(x, y) \in \mathbb{Z}^2 : (x \geqslant 0) \wedge (2x < y)\} = L(\boldsymbol{e}_2, \{\boldsymbol{e}_2, \boldsymbol{p}_2\}), \\ W_1 &= L((1, 1), \{\boldsymbol{p}_1, \boldsymbol{p}_2\}), \qquad \text{and} \qquad W_2 = L((2, 2), \{\boldsymbol{p}_1, \boldsymbol{p}_2\}). \end{aligned}$$

Set U is shown in Fig. 1 (right), and sets V_1, V_2, W_1, W_2 in Fig. 3. ⌟

Let us introduce the Minkowski sum notation for sets: $A + B \overset{\text{def}}{=} \{a + b : a \in A, b \in B\}$. We write $a + B$ instead of $\{a\} + B$. In Example 11, sets $W_i = (i, i) + L(\mathbf{0}, P)$ for $i \in \{1, 2\}$ are shifts of the set $L(\mathbf{0}, P)$, where $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$. Moreover, these three sets form a partition of the set of integer points in the sector $\{(x, y) \in \mathbb{R}^2 : (x \leqslant 2y) \wedge (y \leqslant 2x)\}$; see Fig. 3 (right). The sector itself is definable in linear *real* arithmetic. We see on this example a useful rule of thumb: reasoning about integer points combines reasoning about linear *real* arithmetic (intuitively: geometric constraints in $\mathbb{R}^d$) and reasoning about integer *lattices* (intuitively: divisibility properties, or periodic patterns; see, e.g., [101, Section 2.2]).

Further reading. Over the years, multiple algorithms have appeared for operations on semi-linear sets. Early papers [57, 58] and a monograph [56, Chapter 5] paved the way and are still helpful for developing intuition. Huynh's paper [78] is geometric and optimises the size of description. More recent constructions [33, 35] optimise and analyse the dependence of the size on multiple parameters; these papers provide further references.

If the dimension d is fixed, the decision problem for *existential* Presburger arithmetic (where all quantifiers are $\exists$ and appear at the beginning of the formula) can be solved in polynomial time [128]. In fact, all satisfying assignments (if finitely many) can be efficiently enumerated if the formula, moreover, contains no occurrences of $\vee$ and $\neg$ and no congruence constraints. These are consequences of fundamental results of Lenstra [95] and Barvinok [14, 13]; see also, e.g., monographs [108, 39]. Nguyen and Pak [106] look at not

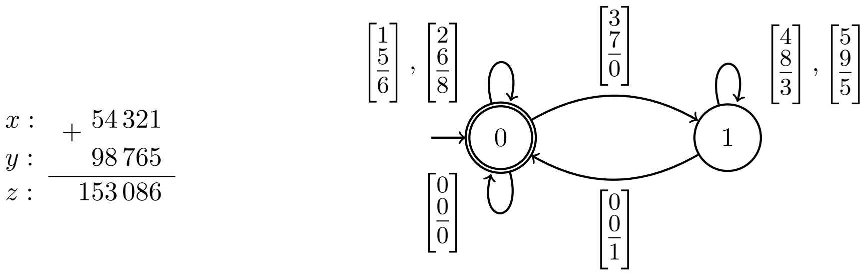


Figure 4 Left: long addition base 10. Right: finite automaton that checks $x + y = z$, reading triplets of digits right to left. Only transitions traversed on the example on the left, as well as the transition for the triplet of leading zeros, are shown.

necessarily orthogonal projections of semi-linear sets (cf. proof idea for Theorem 9) and find a way to compute generating functions for these projections, à la Barvinok. In a certain technical sense, the idea extends to formulas with quantifiers in which d bounds the total number of variables (quantified or not).

In recent years, extensions of semi-linear sets with applications to verification of Petri nets have been considered in the literature [97, 63].

3.2 A view from automata theory: k-automatic sets

For simplicity, throughout this section arithmetic is over $\mathbb{N}$ instead of $\mathbb{Z}$. Representation of sets of numbers using finite automata is a far-reaching development of the following observation. Consider natural numbers divisible by 3. It is well-known that these are precisely the numbers whose sum of decimal digits is divisible by 3. A finite automaton can read the decimal expansion of $n \in \mathbb{N}$ digit by digit (the input alphabet is $\{0, 1, \dots, 9\}$), maintaining the remainder modulo 3 of all digits read so far. Then $3 \mid n$ if and only if the final remainder is 0. In fact, there is nothing special about 3: to capture divisibility by, say, 28, the automaton maintains the remainder modulo 28 of the number read so far.

To move from properties of individual numbers to properties of pairs and triplets of numbers, etc., we use automata over larger input alphabets.

▶ **Example 12.** We describe an automaton that checks addition $x + y = z$ for $x, y, z \in \mathbb{N}$. The idea is to mimic the elementary method of long addition (Fig. 4, left). The automaton will read triplets of decimal digits one at a time, so the input alphabet is $\{0, 1, \dots, 9\}^3$, with 1000 letters. Intuitively, we could say the input tape has three tracks, one for each of x, y, and z. Numbers can be padded with leading zeros (on the left), so that all tracks have the same length. A deterministic finite automaton (DFA) can read the tape from right to left, keeping the carry in its control state; see Fig. 4, right. The state with carry 0 is accepting.

Our automata will usually be incomplete: for example, there are no transitions on input letter $(0, 0, 5) \in \{0, 1, \dots, 9\}^3$, no matter the current carry. To make the DFA complete, we can send all such transitions to a rejecting sink state. ⌟

Let us move from base 10 to base 2. Perhaps counter-intuitively, it will be more convenient for us to use automata that read input from left to right instead, i.e., most significant bit first. (In Example 12, we simply reverse all transitions in the automaton.)

For $d \geqslant 1$, a set $S \subseteq \mathbb{N}^d$ is 2-*automatic* (or: 2-*recognizable*) if there is a deterministic finite automaton (DFA) that accepts the language

$$\{(w_1, \ldots, w_d) \in \left(\{0,1\}^d\right)^* : \text{for some } (n_1, \ldots, n_d) \in S, \text{ each } w_i \text{ is a binary expansion of } n_i\}. \quad (7)$$

Notice that each $n \in \mathbb{N}$ has infinitely many binary expansions: e.g., 110, 0110, 00110, etc. for $6 \in \mathbb{N}$. Replacing the base 2 with a larger integer $k \geqslant 3$, one gets k-automatic sets.

The definition refers to DFA, but any equivalent formalism for regular languages would do just as well, e.g., nondeterministic finite automata (NFA) or regular expressions.

Automata that represent finite sets (as in Fig. 1, left) are not unlike binary decision diagrams (BDDs), but can accept strings of varying length.

▶ **Theorem 13** (Büchi–Bruyère [30, 29], corollary)**.**

1. *Every set $S \subseteq \mathbb{N}^d$ definable in linear integer arithmetic is 2-automatic. The containment is effective: there is an algorithm that, given a formula φ defining S, constructs an automaton accepting the language from Eq.* (7)*.*
2. *There exists a 2-automatic set $S \subseteq \mathbb{N}$ that is not definable in linear integer arithmetic.*

Proof idea. The first part is proved by induction on the structure of the formula. We follow the definition of the syntax from Section 2:

- We need to find an automaton (DFA) for every set defined by an atomic formula (inequality, equality, or congruence). As in the proof of Theorem 9, we can focus on inequalities with no loss of generality. In fact, we reduce the reasoning to the simplest possible case: equalities of the form $x + y = z$ and $2x = z$. For inequalities, introduce *slack variables*: for instance, assuming that all variables range over $\mathbb{N}$,

  $$y \leqslant 2 \qquad \iff \qquad \exists y' \; (y + y' = 2).$$

  Equalities are further rewritten. For example:

  $$s + 2u = 3v \quad \iff \quad \exists y_1 \, \exists y_2 \, \exists y_3 \; (s + y_1 = y_3) \wedge (y_1 = 2u) \wedge (y_2 = 2v) \wedge (y_3 = v + y_2).$$

  We can now use Example 12 with two amendments: input is in base 2 instead of base 10, with most significant bit read first (instead of last).
- Sets defined by formulas of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\neg\varphi$ are 2-automatic by the inductive hypothesis and by the (effective) closure of the family of regular languages under Boolean operations.

  For the complementation to work correctly, it is important that leading zeros not affect acceptance by the given automaton. Otherwise if, say, for $d = 1$ the string 110 is accepted but 0110 is not, then the correspondence between sets of numbers and languages breaks: the complement of the language will still contain a binary expansion of 6.
- We describe the idea how an existential quantifier can be handled. (For universal quantifiers, observe that $\forall x \; \varphi$ is equivalent to $\neg\exists x \; \neg\varphi$.) Let $\boldsymbol{y} = (y_1, \ldots, y_d)$ be the vector of free variables. Intuitively, if binary expansions of $\{(a, \boldsymbol{b}) \in \mathbb{N} \times \mathbb{N}^d : \text{formula } \varphi(x, \boldsymbol{y}) \text{ holds on } (a, \boldsymbol{b})\}$ can be recognised by a DFA over the alphabet $\{0,1\}^{d+1}$, then removing the x-track from all letters leads to the set $\{\boldsymbol{b} \in \mathbb{N}^d : \text{formula } \exists x \; \varphi(x, \boldsymbol{y}) \text{ holds on } \boldsymbol{b}\}$.

  We need to be careful, however: in some accepted strings the erased track may be strictly longer than all other tracks; that is, a has more binary digits than each component of $\boldsymbol{b}$. (In the formula $\exists z \; (x + y = z)$, this happens for the variable z: see Fig. 4 (left).) Thus, after removing x-components from all letters on the transitions of the original DFA, we make a state q accepting if and only if that DFA had an accepting run from q on which all letters have 0 on the d "surviving" tracks.

For the second part of the theorem, observe that the set $P_2 = \{n \in \mathbb{N} \colon n = 2^k \text{ for some } k \in \mathbb{N}\}$ is 2-automatic: binary expansions of powers of 2 are exactly strings matched by regular expression 0^*10^*. Since P_2 is not ultimately periodic, it is not definable in linear integer arithmetic (see Section 3.1). ◀

The first part of Theorem 13, applied to sentences, gives a decision procedure for Presburger arithmetic.

The sketch above offers relatively little intuition as to how automata for various arithmetic constraints really work. In the following example, we show an alternative, direct construction.

▶ **Example 14.** Consider the co-primality formula $C_{a,b}$ from Example 1 (Section 1). Fix $a = 3$ and $b = 2$. Unlike in Section 1, we let all variables range over $\mathbb{N}$ not $\mathbb{Z}$.

For uniformity of notation, let us rename s to x_0. Take the atomic formula $x_0 = 3x_1 - 2x_2$. The idea is similar to the divisibility examples at the beginning of this section. As an automaton reads triplets of bits left to right, it keeps in memory the current *error* (rather than remainder modulo 3 or 28): the integer $e = x_0 - 3x_1 + 2x_2$. (For the moment, we can think of an infinite-state automaton, to be made finite-state shortly.) If $e = 0$, the current state is accepting. Initially $e = 0$, because our equation $x_0 = 3x_1 - 2x_2$ is homogeneous. How does this error change over time?

Suppose the next bit triplet is $(\alpha_0, \alpha_1, \alpha_2) \in \{0,1\}^3$. If x_0, x_1, x_2 are the binary numbers read so far, then after reading the new bits these numbers become

$$x_0' = 2x_0 + \alpha_0, \qquad\qquad x_1' = 2x_1 + \alpha_1, \qquad\qquad x_2' = 2x_2 + \alpha_2.$$

Thus, the new error is

$$e' = x_0' - 3x_1' + 2x_2' = 2(x_0 - 3x_1 + 2x_2) + (\alpha_0 - 3\alpha_1 + 2\alpha_2) = 2e + (\alpha_0 - 3\alpha_1 + 2\alpha_2).$$

Denote $u \overset{\text{def}}{=} \alpha_0 - 3\alpha_1 + 2\alpha_2$ and observe that $u \in [-3, 3]$. Thus, if $|e| \geqslant 3$, then $|e'| \geqslant 3$ as well and so state 0 is no longer reachable. Thus, just five states for $e \in \{-2, -1, 0, 1, 2\}$ suffice, and all other values can be glued into a rejecting sink.

The automaton for the quantifier-free formula $x_0 = 3x_1 - 2x_2$ has a lot of transitions. The table in Fig. 5 (left) shows the update u for each $(\alpha_0, \alpha_1, \alpha_2) \in \{0,1\}^3$. Observe that the destination of transitions depends only on u and not on $\alpha_0, \alpha_1, \alpha_2$. In principle, each bit triplet can be read from each control state, but if the new error $e' = 2e + u$ is outside $[-2, 2]$, then the transition goes into the rejecting sink. Based on this automaton (which we do not depict), we construct an automaton (NFA) for the formula $\varphi(x_0) \colon \exists x_1 \exists x_2\ (x_0 = 3x_1 - 2x_2)$: for each transition, the label $(\alpha_0, \alpha_1, \alpha_2)$ is replaced by just α_0; see Fig. 5 (right).

We omit the acceptance analysis. It turns out that all states except -2 are accepting, and the language recognised by the NFA is $\{0,1\}^*$. Thus, formula $C_{3,2}$ from Example 1 is true even if all variables are interpreted over $\mathbb{N}$. ⌟

Further reading. In connection with the second part of Theorem 13, it is meaningful to ask whether a given set $S \subseteq \mathbb{N}^d$ represented by an automaton is definable in Presburger arithmetic. As it turns out, this definability can be determined in polynomial [96] and, in dimension $d = 1$, even almost linear time [100, 21]. The first of these algorithms was implemented in TaPAS [98], a software framework for the theory of mixed real-integer linear arithmetic [143, 19]. Paper [96] also contains a collection of references for contemporaneous software for linear integer arithmetic. In a different direction, sets represented by automata can be characterised by another logic, so-called Büchi arithmetic (see, e.g., [29, 18]).

α_0	α_1	α_2	u
0	0	0	0
0	0	1	2
0	1	0	-3
0	1	1	-1
1	0	0	1
1	0	1	3
1	1	0	-2
1	1	1	0

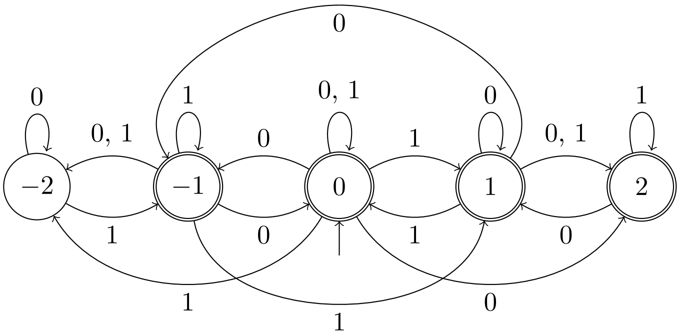


Figure 5 Left: aggregate effect u for each bit triplet. Right: nondeterministic finite automaton for the formula $\varphi(x_0)\colon \exists x_1 \exists x_2\ (x_0 = 3x_1 - 2x_2)$, with variables interpreted over $\mathbb{N}$.

Let us mention a tool MONA [76, 86], which implements an automata-based decision procedure for a more powerful logic: the weak monadic second-order logic with one successor (WS1S). Informally speaking, this logic can "encode" Presburger arithmetic.

A more detailed exploration of the subject of this section (and, more generally, the view of automata as data structures) can be found in Esparza and Blondin's textbook [50, Chapter 9]. Another source is a survey by Boigelot and Wolper [22]. Survey [29] reviews Büchi's idea (including a bugfix) and subsequent developments up to the 1990s. Several chapters in the book [117] discuss topics such as interface of automata theory with number theory (see also Rigo's earlier survey [125]); automatic sequences; and automatic structures. The latter are structures (in logic) in which, informally speaking, relations can be represented by automata (generalising Example 12); see, e.g., Grädel's tutorial [60]. A recent monograph by Shallit [134] explores the use of logic to study automatic sequences (a closely related concept), with applications in combinatorics on words. For a further sample of cutting-edge developments, the reader is referred to papers [66, 139, 70, 43].

3.3 A view from symbolic computation: quantifier elimination

In symbolic computation, we rewrite a given formula iteratively, using a set of predefined rules. The resulting formulas get progressively simpler structurally, usually at the cost of increase in size. Often the objective is to remove quantifiers, so the approach is known as *quantifier elimination*. We first give an example outside Presburger arithmetic.

▶ **Example 15.** Given $p, q \in \mathbb{R}$, a quadratic equation $x^2 + px + q = 0$ has a real solution if and only if its discriminant is nonnegative: $p^2 - 4q \geqslant 0$. Thus, the formulas $\exists x\ [x^2 + px + q = 0]$ and $p^2 - 4q \geqslant 0$ (of a suitable logic over $\mathbb{R}$) are equivalent, that is, we have eliminated the quantifier $\exists x$. (Both formulas have free variables p and q.) ⌟

Importantly, we would like the resulting formula to stay within the syntax of the same logic that we start from. For Presburger arithmetic, this is possible. (As we saw in Section 1 on the example of formula $E(x)\colon \exists y\ (x = 2y)$, congruence constraints cannot be dispensed with.) Geometrically, we saw in Section 3.1 that elimination of an existential quantifier corresponds to orthogonal projection. This is very easy to handle for semi-linear sets in generator representation; for logical formulas, a little more work is required.

A formula is *quantifier-free* if symbols $\exists$ and $\forall$ do not occur in it.

▶ **Theorem 16** (Presburger [121]). *There exists an algorithm that, given a quantifier-free formula φ, outputs a quantifier-free formula φ' equivalent to $\exists x\ \varphi$.*

Theorem 16 gives a decision procedure for linear integer arithmetic, in fact historically the first one. English translation of and commentary on Presburger's 1929 paper are available [137, 122].

Instead of giving the proof in full generality, we will consider several examples. We roughly follow an algorithm given by Cooper [37], later reproduced (under the heading "The present algorithm") in Cooper [38].

▶ **Example 17.** Consider the formula $\exists x\, [(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)]$, which defines the projection of the set T from Fig. 1 (page 2) on the y axis. Only the first two inequalities in the formula mention x. Putting them into a chained inequality $2y \leqslant x \leqslant 3y$, we see that a suitable value for x exists (in $\mathbb{Z}$) if and only if $2y \leqslant 3y$, that is, $y \geqslant 0$. Thus, the entire formula is equivalent to $(y \geqslant 0) \wedge (y \leqslant 2)$. This is exactly the projection $\{0, 1, 2\}$ of the set T on the y axis. ⌟

If the formula has several inequalities bounding x from below (say, $s_1 \leqslant x$, ..., $s_k \leqslant x$, where $s_1, \ldots, s_k$ are some terms not involving x) and several inequalities bounding it from above (say, $x \leqslant t_1$, ..., $x \leqslant t_m$, with $t_1, \ldots, t_m$ not involving x), then a suitable value for x exists (in $\mathbb{Z}$) if and only if $s_i \leqslant t_j$ for all pairs i, j. In words, the greatest lower bound does not exceed the least upper bound.

▶ **Example 18.** Now consider the formula $\exists y\, [(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)]$, which defines the projection of the same set T from Fig. 1 (page 2) on the x axis. To use the same principle as in Example 17, we multiply both sides of each inequality by a positive integer so as to make all the coefficients at y identical. In this case, they will be $6 = \operatorname{lcm}(3, 2, 1)$. For all $x, y \in \mathbb{Z}$, we have $x \leqslant 3y$ if and only if $2x \leqslant 6y$; the other two inequalities are handled similarly. We obtain one lower bound and two upper bounds on $6y$; thus, the formula is true if and only if $2x \leqslant 6y \leqslant \min\{3x, 12\}$, or equivalently $(2x \leqslant 6y \leqslant 3x) \wedge (2x \leqslant 6y \leqslant 12)$.

It is, however, not true that a suitable integer value for y exists if and only if $2x \leqslant \min\{3x, 12\}$. Indeed, $x = 1$ gives a counterexample. Rather, such a value exists if and only if there is an integer divisible by 6 between $2x$ and $\min\{3x, 12\}$. To express this statement without existential quantification ("there is an integer"), we consider several cases depending on the remainder of $2x$ modulo 6. For example, if $2x$ has remainder 2 modulo 6, then instead of inequality $2x \leqslant \min\{3x, 12\}$ the condition to use is $2x + 4 \leqslant \min\{3x, 12\}$, "rounding up" the lower bound to a multiple of 6. (This is because $y = 2x + 4$ is always a suitable choice of y if $2x + 4 \leqslant \min\{3x, 12\}$ and $2x + 4 \equiv 0 \pmod 6$.) The formula is thus equivalent to

$$\bigvee_{r=0}^{5} [(2x + r \equiv 0 \pmod 6) \wedge (2x + r \leqslant \min\{3x, 12\})]$$

or, slightly less succinctly,

$$\bigvee_{r=0}^{5} [(2x + r \equiv 0 \pmod 6) \wedge (2x + r \leqslant 3x) \wedge (2x + r \leqslant 12)]\,.$$

This eliminates the existential quantifier. Observing that only even r are possible and collecting like terms, we can simplify the result further into

$$\begin{aligned} &[(x \equiv 0 \pmod 3)) \wedge (0 \leqslant x) \wedge (2x \leqslant 12)] \vee \\ [(x + 1 &\equiv 0 \pmod 3)) \wedge (2 \leqslant x) \wedge (2x \leqslant 10)] \vee \\ [(x + 2 &\equiv 0 \pmod 3)) \wedge (4 \leqslant x) \wedge (2x \leqslant 8)]. \end{aligned}$$

In the three cases, we get $x \in \{0, 3, 6\}$, $x \in \{2, 5\}$, and $x \in \{4\}$, so the formula defines the set $\mathbb{Z} \cap [0, 6] \setminus \{1\}$. This is the projection of the set T in Fig. 1 (left) on the x axis. ⌟

In Example 18, it is clear that the "hole" $x = 1$ in the projection of T arises because of divisibility constraints, which are necessary because we are eliminating an integer variable. In comparison, for the theory of linear *real* arithmetic, the conjunction $\max s_i \leqslant \min t_j$ already does the job. In fact, over $\mathbb{R}$ this method shows that the orthogonal projection of a convex polyhedron (that is, of the set of real solutions to a system of linear inequalities) is also a convex polyhedron. The method is known as the *Fourier–Motzkin quantifier elimination* for linear real arithmetic (see, e.g., [112, Chapter 1] and [92, Chapter 1]), a useful instrument not only in the development of theory but also in modern practical tools [105].

As in Section 3.1, we see that reasoning about integers requires a combination of an argument for reals and an approach for handling divisibility constraints.

Proof idea for Theorem 16. Using De Morgan's laws and equivalences such as

$$\neg(t_1 \leqslant t_2) \iff (t_2 < t_1), \qquad (t_1 < t_2) \iff (t_1 + 1 \leqslant t_2),$$
$$\neg(t_1 \equiv t_2 \pmod{m}) \iff \bigvee_{r=1}^{m-1} (t_1 + r \equiv t_2 \pmod{m}),$$

bring the given formula φ to disjunctive normal form (DNF), or rather (more precisely) an OR of ANDs of non-strict linear inequalities and congruence constraints (no negations). As $[\exists x\ (A \vee B)] \iff (\exists x\ A) \vee (\exists x\ B)$, the problem reduces to handling one such AND. For this special case, the idea is shown in Examples 17 and 18, and here is a more structured sketch:

1. Multiply both sides of each constraint so that all the coefficients at x become identical, say to $m \in \mathbb{Z}$. (A natural choice for m is the least common multiple of all coefficients at x.)
2. Replace all occurrences of mx by x', where x' is a new (*fresh*) variable. Conjoin (AND) the congruence $x' \equiv 0 \,(\mathrm{mod}\ m)$ to the formula. We are now handling $\exists x'$ instead of $\exists x$. (In Example 18, the auxiliary variable would be y' standing for $6y$.)
3. The formula is now an AND of constraints not involving x', several lower bounds on x', several upper bounds on x', and several congruence constraints on x'. We split into cases (OR) according to which of the lower bounds $s_1, \ldots, s_k$ is the greatest and according to the remainder (of this greatest lower bound) with respect to the least common multiple of moduli in congruence constraints. Within each case, x' can be eliminated. ◀

▶ Remark 19. A different take on quantifier elimination for Presburger arithmetic follows Cooper's algorithm ("the new algorithm") in [38], not requiring conversion to DNF. After all coefficients at variable x are made identical, the inequalities involving x split the set $\mathbb{Z}$ into a finite number of intervals. A suitable value for x exists if and only if one of these intervals contains an integer that satisfies (or perhaps fails) a certain subset of the inequalities and has some divisibility properties (as prescribed by the Boolean structure of the formula).

Suppose the least common multiple of all divisors in the divisibility constraints is $M > 0$. ($M = 1$ if there are no such constraints.) Let $[\alpha, \beta]$ be one of the intervals, where α and β are terms in which x does not appear. Then a suitable value for x (importantly, one with the right divisibility properties) exists in $[\alpha, \beta]$ if and only if one of $\alpha, \alpha + 1, \ldots, \alpha + (M - 1)$ is suitable and does not exceed β; this is because shifting x by M leaves the truth value of all divisibility constraints unchanged. The case $\alpha = -\infty$ is similar and simpler. ⌟

▶ **Observation 20.** Let a formula φ' be output by Cooper's quantifier elimination procedure on input φ, which has eliminated variable x. Constraints of φ' arise from equating pairs of expressions for (or rather bounds on) x. If φ contains inequalities $s \leqslant ax$ and $bx \leqslant t$, where $a, b > 0$, then their conjunction entails $bs \leqslant at$, or equivalently $at - bs \geqslant 0$. A rescaling by nonzero $\lambda \in \mathbb{Q}$ may be necessary if $\mathrm{lcm}(a, b) \neq ab$, or if other coefficients are taken into account when computing the least common multiple. If, as in Example 18, divisibility constraints come into the picture, then a constant $\alpha \in \mathbb{Z}$ may be added or subtracted.

In fact, *all* inequalities in φ' can be shown to arise this way. Suppose that an atomic formula $\tau \leqslant 0$ mentions two or more variables and appears in φ'. Then there are numbers $\lambda \in \mathbb{Q}$, $\alpha \in \mathbb{Z}$ such that τ is $\lambda \cdot (a_1\tau_2 - a_2\tau_1) + \alpha$ for some terms $a_1x + \tau_1$ and $a_2x + \tau_2$ which appear in the original formula φ. (For brevity, we do not make precise the notion of appearance of a term in a formula.)

Further reading. Cooper's paper [38] is a lucid introduction to quantifier elimination, and his decision procedures have been analysed in more detail and extended (see, e.g., Oppen [111] as well as Section 4.1 of the present paper). A formalisation in Isabelle/HOL is available [109].

The "big" disjunctions in the formulas resulting from quantifier elimination in Example 18 can instead be replaced by a bounded version of the existential quantifier ($\exists r \in [0, 5]$). Such quantifiers are not part of the syntax (Section 2) but can be added. This leads to so-called *uniform* (generalisation of) Presburger arithmetic and the idea of *weak* quantifier elimination [142], developed further [90] and implemented in Redlog. Redlog [42] is part of the general-purpose computer algebra system REDUCE and offers quantifier elimination methods for multiple arithmetic theories, Presburger arithmetic being one of them.

Chapter 4 of Kreisel and Krivine's book [88] introduces quantifier elimination for several arithmetic theories, such as Presburger arithmetic (although this name is not mentioned). The reader should beware that this book uses symbols $\bigwedge$ and $\bigvee$ in place of now ubiquitous $\forall$ and $\exists$. Unlike our presentation, the book also discusses not only model-theoretic aspects of quantifier elimination but also proof-theoretic aspects (which axioms can justify the equivalence of formulas).

Multiple further versions and extensions of quantifier elimination, for a variety of logical theories, have been proposed in the literature and implemented in software. Most famously, Tarski's quantifier elimination procedure for the first-order theory of $\mathbb{R}$ with addition, multiplication, and ordering led in the 1970s to Collins's cylindrical algebraic decomposition, a fundamental concept in computer algebra. In the first-order theory of the reals, atomic formulas are inequalities between multivariate polynomials; see, e.g., [44] and [103, Section 5].

Many extensions of Presburger arithmetic are known to have quantifier elimination (see, e.g., [132, 99, 133, 130]). For Presburger arithmetic itself, Weispfenning [141] analyses the size of formulas output by (his) quantifier elimination procedure. Bounds that he obtains in terms of the number of quantifier blocks (alternation) and the number of variables in each block are a versatile instrument for other problems. An early implementation of such a procedure is the Omega test [123]. In the SMT solver cvc5, quantifier elimination for linear arithmetic uses so-called counterexample-guided instantiation [124, 9].

4 Alternation of quantifiers and computational complexity

The three views shown in Section 3 give rise to three (kinds of) decision procedures for Presburger arithmetic. In the current section, we look further into the computational complexity aspects.

If we analyse the decision procedures sketched above directly, we get overly pessimistic estimates of the running time and memory requirements. These estimates can be improved substantially (Section 4.1). Nevertheless, the decision problem turns out inherently hard. Clever formulas in linear integer arithmetic can express rather intricate sets, and it is possible to prove worst-case lower bounds on the complexity of the problem itself (Section 4.2).

Table 1 Effect of logical connectives and quantifiers on the size of representation (informally). For all three views, $\forall$ can be replaced by $\neg\exists\neg$.

View	Geometry	Automata theory	Symbolic computation (quantifier elimination)
Representation	Semi-linear sets	Automata	Logical formulas
$\neg$	expensive	NFA $\rightsquigarrow$ DFA	CNF $\rightsquigarrow$ DNF*
$\vee$	trivial	trivial	trivial
$\wedge$	OK	easy	easy*
$\exists$	trivial	easy	expensive

* For some but not all quantifier elimination procedures. Trivial for other procedures.

As we will see, the high computational complexity is in some sense linked with the alternation of quantifiers of type $\exists$ and $\forall$ in the logical formulas. The present section focuses on formulas with alternation, and Section 5 on formulas without it.

4.1 Handling quantifiers in decision procedures

As seen from Table 1, in each of the three views on Presburger arithmetic some of the logical connectives and quantifiers may require a big increase in the size required to represent the object – a semi-linear set, an automaton, or a logical formula. In slightly more detail:

For semi-linear sets, complementation is the most difficult operation. Intersection requires work but the increase in size is smaller in comparison. (In dimension 1, the new period can be chosen as the least common multiple of old periods; in higher dimension a suitable generalisation is required.)

For automata, the existential quantifier requires an update to the set of accepting states (which is easy) and, most importantly, makes the automaton nondeterministic (NFA not DFA). Complementation of the NFA is then expensive, e.g., requiring the subset construction.

For logical formulas, some quantifier elimination algorithms require conversion to DNF. For them, negation is difficult (conversion from CNF to DNF) and conjunction requires the application of the distributive law. Other algorithms can handle arbitrary Boolean structure. In either case, the existential quantifier presents the main challenge.

In summary, for each method some operation requires an exponential growth in size, also referred to as an exponential blow-up, in the worst case.

If n is the size of the input sentence, then stacking n exponentials leads to a worst-case upper bound on the running time of decision procedures that has the form $2^{\cdot^{\cdot^{\cdot^{2}}}}$, where the height of the tower grows as n. Differences between functions such as 2^n, 2^{cn}, 2^{n^2}, etc., can be ignored for the purpose of this crude estimate. For all three views, a sequence of n alternating quantifiers $\exists x_1 \forall x_2 \exists x_3 \dots$ presents a challenge.

Importantly, better decision procedures (and sometimes better analyses) are available.

A function $f\colon \mathbb{N} \to \mathbb{N}$ is called *elementary* if $f(n) \leqslant 2^{\cdot^{\cdot^{\cdot^{2^n}}}}$, where the tower has some fixed height $k \in \mathbb{N}$. A yes–no problem belongs to the complexity class ELEMENTARY if it has an algorithm with elementary running time. For example, all problems in complexity classes P, NP, PSPACE, EXP, 2-EXP, etc., also belong to this huge class. According to English Wikipedia, "The name was coined by László Kalmár, in the context of recursive functions and undecidability; most problems in it are far from elementary." For computationally hard problems in logic, achieving polynomial running time or even polynomial space (memory usage) is often not possible, and the dichotomy between elementary and non-elementary bounds can indicate problems for which practical implementations can hope to handle some medium-size inputs in reasonable time.

▶ **Theorem 21** (Oppen [111]). *There is an algorithm that solves the decision problem for Presburger arithmetic in triply exponential time.*

As we discuss below, in fact all three approaches (based on semi-linear sets, on automata, and on the elimination of quantifiers) provide elementary decision procedures.

Proof idea. Cooper's quantifier elimination algorithm [38] sketched in Section 3.3 can be shown to require at most $2^{2^{2^{pn}}}$ basic computational steps on sentences of size n, where $p > 0$ is some universal constant. To improve upon the crude non-elementary upper bound, we need to measure the complexity of formulas using several parameters, rather than just one (size of the formula). The dependence of these parameters on one another may still seem to lead to a tower of exponentials, but the idea is to find a parameter that enjoys a tighter (elementary) estimate. We can think of it as a *controlling parameter*, because the other parameters can then be bounded once we know this key one is under control.

Our controlling parameter will be the number of distinct coefficients of variables in the formula. For a formula φ, let us denote this quantity by $z(\varphi)$. We only sketch the core of the argument, leaving many details to the reader. Consider the elimination of a single existential quantifier. Let φ be a formula given to Cooper's algorithm, and let φ' be the output formula equivalent to $\exists x\ \varphi$. Let us review Observation 20 from Section 3.3. Each inequality in φ' arises from two inequalities of φ by cross-multiplication. Given terms $ax + by + t'$ and $cx + dy + t''$ that appear in inequalities of φ, the result is

$$(ad - bc)\, y + (at'' - ct') \sim 0, \tag{8}$$

where $\sim$ is one of the signs $\leqslant$, $\geqslant$; possibly rescaled by some nonzero $\lambda \in \mathbb{Q}$ and with an additive shift $\alpha \in \mathbb{Z}$. We ignore the scaling factor $\lambda \in \mathbb{Q}$ in this proof. So, for a variable y, every coefficient at y in the new formula φ' is a combination of at most 4 coefficients in φ, two of them at y and two at the variable x that is being eliminated. Thus, $z(\varphi') \leqslant z(\varphi)^4$.

Now denote by φ_k the formula obtained from φ after eliminating k quantifiers; $\varphi_0 = \varphi$. In the worst case, $z(\varphi_k)$ grows with k as follows:

$$z(\varphi) = z \quad \rightsquigarrow \quad z^4 \quad \rightsquigarrow \quad z^{4^2} \quad \rightsquigarrow \quad z^{4^3} \quad \rightsquigarrow \quad \ldots \quad \rightsquigarrow \quad z^{4^k}.$$

Formally, one can prove by induction an upper bound $z(\varphi_k) \leqslant M_k \overset{\text{def}}{=} z(\varphi)^{4^k}$. Since k is bounded from above by the size of the input formula φ, the number of distinct coefficients throughout the entire procedure is at most doubly exponential in the size of φ. With this elementary bound in hand, we can bound other parameters, such as the magnitude of coefficients, of moduli in congruence constraints, of constant terms, etc. For instance, the number of *linear terms* of the form $a_1x_1 + \ldots + a_mx_m$ which may arise in the formula is bounded by $(M_k)^m$, where the number of variables, m, is again at most the size of φ. This is also an elementary bound. (We note that the total number of inequalities might be much higher than the number of linear terms, because of additive constants.) Bounds can then be combined into an elementary bound on the size of formulas. ◀

There are a number of factors that we have glossed over. For example, in the sketch above we have considered two constraints in isolation, whereas the least common multiple of all coefficients at x may be much bigger than just ac. However, the principle remains sound.

Starting with Oppen's theorem, elementary decision procedures have been given for all three views on linear integer arithmetic. Oppen's result was extended to automata [85, 45, 46] and recently to semi-linear sets [35]. Weispfenning's quantifier elimination procedure [141] also runs in triply exponential time, as best procedures for all three views do.

We briefly comment on the geometric view. It is convenient to extend the $L(\boldsymbol{b}, P)$ notation (Eq. (6) in Section 3.1) to $L(B, P) \overset{\text{def}}{=} \bigcup_{\boldsymbol{b} \in B} L(\boldsymbol{b}, P)$. Sets of this form have been studied by Ginsburg and Spanier [57, 56], and we refer to them as *hybrid linear* sets. As we will see in Section 5.1, these are sets of integer solutions to systems of linear inequalities. Semi-linear sets are exactly unions

$$S = \bigcup_{i \in I} L(B_i, P_i), \qquad |I|, |B_i|, |P_i| < \infty. \tag{9}$$

As it turns out, in generator representation the cardinality $|I|$ can be chosen as the controlling parameter for elementary bounds. By a discrete version of Carathéodory's theorem (see, e.g., [48] and [33, Proposition 5]), the reasoning can be reduced to the case where each set P_i contains linearly independent vectors only. The reader may now see the link with the number of *linear terms* in the logical formulas during quantifier elimination. A useful ingredient in the constructions is the observation, for every fixed $d \geqslant 1$, that n hyperplanes in $\mathbb{R}^d$ split it into at most $O(n^d)$ regions [101, Chapter 6].

4.2 Using quantifiers to write succinct formulas

We typically use arbitrary integers in Presburger formulas. One can ask if the succinctness of the logic changes if only small integers (say, between -2 and 2) are allowed. The answer to this question is negative: for instance, the formula $y = 2^n \cdot x$ is equivalent to

$$\exists y_0 \, \exists y_1 \, \ldots \exists y_n \; (y_0 = x) \wedge (y_1 = 2y_0) \wedge (y_2 = 2y_1) \wedge \ldots \wedge (y_n = 2y_{n-1}) \wedge (y = y_n).$$

Thus, numbers up to $2^{\mathrm{poly}(n)}$ can be expressed by formulas of size $\mathrm{poly}(n)$ with coefficients from $[-2, 2]$. Can we express even larger numbers using small formulas?

► **Example 22** (Fischer and Rabin [53]). There is a sequence of formulas $F_n(x, y)$ of size $O(n)$ such that $F_n(x, y)$ holds if and only if $x = 2^{2^n} \cdot y$. According to Fischer and Rabin [53], "[this] device is a special case of a more general theorem due to M. Fischer and A. Meyer. It was rediscovered independently by several people including V. Strassen."

We can take $F_0(x, y)\colon\; x = 2y$, since $2 = 2^{2^0}$. The sequence is then constructed inductively. The idea is to have $F_{n+1}(x, y)$ equivalent to $\exists z_n \, [F_n(x, z_n) \wedge F_n(z_n, y)]$. This direct definition, however, would unravel into a very big formula, of size $O(2^n)$ for index n. Instead, the following construction has only one instance of F_n and thus unravels into a formula of size $O(n)$:

$$F_{n+1}(x, y)\colon\; \exists z_n \, \forall u_n \, \forall v_n \, \big[\big(\big((u_n = x) \wedge (v_n = z_n)\big) \vee \big((u_n = z_n) \wedge (v_n = y)\big)\big) \to F_n(u_n, v_n)\big].$$

The *bit size* of the formula is actually $O(n \log n)$, with the $\log n$ factor due to the need to refer to at least n distinct variables. Interestingly, F_n works just as well over $\mathbb{R}$. ⌟

Combining Example 22 with the Chinese remainder theorem (generalising Example 2), Fischer and Rabin also give a sequence of formulas that define triply exponential rather than doubly exponential numbers; see Kozen's textbook [87, Lecture 24] for details.

Formulas from Example 22 and more sophisticated ones are then used to "define" and manipulate long bit strings. This led Fischer and Rabin [53] to lower bounds on the computational complexity of the decision problem for linear integer (as well as real) arithmetic.

▶ **Theorem 23** (Fischer and Rabin [53]). *The decision problem for Presburger arithmetic requires at least doubly exponential time in the worst case, even for nondeterministic algorithms.*

For linear real arithmetic, the lower bound is a single exponential. A precise characterisation of the computational complexity of the problem was subsequently given by Berman [17]: the decision problem for Presburger arithmetic is complete for $\mathrm{STA}(*, 2^{2^{O(n)}}, n)$. The acronym STA stands for “space, time, alternation”; this class corresponds to Turing machines that run in time $2^{2^{cn}}$ for some $c > 0$ on inputs of length n, using up to $n-1$ alternations between nondeterministic and universal modes (with $n = 0$ corresponding to deterministic algorithms). Kozen [87, Lectures 23–24] offers a modern exposition of results of this kind.

Theorem 23 and Berman’s strengthening of it suggest that triply exponential algorithms (Section 4.1) are optimal in the worst case: known simulation results of 2-NEXP and $\mathrm{STA}(*, 2^{2^{O(n)}}, n)$ machines by deterministic algorithms run in triply exponential time.

In logical formulas arising from applications, *alternation depth* is often low, i.e., formulas may have the form $\Phi(\boldsymbol{u})\colon \forall \boldsymbol{x}^1 \exists \boldsymbol{x}^2 \ldots \mathrm{Q}\boldsymbol{x}^k\ \varphi(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k, \boldsymbol{u})$ with small k. We can think of k as being fixed. Here φ is quantifier-free, Q is either $\exists$ or $\forall$ depending on the parity of k, and all variables in each block $\boldsymbol{x}^i$ are bound by the same kind of quantifier (existential or universal). As an example, $\forall^*\exists^*$ formulas have alternation depth $k = 2$. Naturally, sentences may also start with the existential block.

▶ **Example 24** (Grädel [59], see also Haase [64]). There exists a constant $c > 0$, a sequence of integers $(r_n)_{n\geqslant 1}$, and a sequence of $\forall\exists^*$-formulas $G_n(z)$ of size $O(n)$ such that (i) $G_n(z)$ holds if and only if r_n divides z and (ii) $r_n \geqslant 2^{2^{cn}}$ for all n.

The construction consists of three steps. In the first step, we construct a sequence of formulas $M_n(x, y, z)$ for *bounded multiplication:* under the assumption $0 \leqslant x < 2^n$, the formula $M_n(x, y, z)$ will be true if and only if $x \cdot y = z$. For other values of x, the truth value of M_n is unconstrained. Here is the formula:

$$\begin{aligned} M_n(x,y,z)\colon \quad & \exists x_1 \ldots \exists x_n\ \exists z_1 \ldots \exists z_n \\ & \qquad (x = x_1 \cdot 2^{n-1} + x_2 \cdot 2^{n-2} + \ldots + x_n) \wedge \bigwedge_{i=1}^{n} [(0 \leqslant x_i) \wedge (x_i \leqslant 1)] \wedge \\ & \qquad (z = z_1 \cdot 2^{n-1} + z_2 \cdot 2^{n-2} + \ldots + z_n) \wedge \\ & \bigwedge_{i=1}^{n} \big[((x_i = 0) \to (z_i = 0)) \wedge \\ & \qquad ((x_i = 1) \to (z_i = y)) \big]. \end{aligned}$$

This is an *existential* formula of size $O(n)$.

The second step constructs the formula $D_n(x, z)\colon \exists y\ M_n(x, y, z)$. Under the assumption $0 \leqslant x < 2^n$, this formula asserts that z is a multiple of x. In the third step we arrive at $G_n(z)\colon \forall x\ [((0 < x) \wedge (x < 2^n)) \to D_n(x, z))]$. Now $r_n = \mathrm{lcm}(1, 2, \ldots, 2^n - 1)$. ⌟

Example 24 leads to Grädel’s lower bound in the following theorem:

▶ **Theorem 25** (Grädel [59] and Haase [64]). *The decision problem for $\forall^*\exists^*$-Presburger arithmetic is complete for* coNEXP.

Class coNEXP contains complements of problems that can be decided in nondeterministic time $O(2^{n^c})$, for some $c > 0$. The upper bound is due to Haase, who also proved that, for each $k \geqslant 2$, the fragment of Presburger arithmetic with alternation depth k is complete for the $(k-1)$st level of the so-called weak EXP hierarchy [64].

More recently, Nguyen and Pak have studied the computational complexity of further restricted fragments. By encoding problems related to continued fractions, they were able to show, for instance, that the decision problem for sentences of the form $\exists z \, \forall y_1 \, \forall y_2 \, \exists x_1 \exists x_2 \colon \varphi(x_1, x_2, y_1, y_2, z)$ is NP-hard even for φ with 10 inequalities [107]. In this problem, the sentence is fixed entirely except for the numbers in atomic formulas.

5 Three views on integer programming

The feasibility problem of *integer (linear) programming* is the following problem:

Input: A system $A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}$ of m linear inequalities in n variables, where $A \in \mathbb{Z}^{m \times n}$ and $\boldsymbol{c} \in \mathbb{Z}^m$.
Output: Does the system have a solution in $\mathbb{Z}^n$?

Here the relation $\leqslant$ between vectors is component-wise: for $\boldsymbol{a} = (a_1, \ldots, a_m)$ and $\boldsymbol{b} = (b_1, \ldots, b_m)$ we write $\boldsymbol{a} \leqslant \boldsymbol{b}$ if and only if $a_i \leqslant b_i$ for each i between 1 and m.

In the present paper, "integer programming" will refer to this problem (that is, we will not consider optimisation versions thereof). It is a special case of the decision problem for Presburger arithmetic, and even for *existential* Presburger arithmetic, where all quantifiers are existential and appear at the beginning of the formula.

▶ **Theorem 26** (Borosh and Treybig [24] and Papadimitriou [113]). *Integer programming is* NP*-complete.*

▶ **Corollary 27** (see, e.g., [111]). *Existential Presburger arithmetic is* NP*-complete.*

In these statements, we omit the words "the decision problem for". NP-hardness of both problems is easy to justify, by an encoding of the subset sum problem (Example 3, conjunctive formula) or Boolean satisfiability (SAT). We now derive Corollary 27 from Theorem 26.

Proof of Corollary 27. Given a sentence in Presburger arithmetic, where all quantifiers are existential and appear at the beginning of the formula, we eliminate all negations (except on congruence constraints) using De Morgan's laws and the equivalence $(t_1 < t_2) \iff (t_1 + 1 \leqslant t_2)$. We can similarly eliminate all atomic formulas except inequalities and the remaining negated congruences. The quantifier-free part of the formula is now a monotone Boolean combination of atoms of two types: non-strict linear inequalities and negated congruences. Such a sentence is true if and only if there exists a subset of these atoms and, for each negated congruence $\neg(t_1 \equiv t_2 \pmod{m})$, a remainder $r \in \{1, \ldots, m-1\}$ such that:

(1) if all the atoms in the subset are true, then the formula is true;
(2) if each negated congruence in the subset is replaced by the equality $t_1 + r = t_2 + mw$, where w is a fresh variable, and then by two inequalities $t_1 + r \leqslant t_2 + mw$ and $t_1 + r \geqslant t_2 + mw$, then the resulting system of inequalities is a positive instance of the integer programming problem.

The first condition is checked directly, and the second is an NP condition by Theorem 26. ◀

Thus, existential Presburger arithmetic is very close to integer programming. Membership of integer programming in NP is not obvious. It is indeed the case that a guessed solution can be verified quickly; but a separate argument is required to show that, whenever some solution exists, a *small* solution exists too, where "small" means "of polynomial bit size relative to the bit size of the system".

At first sight, appeal to the three views on Presburger arithmetic does not seem to resolve the problem: the size of representation in the decision procedures might well increase to doubly exponential. Nevertheless, all three views *can* actually be used to prove the NP upper bound of Theorem 26. This is the subject of three subsections below.

For several perspectives on integer programming, one can recommend books [81, 129, 145].

5.1 A view from geometry: discrete convex polyhedra

In this section we take the geometric view, following the ideas of von zur Gathen and Sieveking [55]. The resulting proof of Theorem 26 makes use of the principle we have already seen: first consider constraints over $\mathbb{R}$ (or $\mathbb{Q}$) instead of $\mathbb{Z}$.

We recall basic definitions from polyhedral geometry and convex analysis. For an introduction, the reader is referred to Lauritzen's undergraduate textbook [92] or lecture notes [91], and to Paffenholz's lecture notes [112]. More advanced material can be found in De Loera, Hemmecke, and Köppe [39], Rockafellar [127], and Schrijver [129].

A linear combination $\lambda_1 \boldsymbol{v}_1 + \ldots + \lambda_n \boldsymbol{v}_n$ of $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$ is called:

- *conic* (or: positive) if all $\lambda_i \geqslant 0$,
- *affine* if $\lambda_1 + \ldots + \lambda_n = 1$, and
- *convex* if it is conic and affine.

Given a set $S \subseteq \mathbb{R}^d$, the set of all its conic (affine, convex) combinations is the *cone* generated by S, the *affine hull* and the *convex hull* of S, denoted $\operatorname{cone} S$, $\operatorname{aff} S$, and $\operatorname{conv} S$, respectively. We will use the Minkowski sum and product notation: $A + B = \{a + b : a \in A, b \in B\}$ and $A \cdot B = \{a \cdot b : a \in A, b \in B\}$.

The following famous theorem gives two equivalent characterisations of *convex polyhedra:*

▶ **Theorem 28** (Minkowski–Weyl, 1896, 1935). *For $P \subseteq \mathbb{R}^d$, the following are equivalent:*
(H) $P = \{\boldsymbol{x} : A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}\}$ *for some matrix A and vector $\boldsymbol{c}$;*
(V) $P = \operatorname{conv} F + \operatorname{cone} G$ *for some finite sets F and G.*

In words of Rockafellar [127, Section 19]: "This classical result is an outstanding example of a fact which is completely obvious to geometric intuition, but which wields important algebraic content and is not trivial to prove." The (V) ⇒ (H) direction can in fact be obtained by a direct application of Fourier–Motzkin quantifier elimination (Section 3.3) over $\mathbb{R}$.

Representations (H) and (V) are referred to as *H-representation* and *V-representation* of P. In the former, P is the intersection of a finite collection of **h**alf-spaces. In the latter, elements of F can be thought of as **v**ertices and elements of G as directions in which P "is infinite" (directions of recession).

The *size* of each representation is the number of bits required to write it (that is, $A, \boldsymbol{c}$ or F, G) down. Let us assume henceforth that we deal with rational numbers only. An effective version of the Minkowski–Weyl theorem over $\mathbb{Q}$ states that the two representations can be translated from one to another; see, e.g., [129, Chapter 10] and [112, Chapter 5]:

▶ **Lemma 29.** *In Theorem 28:*

1. *A and $\boldsymbol{c}$ can be made rational if and only if F and G can be made rational.*
2. *Let P be nonempty and suppose either representation is given, of bit size s. Then the other can be computed, with the bit size of each number at most* $\operatorname{poly}(s)$*.*

We do not give a proof of Lemma 29. One of the possible paths is through the theory of structure of convex polyhedra: intuitively, in $\mathbb{R}^d$ each vertex (element of F) is determined as the intersection of d hyperplanes (facets), and each "infinite direction" (element of G) by the intersection of $d - 1$ hyperplanes. By Cramer's rule from linear algebra, solutions to systems of linear equations are formed by ratios of appropriately formed determinants. This yields a polynomial bound on the bit size of numbers. However, in degenerate situations (e.g., linear dependencies or underdetermined systems) some more work is needed.

In both translations (H) $\Rightarrow$ (V) and (V) $\Rightarrow$ (H), the total blow-up in size can be exponential while the size of individual numbers stays polynomial. An example is the "box" $[0,1]^d = \{(x_1, \ldots, x_d) \in \mathbb{R}^d : 0 \leqslant x_i \leqslant 1 \text{ for all } i\}$, a convex polyhedron which has a short H-representation but 2^d vertices. A realisation of the translations is the double description method [92, Chapter 5], see also [112, Chapters 2–5] and [54, Chapter 5].

Let us move from $\mathbb{Q}$ to $\mathbb{Z}$. Recall the definition of hybrid linear sets $L(B, P)$ from the end of Section 4.1.

▶ **Theorem 30** (von zur Gathen and Sieveking [55]). *For $S \subseteq \mathbb{Z}^d$, the following are equivalent:*
(a) *S is a projection of $\{\boldsymbol{x} \in \mathbb{Z}^k : A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}\}$ for some $A \in \mathbb{Z}^{m \times k}$, $\boldsymbol{c} \in \mathbb{Z}^m$, and $k \geqslant d$;*
(b) *$S = L(C, Q)$ for some finite sets $C \subseteq \mathbb{Z}^d$ and $Q \subseteq \mathbb{Z}^d$.*
Moreover, suppose $P \neq \varnothing$ and either representation is given of bit size s. Then the other can be computed, with the bit size of each number at most $\mathrm{poly}(s)$*.*

The projection in Theorem 30 is orthogonal to the principal axes: some $k - d$ coordinates are simply crossed out. This theorem entails the NP upper bound of Theorem 26. It highlights and uses the idea that hybrid linear sets are a discrete analog of convex polyhedra. Indeed, the set

$$\Big\{ \sum \lambda_i \boldsymbol{b}_i + \sum \mu_j \boldsymbol{p}_j : \quad \sum \lambda_i = 1,\ \lambda_i \geqslant 0,\ \mu_j \geqslant 0 \Big\} \tag{10}$$

is the convex polyhedron $\operatorname{conv} B + \operatorname{cone} P$ if $\lambda_i, \mu_j \in \mathbb{R}$; and the hybrid linear set $L(B, P)$ if $\lambda_i, \mu_j \in \mathbb{Z}$.

Proof of Theorem 30. Direction (b) to (a) is straightforward. Auxiliary variables λ_i and μ_j in Eq. (10) correspond to coordinates that are crossed out (projected away).

For (a) to (b), we apply Lemma 29. Taking an arbitrary $\boldsymbol{x} \in \mathbb{Z}^k$ from the set $\{\boldsymbol{x} : A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}\} = \operatorname{conv} F + \operatorname{cone} G$, we can write it, for some $\lambda_i, \mu_j \geqslant 0$ such that $\sum \lambda_i = 1$, as

$$\boldsymbol{x} = \sum \lambda_i \boldsymbol{f}_i + \sum \mu_j \boldsymbol{g}_j = \Big(\sum \lambda_i \boldsymbol{f}_i + \sum (\mu_j - \lfloor \mu_j \rfloor) \boldsymbol{g}_j\Big) + \sum \lfloor \mu_j \rfloor \boldsymbol{g}_j,$$

where $\lfloor a \rfloor$ stands for the greatest integer not exceeding $a \in \mathbb{R}$, and $\boldsymbol{f}_i$, $\boldsymbol{g}_j$ are elements of F and G, respectively. Note that we can assume with no loss of generality that all vectors in G belong to $\mathbb{Z}^k$: indeed, they must be in $\mathbb{Q}^k$ by Lemma 29, and therefore each of them can be multiplied by the least common multiple of the denominators of its components. This shows that each vector $\lfloor \mu_j \rfloor \boldsymbol{g}_j$ is also in $\mathbb{Z}^k$. But then, since $\boldsymbol{x} \in \mathbb{Z}^k$, the difference of these two vectors (which is the vector in the big round brackets) is also integral. Therefore, $\boldsymbol{x}$ belongs to $L(C', Q')$, where Q' is the rescaled version of G and C' is the set of integer points in the Minkowski sum $\operatorname{conv} F + \sum [0, 1) \cdot \{\boldsymbol{g}_j\}$. Since F and G are finite sets, this sum is bounded and so C' is also finite. Finally, sets C and Q are obtained from C' and Q', respectively, by crossing out some of the components. We leave the verification of the "moreover" part of the statement to the reader. ◀

▶ **Example 31.** Consider the triangle example from Eq. (1), page 2, with formula $\varphi(x, y)\colon\ (x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)$. All vertices of the triangle are actually integer points, see Fig. 1 (left). Consider, however, the modified formula $\varphi(x, y) \wedge (x \geqslant 1)$. The origin $(0, 0)$ is no longer a feasible point, and in fact the set of rational solutions is $\operatorname{conv}(\boldsymbol{u}', \boldsymbol{u}'', \boldsymbol{v}, \boldsymbol{w})$, where $\boldsymbol{v} = (4, 2)$ and $\boldsymbol{w} = (6, 2)$ are unchanged and $\boldsymbol{u}' = (1, 1/3)$ and $\boldsymbol{u}'' = (1, 1/2)$. The two new vertices are not integer points, and it is not difficult to imagine all of the "corners" of our polytope (polygon in this example) to be cut off. Feasibility of the integer program would thus hinge on combining a rational point from the convex hull, $\sum \lambda_i \boldsymbol{f}_i$ in the proof of Theorem 30, with a rational point from the cone, $\sum (\mu_j - \lfloor \mu_j \rfloor) \boldsymbol{g}_j$. ⌟

We now give an example showing that the number of generators of the hybrid linear set $L(C, Q)$ in Theorem 30 may be big, namely comparable to the magnitude of integers in the system $A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}$.

▶ **Example 32.** Consider the set $S = \{(x, y, z) \in \mathbb{Z}^3 : (x + y - 50z = 0) \wedge (x \geqslant 0) \wedge (y \geqslant 0)\}$. We have $S = L(\boldsymbol{0}, \{\boldsymbol{v}^0, \boldsymbol{v}^1, \ldots, \boldsymbol{v}^{50}\})$, where $\boldsymbol{v}^i = (i, 50 - i, 1)$. This blow-up in size is unavoidable: if $S = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$, then $|I| + |\bigcup_{i \in I} P_i| \geqslant 52$. Indeed, notice that $\bigcup_{i \in I} P_i \subseteq S \subseteq \mathbb{N}^3$. The first containment holds because S is defined by a conjunction of homogeneous constraints. Thus, $\boldsymbol{v}^m \notin L(\boldsymbol{0}, \bigcup_{i \in I} P_i \setminus \{\boldsymbol{v}^m\})$ for all m. So, whenever $\boldsymbol{v}^m \in L(\boldsymbol{b}_i, P_i)$, we have either $\boldsymbol{v}^m = \boldsymbol{b}_i$, or $\boldsymbol{v}^m \in P_i$ and $\boldsymbol{b}_i = \boldsymbol{0}$. This proves the inequality stated above.

In the absence of inequalities $x \geqslant 0$ and $y \geqslant 0$, we would instead have $S' = \{(x, y, z) \in \mathbb{Z}^3 : x + y - 50z\} = L(\boldsymbol{0}, \{\pm \boldsymbol{u}, \pm \boldsymbol{v}\})$, where $\boldsymbol{u} = (50, 0, 1)$ and $\boldsymbol{v} = (-1, 1, 0)$. Intuitively, in the definition of the set S the two inequalities prevent the vectors $\pm \boldsymbol{v}$ from becoming periods. More generally, a nonempty set $C \subseteq \mathbb{Z}^d$ defined by a system (conjunction) of linear equations with integer coefficients is a coset of a finitely generated *lattice* (an analog of an affine subspace, but over $\mathbb{Z}$ not $\mathbb{R}$). The number of generators of the lattice is $d - r$, where r is the rank of the system, which means that C can be written as a linear set with one base vector and $2(d - r)$ periods. ⌟

The set P of periods of a linear set $L(\boldsymbol{0}, P)$ is also known as the *Hilbert basis*. If the positive cone of the set is pointed, then there is a unique Hilbert basis of minimal size [39, Corollary 2.6.4]. Software tools and libraries that can compute Hilbert bases are available, e.g., Normaliz [28, 27] and polymake [7].

Ideas presented in this section tend to be very useful, and so is Theorem 30. Another presentation is [39, Section 2.6]. Some recent extensions and applications can be found in [94, 34, 40, 36]. Further development of ideas presented in this section have recently led to a new quantifier elimination procedure for Presburger arithmetic [68].

5.2 A view from automata theory: pumping lemma

In Section 3.2 we have described and used representation of integers base $k \geqslant 2$, for so-called k-automatic sets. The author is not aware of an argument proving the NP upper bound of Theorem 26 using such representations. Indeed, extension of this NP upper bound to existential Büchi arithmetic [62, 66] rather required *building upon* Theorem 26 (or, more precisely, Theorem 30). We present a slightly different approach instead, which can also be interpreted as automata-theoretic, but in the unary representation (base 1). That is, $4 \in \mathbb{N}$ is represented as $1 + 1 + 1 + 1$ in this numeration system.

It is a well-known fact that the integer programming problem can equivalently be formulated as follows:

Input: A system $A \cdot \boldsymbol{x} = \boldsymbol{b}$ of linear equations, with A an integer matrix and $\boldsymbol{b}$ an integer vector.

Output: Does the system have a solution over $\mathbb{N}$?

As a reminder, we use $\mathbb{N} = \{0, 1, \ldots\}$ in this paper. Our goal is to bound the norm of smallest solutions from above: this is easy if all entries of A have the same sign, but not necessarily obvious in general.

For $\boldsymbol{v} = (v_1, \ldots, v_d) \in \mathbb{R}^d$, denote by $\|\boldsymbol{v}\|$ its Euclidean length. In fact, any norm would work just as well.

The following classical theorem is sometimes known as the Steinitz lemma.

▶ **Theorem 33** (Steinitz lemma). *Let $d \geqslant 1$ and let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be a sequence of vectors from $\mathbb{R}^d$. Assume $\|\boldsymbol{v}_i\| \leqslant 1$ for all i. If $\sum_{i=1}^{n} \boldsymbol{v}_i = \boldsymbol{0}$, then by reordering the vectors we can obtain another sequence $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ such that $\| \sum_{i=1}^{k} \boldsymbol{u}_i\| \leqslant d$ for all $k = 0, \ldots, n$.*

It is not immediately clear why this is true. Think of each $\boldsymbol{v}_i$ as a vector of travel (movement), and of a sequence of vectors as a travel itinerary. If we start from the origin $\boldsymbol{0}$, then following the entire sequence takes us back to the origin. Reordering the vectors will not change this, but can we reorder so that we never go far from $\boldsymbol{0}$ on the way?

It is rather remarkable that the distance can be kept small, at most d, no matter how big or small n is. Matoušek's wonderful book on applications of linear algebra in combinatorics, geometry, and computer science presents a short proof [102, Miniature 20]. The upper bound of d is due to Grinberg and Sevastyanov [61].

Let us apply the Steinitz lemma to integer programming, following Eisenbrand and Weismantel [49]. For simplicity, consider a homogeneous equation $A \cdot \boldsymbol{x} = \boldsymbol{0}$ first. To show an upper bound on the norm of *smallest* nonzero solutions over $\mathbb{N}$, take *some* solution $\boldsymbol{x} = (p_1, \ldots, p_s)$, where s is the number of variables in the system and all $p_i \in \mathbb{N}$. Let $n = p_1 + \ldots + p_s$ and let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be the sequence of columns of A with the ith column taken p_i times. (This is the "unary" representation of solution $(p_1, \ldots, p_s)$.) As we know that $A \cdot (p_1, \ldots, p_s) = \boldsymbol{0}$, we can apply the Steinitz lemma to scaled vectors $\boldsymbol{p}_i / \max_i \|\boldsymbol{p}_i\|$. In the reordered sequence $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ (without scaling: each $\boldsymbol{u}_i$ is some $\boldsymbol{v}_j$), all partial sums $\sum_{i=1}^{k} \boldsymbol{u}_i$ do not deviate much from $\boldsymbol{0}$; here $k = 0, \ldots, n$. Instead of $\leqslant d$ in Theorem 33, we get $\leqslant H \overset{\text{def}}{=} d \cdot \max_i \|\boldsymbol{p}_i\|$ instead.

The remaining part of the argument is simple. Recall that all $\boldsymbol{u}_i$ are integer vectors, and there are not too many integer points in $\mathbb{R}^m$ with norm at most H; let us say at most $N_m(H)$ such points. (Here m is the number of equations in the system.)

- If $n > N_m(H)$, then some point apart from $\boldsymbol{0}$ is visited twice (or $\boldsymbol{0}$ is visited three times or more). Hence a part of the sequence $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ can be removed without affecting the total sum. In other words, the solution $\boldsymbol{x}$ was not minimal to start with.
- Otherwise $n \leqslant N_m(H)$, and this is already an upper bound on the norm of solution.

We omit the calculations (see, e.g., [49]), but the bounds are sufficient for the NP upper bound of Theorem 26. It remains to remark that nonzero $\boldsymbol{b}$ in $A \cdot \boldsymbol{x} = \boldsymbol{b}$ can be incorporated in the argument too, for example by putting $-\boldsymbol{b}$ into the sequence.

The argument in the bullet list is reminiscent of the pumping lemma in automata theory: in a finite-state automaton, every sufficiently long accepting run must repeat a state and can therefore be shortened.

For further reading and applications, we refer to [8, 110].

5.3 A view from symbolic computation: Gaussian elimination

In 2024, two new approaches to quantifier elimination for existential Presburger arithmetic were proposed. Each can be thought of as a nondeterministic polynomial-time rewriting procedure for existential formulas. Run deterministically, the procedures output quantifier-free formulas of exponential size; polynomial-size formulas can instead be produced in an appropriately extended syntax. One of the procedures, by Chistikov, Mansutti, and Starchak [36], is based on Gaussian elimination and the other, by Haase, Krishna, Madnani, Mishra, and Zetzsche [68], on geometry of convex polyhedra. We outline key ideas behind the former, focusing on integer programming (conjunctive formulas).

Fraction-free Gaussian elimination. The approach develops a remark by Weispfenning [142, Corollary 4.3]. The basis of the algorithm is Gaussian elimination, a standard method for solving systems of linear equations. It is well known that, over rational numbers ($\mathbb{Q}$), it can be performed in polynomial time. An elegant argument showing that each number appearing in the computation is a ratio of two minors (sub-determinants) of the original matrix is given in Schrijver's book [129, Section 3.3]. Thus, the bit size of numbers stays bounded by a polynomial in the bit size of the input.

We need instead a version of Gaussian elimination for integers ($\mathbb{Z}$). The simplest *division-free* algorithm works as follows: given equations $ax + by + \ldots = 0$ and $cx + dy + \ldots = 0$, to eliminate x we "cross-multiply" them by c and a, respectively, and then subtract one from the other (cf. Observation 20). The result is

$$(ad - bc)\, y + \ldots = 0. \tag{11}$$

The bit size of coefficients may double, and repeated cross-multiplication is known to lead (in the worst case) to numbers of doubly exponential magnitude, i.e., of exponential bit size.

As it turns out, there exists a method avoiding this blow-up, which appears in the works of Edmonds [47] and Bareiss [10] and was apparently known as early as 1888 to Clasen [6]. The following question is instructive. The coefficient at y in Eq. (11) is a 2×2 determinant. Gaussian elimination can be seen as computing many such 2×2 determinants, inductively. *How* does the determinant of a big (say $n \times n$) square matrix, which can be computed using Gaussian elimination, arise in this process?

The answer appears in a beautiful paper by Dodgson [41]. Let $A = (a_{ij})$ be a 3×3 matrix. Let us apply the first two steps of Gaussian elimination: after the first step, the coefficient matrix becomes

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{11}a_{22} - a_{12}a_{21} & a_{11}a_{23} - a_{13}a_{21} \\ 0 & a_{11}a_{32} - a_{12}a_{31} & a_{11}a_{33} - a_{13}a_{31} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ 0 & \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} \end{pmatrix}$$

and then, after the second elimination step, the entry in position $(3, 3)$ becomes

$$\begin{vmatrix} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} \end{vmatrix}.$$

By (a special case of) an identity due to Sylvester and Jacobi–Desnanot (see, e.g., [10, 41, 82]), this coefficient is actually equal to the product $a_{11} \cdot \det A$. So, when division-free Gaussian elimination is run on a bigger matrix, after *two* initial steps all coefficients in row 3 and below become *divisible* by a_{11}, as long as all original entries of A are integers. We can divide by a_{11}, and carry on, performing similar divisions after each step. The result is a *fraction-free* Gaussian elimination [47, 10].

Handling inequalities with slack variables and nondeterminism. A second ingredient is required to handle inequalities. We turn each of them into an equality, introducing *slack variables* as, e.g., in our proof sketch for Theorem 13. In the run of Gaussian elimination, each iteration takes an equality from the system and eliminates one actual (non-slack) variable.

If this equality contains a slack variable not handled so far, a small value for this slack variable is guessed. The fact that small values (i.e., of polynomial bit size) suffice requires a proof. These guessed values have the same nature as remainders r in Example 18, or shifts $+0, \ldots, +(M-1)$ in Remark 19.

For further details, we refer the reader to the paper [36]. The algorithm can be extended to arbitrary existential Presburger formulas, in which case (i) the shifts assume both positive and negative values, and (ii) the equation to pick at each iteration is guessed nondeterministically. Unlike [68], the paper [36] does not explicitly describe the formula produced by elimination process but shows how to integrate the algorithm into an NP decision procedure for an extension of existential Presburger arithmetic with nonlinear constraints of the forms $y = 2^x$ and $z = (x \bmod 2^y)$. (This extension was previously shown decidable by automata-theoretic methods [43].) Paper [68] draws a multitude of other consequences from efficient quantifier elimination.

6 Further directions

There are several sources to help new people enter this field. Bradley and Manna's textbook [25] offers a detailed introduction to computational logic, including logical theories of arithmetic, and applications in program verification. Kroening and Strichman's textbook [89] focuses on decision procedures and includes a chapter on quantifier-free linear real and integer arithmetic and another on handling quantified formulas. Kirby's recent textbook on model theory [84] targets an audience of undergraduate and Master's students, assuming little in the way of background knowledge beyond the basics of abstract algebra.

Haase's survey [65] provides a variety of references on Presburger arithmetic. An earlier survey by Bès [18] gives an overview of definability and decidability questions for arithmetic theories more broadly. Michaux and Villemaire's survey of open questions [104], with a focus on Presburger and Büchi arithmetic (see Section 3.2), instigated several subsequent developments. For arithmetic theories that include both addition and multiplication, beyond the literature mentioned at the end of Section 3.3 we refer the reader to Poonen's lecture notes on Hilbert's 10th problem over rings [120] and a recent article by Pasten [115]. An early reference on the computational complexity of logical theories is Ferrante and Rackoff's monograph [52].

We already highlighted (in Section 1) several sources on satisfiability modulo theories (SMT) solving. Among further such sources focusing on logical theories of arithmetic are an overview [3] and an evolving electronic book on SAT and SMT [147] containing many worked examples and aimed at programmers. A new self-contained overview of the arithmetic engine of Z3 (a successful SMT solver and theorem prover) has recently appeared [20].

Applications in verification. An early application of Presburger arithmetic appeared in the work of Parikh [114] and Ginsburg and Spanier [57, 58]. Parikh considered the commutative image, nowadays also known as Parikh image, of formal languages. For example, the commutative image of a word $w \in \{a, b\}^*$ is a 2-dimensional vector $\psi(w) = (m, n)$ such that the letters a and b occur in w exactly m and n times, respectively: $\psi(abbab) = (2, 3)$. The commutative image of a language $L \subseteq \{a, b\}^*$ is $\psi(L) = \{\psi(w) : w \in L\}$. Parikh's theorem shows that the commutative image of every context-free language forms a *semi-linear set*, and in fact for every semi-linear set S there exists a regular language with commutative image S. Ginsburg and Spanier later proved that semi-linear sets are exactly sets definable in Presburger arithmetic [58]; see Section 3.1.

Intuitively, context-free languages can be used to model procedure calls and recursion in software. More generally, linear arithmetic constraints can capture periodic and ultimately periodic behaviours. A well-known result constructs a polynomial-size existential Presburger formula for the Parikh image of a context-free language, see [131, 140] and [75]. This has further applications and extensions [67, 51, 74, 79, 72]. Thanks to the closure properties of semi-linear sets, in the algorithmic foundations of verification "expressible in Presburger arithmetic" often means tractable: complicated behaviours of systems can be approximated and analysed with the help of this logic (see, e.g., [1, 73, 16]).

Extensions and other arithmetic theories. Ongoing research on arithmetic theories extends decidability to more expressive theories and characterises the complexity of decidable ones.

Some extensions of Presburger arithmetic do not, in fact, change the family of definable sets, but increase the succinctness of the logic. For Presburger arithmetic with counting quantifiers [130, 71] and with the star operator [118, 69], complexity questions are open.

As already mentioned, adding arbitrary polynomial constraints to Presburger arithmetic makes the theory undecidable. There is, however, a multitude of decidable theories. Such is the extension of linear integer arithmetic with exponentiation base 2 [133, 32, 119], see also [146, 15, 36]. Effectively, this means a new type of constraints is allowed, namely those of the form $y = 2^x$, where x, y are variables. The choice of integer base $k \geqslant 2$ is immaterial but needs to remain the same throughout the formula.

Alternatively, the logic can permit, in addition to linear integer constraints, assertions of the form "x is a Fibonacci number" as basic building blocks. Such building blocks are referred to as predicates. Many other *sparse* predicates can be added instead of the Fibonacci one, whilst keeping the resulting theory decidable [132, 99]. For the predicate "x is a power of 2", an algorithm with elementary running time has been found [15]. On the theme of adding several "powers" predicates simultaneously, recent references are [77] and [83].

Recall that linear integer arithmetic includes predicates for divisibility by fixed integers. Adding the full divisibility predicate "x divides y" (where x and y are variables) renders the theory undecidable [126, 18], whilst keeping decidable its *existential fragment* (sentences with existential quantifiers only, all of which must appear at the beginning of the formula without negations in between). Whether the decision problem for this fragment belongs to NP is an open question; see, e.g., [94, 138, 116, 40]. In comparison, if instead of divisibility $(x \mid y)$ we add multiplication $(x \cdot y = z)$, then even the existential fragment becomes undecidable, by a celebrated result due to Davis, Putnam, Robinson, and Matiyasevich (negative resolution of Hilbert's 10th problem); see, e.g., [120].

Famous open problems in number theory can be expressed using sentences in linear integer arithmetic extended with a predicate $P(x)$ asserting that x is prime. Indeed, the twin prime conjecture is expressed with the sentence $\forall x \, \exists y \, [(y \geqslant x) \wedge P(y) \wedge P(y+2)]$, and the Goldbach conjecture with the sentence $\forall x \, \exists y \, [(x \leqslant 2) \vee (P(y) \wedge P(x-y))]$. It remains open whether Presburger arithmetic with the primes predicate (P) is decidable; see, e.g., [31, 93].

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Roland Meyer, and Mehdi Seyed Salehi. What's decidable about availability languages? In *FSTTCS*, pages 192–205, 2015. doi:10.4230/LIPICS.FSTTCS.2015.192.

2 Erika Ábrahám, József Kovács, and Anne Remke. SMT: something you must try. In *iFM*, pages 3–18, 2023. doi:10.1007/978-3-031-47705-8_1.

3 Erika Ábrahám and Gereon Kremer. SMT solving for arithmetic theories: Theory and tool support. In *SYNASC*, pages 1–8, 2017. doi:10.1109/SYNASC.2017.00009.

4 Jorge L. Ramírez Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16(1):143–147, 1996. doi:10.1007/BF01300131.

5 Jorge L. Ramírez Alfonsín. *The Diophantine Frobenius problem.* Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. doi:10.1093/acprof:oso/9780198568209.001.0001.

6 Steven C. Althoen and Renate McLaughlin. Gauss-Jordan reduction: A brief history. *Am. Math. Mon.*, 94(2):130–142, 1987.

7 Benjamin Assarf, Ewgenij Gawrilow, Katrin Herr, Michael Joswig, Benjamin Lorenz, Andreas Paffenholz, and Thomas Rehn. Computing convex hulls and counting integer points with polymake. *Math. Program. Comput.*, 9(1):1–38, 2017. doi:10.1007/S12532-016-0104-Z.

8 Imre Bárány. On the power of linear dependencies. In *Building Bridges: Between Mathematics and Computer Science*, volume 19 of *Bolyai Society Mathematical Studies*, pages 31–45. Springer, 2008. doi:10.1007/978-3-540-85221-6_1.

9 Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS*, pages 415–442, 2022. doi:10.1007/978-3-030-99524-9_24.

10 Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comput.*, 22:565–578, 1968. doi:10.1090/S0025-5718-1968-0226829-0.

11 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability — Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021. doi:10.3233/FAIA201017.

12 Clark W. Barrett, Cesare Tinelli, Haniel Barbosa, Aina Niemetz, Mathias Preiner, Andrew Reynolds, and Yoni Zohar. Satisfiability modulo theories: A beginner's tutorial. In *FM*, pages 571–596, 2024. doi:10.1007/978-3-031-71177-0_31.

13 Alexander Barvinok. *Integer points in polyhedra.* EMS Press, 2008. doi:10.4171/052.

14 Alexander I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res.*, 19(4):769–779, 1994. doi:10.1287/MOOR.19.4.769.

15 Michael Benedikt, Dmitry Chistikov, and Alessio Mansutti. The complexity of Presburger arithmetic with power or powers. In *ICALP*, pages 112:1–112:18, 2023. doi:10.4230/LIPICS.ICALP.2023.112.

16 Pascal Bergsträßer, Moses Ganardi, Anthony W. Lin, and Georg Zetzsche. Ramsey quantifiers in linear arithmetics. *Proc. ACM Program. Lang.*, 8(POPL):1–32, 2024. doi:10.1145/3632843.

17 Leonard Berman. The complexity of logical theories. *Theor. Comput. Sci.*, 11:71–77, 1980. doi:10.1016/0304-3975(80)90037-7.

18 Alexis Bès. A survey of arithmetical definability. *A tribute to Maurice Boffa. B. Belg. Math. Soc.-Sim.*, suppl.:1–54, 2001. URL: http://lacl.u-pec.fr/bes/publi/survey.pdf.

19 Alexis Bès and Christian Choffrut. Theories of real addition with and without a predicate for integers. *Log. Methods Comput. Sci.*, 17(2), 2021. doi:10.23638/LMCS-17(2:18)2021.

20 Nikolaj S. Bjørner and Lev Nachmanson. Arithmetic solving in Z3. In *CAV, part I*, pages 26–41, 2024. doi:10.1007/978-3-031-65627-9_2.

21 Bernard Boigelot, Isabelle Mainz, Victor Marsault, and Michel Rigo. An efficient algorithm to decide periodicity of b-recognisable sets using MSDF convention. In *ICALP*, pages 118:1–118:14, 2017. doi:10.4230/LIPICS.ICALP.2017.118.

22 Bernard Boigelot and Pierre Wolper. Representing arithmetic constraints with finite automata: An overview. In *ICLP*, LNCS, pages 1–19, 2002. doi:10.1007/3-540-45619-8_1.

23 George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, 5th edition, 2007. doi:10.1017/CBO9780511804076.

24 Itshak Borosh and Leon Bruce Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Am. Math. Soc.*, 55(2):299–304, 1976. doi:10.1090/S0002-9939-1976-0396605-3.

25 Aaron R. Bradley and Zohar Manna. *The calculus of computation: Decision procedures with applications to verification*. Springer, 2007. doi:10.1007/978-3-540-74113-8.

26 Martin Bromberger. *Decision Procedures for Linear Arithmetic*. PhD thesis, Saarland University, Saarbrücken, Germany, 2019. URL: https://tel.archives-ouvertes.fr/tel-02427371.

27 Winfried Bruns, Bogdan Ichim, and Christof Söger. The power of pyramid decomposition in Normaliz. *J. Symb. Comput.*, 74:513–536, 2016. doi:10.1016/J.JSC.2015.09.003.

28 Winfried Bruns, Bogdan Ichim, Christof Söger, and Ulrich von der Ohe. Normaliz. Algorithms for rational cones and affine monoids. URL: https://www.normaliz.uni-osnabrueck.de/.

29 Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *B. Belg. Math. Soc.-Sim.*, 1(2):191–238, 1994. doi:10.36045/bbms/1103408547.

30 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Math. Log. Q.*, 6(1–6):66–92, 1960. doi:10.1002/malq.19600060105.

31 Patrick Cegielski, Denis Richard, and Maxim Vsemirnov. On the additive theory of prime numbers II. In *CSIT 2005 (Computer Science and Information Technologies, September 19–23, 2005, Yerevan, Armenia)*, pages 39–47, 2005. doi:10.48550/arXiv.math/0609554.

32 Gregory Cherlin and Françoise Point. On extensions of Presburger arithmetic. In *4th Easter Conference on Model Theory*, volume 86 of *Humboldt-Univ. Berlin Seminarberichte*, pages 17–34, 1986. URL: https://webusers.imj-prg.fr/~francoise.point/papiers/cherlin_point86.pdf.

33 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *ICALP*, pages 128:1–128:13, 2016. doi:10.4230/LIPICS.ICALP.2016.128.

34 Dmitry Chistikov and Christoph Haase. On the complexity of quantified integer programming. In *ICALP*, pages 94:1–94:13, 2017. doi:10.4230/LIPICS.ICALP.2017.94.

35 Dmitry Chistikov, Christoph Haase, and Alessio Mansutti. Geometric decision procedures and the VC dimension of linear arithmetic theories. In *LICS*, pages 59:1–59:13, 2022. doi:10.1145/3531130.3533372.

36 Dmitry Chistikov, Alessio Mansutti, and Mikhail R. Starchak. Integer linear-exponential programming in NP by quantifier elimination. In *ICALP*, pages 132:1–132:20, 2024. doi:10.4230/LIPICS.ICALP.2024.132.

37 David C. Cooper. Programs for mechanical program verification. In *Machine Intelligence*, volume 6, pages 43–59. Edinburgh University Press, 1971.

38 David C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99. Edinburgh University Press, 1972.

39 Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and geometric ideas in the theory of discrete optimization*, volume MO14 of *MOS-SIAM Series on Optimization*. SIAM and MOS, 2013. doi:10.1137/1.9781611972443.

40 Rémy Défossez, Christoph Haase, Alessio Mansutti, and Guillermo A. Pérez. Integer programming with GCD constraints. In *SODA*, pages 3605–3658, 2024. doi:10.1137/1.9781611977912.128.

41 Charles L. Dodgson. Condensation of determinants, being a new and brief method for computing their arithmetical values. *Proc. R. Soc. Lond.*, 15:150–155, 1867. doi:10.1098/rspl.1866.0037.

42 Andreas Dolzmann and Thomas Sturm. REDLOG: computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, 1997. doi:10.1145/261320.261324.

43 Andrei Draghici, Christoph Haase, and Florin Manea. Semënov arithmetic, affine VASS, and string constraints. In *STACS*, pages 29:1–29:19, 2024. doi:10.4230/LIPICS.STACS.2024.29.

44 Lou van den Dries. Alfred Tarski's elimination theory for real closed fields. *J. Symb. Log.*, 53(1):7–19, 1988. doi:10.1017/S0022481200028899.

45 Antoine Durand-Gasselin and Peter Habermehl. On the use of non-deterministic automata for Presburger arithmetic. In *CONCUR*, pages 373–387, 2010. doi:10.1007/978-3-642-15375-4_26.

46 Antoine Durand-Gasselin and Peter Habermehl. Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree. In *STACS*, pages 242–253, 2012. doi:10.4230/LIPIcs.STACS.2012.242.

47 Jack Edmonds. Systems of distinct representatives and linear algebra. *J. Res. NBS-B. Math. Sci.*, 71B(4):241–245, 1967. doi:10.6028/jres.071B.033.

48 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

49 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

50 Javier Esparza and Michael Blondin. *Automata theory: An algorithmic approach.* MIT Press, 2023.

51 Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL*, pages 499–510, 2011. doi:10.1145/1926385.1926443.

52 Jeanne Ferrante and Charles W. Rackoff. *The computational complexity of logical theories*, volume 718 of *Lecture Notes in Mathematics.* Springer, 1979. doi:10.1007/BFb0062837.

53 Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 27–41. AMS, 1974.

54 Jean Gallier and Jocelyn Quaintance. Aspects of convex geometry: Polyhedra, linear programming, shellings, Voronoi diagrams, Delaunay triangulations, 2022. URL: https://www.cis.upenn.edu/~jean/gbooks/convexpoly.html.

55 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proc. Am. Math. Soc.*, 72(1):155–158, 1978. doi:10.1090/S0002-9939-1978-0500555-0.

56 Seymour Ginsburg. *The mathematical theory of context-free languages.* McGraw-Hill, 1966.

57 Seymour Ginsburg and Edwin H. Spanier. Bounded ALGOL-like languages. *Trans. Am. Math. Soc.*, 113(2):333–368, 1964. doi:10.1090/S0002-9947-1964-0181500-1.

58 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. doi:10.2140/pjm.1966.16.285.

59 Erich Grädel. Dominoes and the complexity of subclasses of logical theories. *Ann. Pure Appl. Log.*, 43(1):1–30, 1989. doi:10.1016/0168-0072(89)90023-7.

60 Erich Grädel. Automatic structures: Twenty years later. In *LICS*, pages 21–34, 2020. doi:10.1145/3373718.3394734.

61 Victor S. Grinberg and Sergey V. Sevast'yanov. Value of the Steinitz constant. *Funct. Anal. Appl.*, 14(2):125–126, 1980. doi:10.1007/BF01086559.

62 Florent Guépin, Christoph Haase, and James Worrell. On the existential theories of Büchi arithmetic and linear *p*-adic fields. In *LICS*, 2019. doi:10.1109/LICS.2019.8785681.

63 Roland Guttenberg, Mikhail A. Raskin, and Javier Esparza. Geometry of reachability sets of vector addition systems. In *CONCUR*, pages 6:1–6:16, 2023. doi:10.4230/LIPICS.CONCUR.2023.6.

64 Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *CSL-LICS*, pages 47:1–47:10, 2014. doi:10.1145/2603088.2603092.

65 Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. URL: https://www.cs.ox.ac.uk/people/christoph.haase/home/publication/haa-18/haa-18.pdf, doi:10.1145/3242953.3242964.

66 Christoph Haase. Approaching arithmetic theories with finite-state automata. In *LATA*, pages 33–43, 2020. doi:10.1007/978-3-030-40608-0_3.

67 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR*, pages 369–383, 2009. doi:10.1007/978-3-642-04081-8_25.

68 Christoph Haase, Shankara Narayanan Krishna, Khushraj Madnani, Om Swostik Mishra, and Georg Zetzsche. An efficient quantifier elimination procedure for Presburger arithmetic. In *ICALP*, pages 142:1–142:17, 2024. doi:10.4230/LIPICS.ICALP.2024.142.

69 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *LICS*, 2019. doi:10.1109/LICS.2019.8785850.

70 Peter Habermehl, Vojtech Havlena, Michal Hecko, Lukás Holík, and Ondrej Lengál. Algebraic reasoning meets automata in solving linear integer arithmetic. In *CAV, part I*, pages 42–67, 2024. doi:10.1007/978-3-031-65627-9_3.

71 Peter Habermehl and Dietrich Kuske. On Presburger arithmetic extended with non-unary counting quantifiers. *Log. Methods Comput. Sci.*, 19(3), 2023. doi:10.46298/LMCS-19(3:4)2023.

72 Matthew Hague, Artur Jez, and Anthony W. Lin. Parikh's theorem made symbolic. *Proc. ACM Program. Lang.*, 8(POPL):1945–1977, 2024. doi:10.1145/3632907.

73 Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Monadic decomposition in integer linear arithmetic. In *IJCAR*, pages 122–140, 2020. doi:10.1007/978-3-030-51074-9_8.

74 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *CAV*, pages 743–759, 2011. doi:10.1007/978-3-642-22110-1_60.

75 Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, pages 260–276, 2012. Full version: https://anthonywlin.github.io/papers/cav12-long.pdf. doi:10.1007/978-3-642-31424-7_22.

76 Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, pages 89–110, 1995. doi:10.1007/3-540-60630-0_5.

77 Philipp Hieronymi and Christian Schulz. A strong version of Cobham's theorem. In *STOC*, pages 1172–1179, 2022. doi:10.1145/3519935.3519958.

78 Thiet-Dung Huynh. The complexity of semilinear sets. *Elektron. Inf.verarb. Kybern.*, 18(6):291–338, 1982.

79 Petr Janku and Lenka Turonová. Solving string constraints with approximate Parikh image. In *EUROCAST (I)*, volume 12013, pages 491–498, 2019. doi:10.1007/978-3-030-45093-9_59.

80 Klaus Jansen and Kim-Manuel Klein. About the structure of the integer cone and its application to bin packing. *Math. Oper. Res.*, 45(4):1498–1511, 2020. doi:10.1287/MOOR.2019.1040.

81 Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey. *50 years of integer programming 1958–2008: From the early years to the state-of-the-art*. Springer, 2009. doi:10.1007/978-3-540-68279-0.

82 Anna Karapiperi, Michela Redivo-Zaglia, and Maria Rosaria Russo. Generalizations of Sylvester's determinantal identity, 2015. arXiv:1503.00519.

83 Toghrul Karimov, Florian Luca, Joris Nieuwveld, Joël Ouaknine, and James Worrell. On the decidability of Presburger arithmetic expanded with powers. In *SODA*, 2025. To appear. arXiv:2407.05191.

84 Jonathan Kirby. *An invitation to model theory*. Cambridge University Press, 2019. doi:10.1017/9781316683002.

85 Felix Klaedtke. Bounds on the automata size for Presburger arithmetic. *ACM Trans. Comput. Log.*, 9(2):11:1–11:34, 2008. doi:10.1145/1342991.1342995.

86 Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual.* BRICS, Department of Computer Science, University of Aarhus, January 2001. Notes Series NS-01-1. URL: http://www.brics.dk/mona/.

87 Dexter Kozen. *Theory of computation.* Texts in Computer Science. Springer, 2006. doi:10.1007/1-84628-477-5.

88 Georg Kreisel and Jean-Louis Krivine. *Elements of mathematical logic (model theory).* North Holland, 1967.

89 Daniel Kroening and Ofer Strichman. *Decision procedures: An algorithmic point of view.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2nd edition, 2016. doi:10.1007/978-3-662-50497-0.

90 Aless Lasaruk and Thomas Sturm. Weak integer quantifier elimination beyond the linear case. In *CASC*, pages 275–294, 2007. doi:10.1007/978-3-540-75187-8_22.

91 Niels Lauritzen. Lectures on convex sets, 2009. URL: https://users.fmf.uni-lj.si/lavric/lauritzen.pdf.

92 Niels Lauritzen. *Undergraduate convexity: From Fourier and Motzkin to Kuhn and Tucker.* World Scientific, 2013. doi:10.1142/8527.

93 Thierry Lavendhomme and Arnaud Maes. Note on the undecidability of $\langle\omega; +, P_{m,r}\rangle$. In *Definability in Arithmetics and Computability*, volume 11 of *Cahiers du Centre de logique*, pages 61–68. Academia-Bruylant, 2000. URL: http://www.cahiersdelogique.be/cahier11angl.html.

94 Antonia Lechner, Joël Ouaknine, and James Worrell. On the complexity of linear arithmetic with divisibility. In *LICS*, pages 667–676, 2015. doi:10.1109/LICS.2015.67.

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

96 Jérôme Leroux. A polynomial time Presburger criterion and synthesis for number decision diagrams. In *LICS*, pages 147–156, 2005. doi:10.1109/LICS.2005.2.

97 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*, pages 307–316, 2011. doi:10.1145/1926385.1926421.

98 Jérôme Leroux and Gérald Point. TaPAS: The Talence Presburger arithmetic suite. In *TACAS*, pages 182–185, 2009. doi:10.1007/978-3-642-00768-2_18.

99 Arnaud Maes. Revisiting Semenov's results about decidability of extensions of Presburger arithmetic. In *Definability in Arithmetics and Computability*, volume 11 of *Cahiers du Centre de logique*, pages 11–59. Academia-Bruylant, 2000. URL: http://www.cahiersdelogique.be/cahier11angl.html.

100 Victor Marsault and Jacques Sakarovitch. Ultimate periodicity of *b*-recognisable sets: A quasilinear procedure. In *DLT*, pages 362–373, 2013. doi:10.1007/978-3-642-38771-5_32.

101 Jiří Matoušek. *Lectures on discrete geometry.* Graduate Texts in Mathematics. Springer, 2002. doi:10.1007/978-1-4613-0039-7.

102 Jiří Matoušek. *Thirty-three miniatures: Mathematical and algorithmic applications of linear algebra*, volume 53 of *Student Mathematical Library.* AMS, 2010. doi:10.1090/stml/053.

103 Jirí Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$, 2014. arXiv:1406.2636.

104 Christian Michaux and Roger Villemaire. Open questions around Büchi and Presburger arithmetics. In *Logic: from Foundations to Applications: European logic colloquium*, pages 353–384. Oxford University Press, 1996. doi:10.1093/oso/9780198538622.003.0015.

105 Jasper Nalbach, Valentin Promies, Erika Ábrahám, and Paul Kobialka. FMplex: A novel method for solving linear real arithmetic problems. In *GandALF*, pages 16–32, 2023. doi:10.4204/EPTCS.390.2.

106 Danny Nguyen and Igor Pak. Enumerating projections of integer points in unbounded polyhedra. *SIAM J. Discret. Math.*, 32(2):986–1002, 2018. doi:10.1137/17M1118907.

107 Danny Nguyen and Igor Pak. Short Presburger arithmetic is hard. *SIAM J. Comput.*, 51(2):1–31, 2022. doi:10.1137/17M1151146.

108 Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL algorithm: Survey and applications.* Information Security and Cryptography. Springer, 2010. doi:10.1007/978-3-642-02295-1.

109 Tobias Nipkow. Linear quantifier elimination. *J. Autom. Reason.*, 45(2):189–212, 2010. doi:10.1007/S10817-010-9183-0.

110 Timm Oertel, Joseph Paat, and Robert Weismantel. A colorful Steinitz lemma with application to block-structured integer programs. *Math. Program.*, 204(1):677–702, 2024. doi:10.1007/S10107-023-01971-3.

111 Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332, 1978. doi:10.1016/0022-0000(78)90021-1.

112 Andreas Paffenholz. Polyhedral geometry and linear optimization (summer semester 2010), 2013. URL: http://www.mathematik.tu-darmstadt.de/~paffenholz/data/preprints/geometry_of_optimization.pdf.

113 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

114 Rohit Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

115 Hector Pasten. Definability and arithmetic. *Notices of the AMS*, 70(9):1385–1393, 2023. doi:10.1090/noti2777.

116 Guillermo A. Pérez and Ritam Raha. Revisiting parameter synthesis for one-counter automata. In *CSL*, pages 33:1–33:18, 2022. doi:10.4230/LIPICS.CSL.2022.33.

117 Jean-Éric Pin, editor. *Handbook of automata theory. Volume II. Automata in mathematics and selected applications.* EMS Press, 2021. doi:10.4171/AUTOMATA-2.

118 Ruzica Piskac and Viktor Kuncak. Linear arithmetic with stars. In *CAV*, pages 268–280, 2008. doi:10.1007/978-3-540-70545-1_25.

119 Françoise Point. On the expansion $(\mathbb{N}; +, 2^x)$ of Presburger arithmetic, 2007. Preprint. URL: https://webusers.imj-prg.fr/~francoise.point/papiers/Pres.pdf.

120 Bjorn Poonen. Hilbert's Tenth Problem over rings of number-theoretic interest. *Notes from the lectures at the Arizona Winter School "Logic and Number Theory"*, 2003. URL: http://math.mit.edu/~poonen/papers/aws2003.pdf.

121 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès des Mathématiciens des Pays Slaves, Varsovie, 1929*, pages 92–101. Skład Główny, 1930.

122 Mojżesz Presburger and Dale Jabcquette. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *History and Philosophy of Logic*, 12(2):225–233, 1991. Translation of ref. [121] and commentary. doi:10.1080/014453409108837187.

123 William W. Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, 1992. doi:10.1145/135226.135233.

124 Andrew Reynolds, Tim King, and Viktor Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods Syst. Des.*, 51(3):500–532, 2017. doi:10.1007/S10703-017-0290-Y.

125 Michel Rigo. Numeration systems: A link between number theory and formal language theory. In *DLT*, pages 33–53, 2010. doi:10.1007/978-3-642-14455-4_6.

126 Julia Robinson. Definability and decision problems in arithmetic. *J. Symb. Log.*, 14(2):98–114, 1949. doi:10.2307/2266510.

127 R. Tyrrell Rockafellar. *Convex analysis.* Princeton University Press, 1970. doi:10.1515/9781400873173.

128 Bruno Scarpellini. Complexity of subcases of Presburger arithmetic. *Trans. Am. Math. Soc.*, 284(1):203–218, 1984. doi:10.1090/S0002-9947-1984-0742421-9.

129 Alexander Schrijver. *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

130 Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Trans. Comput. Log.*, 6(3):634–671, 2005. doi:10.1145/1071596.1071602.

131 Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *ICALP*, pages 1136–1149, 2004. doi:10.1007/978-3-540-27836-8_94.

132 Aleksei L. Semenov. On certain extensions of the arithmetic of addition of natural numbers. *Math. USSR Izv.*, 15(2):401–418, 1980. doi:10.1070/IM1980v015n02ABEH001252.

133 Aleksei L. Semenov. Logical theories of one-place functions on the set of natural numbers. *Math. USSR Izv.*, 22(3):587–618, 1984. doi:10.1070/IM1984v022n03ABEH001456.

134 Jeffrey Shallit. *The logical approach to automatic sequences: Exploring combinatorics on words with Walnut*, volume 482 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 2022. doi:10.1017/9781108775267.

135 Alexander Shen and Nikolay K. Vereshchagin. *Computable functions*, volume 19 of *Student Mathematical Library.* AMS, 2003. doi:10.1090/stml/019.

136 Michael Sipser. *Introduction to the theory of computation.* Cengage Learning, 2013. 3rd ed.

137 Ryan Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR 84-639, Department of Computer Science, Cornell University, Ithaca, New York, September 1984. URL: https://hdl.handle.net/1813/6478.

138 Mikhail R. Starchak. Positive existential definability with unit, addition and coprimeness. In *ISSAC*, pages 353–360, 2021. doi:10.1145/3452143.3465515.

139 Mikhail R. Starchak. On the existential arithmetics with addition and bitwise minimum. In *FoSSaCS*, pages 176–195, 2023. doi:10.1007/978-3-031-30829-1_9.

140 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational Horn clauses. In *CADE*, pages 337–352, 2005. doi:10.1007/11532231_25.

141 Volker Weispfenning. The complexity of almost linear diophantine problems. *J. Symb. Comput.*, 10(5):395–404, 1990. doi:10.1016/S0747-7171(08)80051-X.

142 Volker Weispfenning. Complexity and uniformity of elimination in Presburger arithmetic. In *ISSAC*, pages 48–53, 1997. doi:10.1145/258726.258746.

143 Volker Weispfenning. Mixed real-integer linear quantifier elimination. In *ISSAC*, pages 129–136, 1999. doi:10.1145/309831.309888.

144 Herbert S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *Am. Math. Mon.*, 85(7):562–565, 1978. doi:10.2307/2320864.

145 H. Paul Williams. *Logic and integer programming*, volume 130 of *International Series in Operations Research & Management Science.* Springer, 2009. doi:10.1007/978-0-387-92280-5.

146 Hao Wu, Yu-Fang Chen, Zhilin Wu, Bican Xia, and Naijun Zhan. A decision procedure for string constraints with string/integer conversion and flat regular constraints. *Acta Inform.*, 61(1):23–52, 2024. doi:10.1007/S00236-023-00446-4.

147 Dennis Yurichev. SAT/SMT by example. Version of May 12, 2024. URL: https://smt.st/.

Advances in Algorithmic Meta Theorems

Sebastian Siebertz
University of Bremen, Germany

Alexandre Vigny
University Clermont Auvergne, France

Abstract

Tractability results for the model checking problem of logics yield powerful algorithmic meta theorems of the form:

> *Every computational problem expressible in a logic $\mathcal{L}$ can be solved efficiently on every class $\mathscr{C}$ of structures satisfying certain conditions.*

The most prominent logics studied in the field are (counting) monadic second-order logic (C)MSO and first-order logic FO and its extensions. The complexity of CMSO model checking in general and of FO model checking on monotone graph classes is very well understood. In recent years there has been a rapid and exciting development of new algorithmic meta theorems. On the one hand there has been major progress for FO model checking on hereditary graph classes. This progress was driven by the development of a combinatorial structure theory for the logically defined monadically stable and monadically dependent graph classes, as well as by the advent of the new width measure twinwidth. On the other hand new algorithmic meta theorems for new logics with expressive power between FO and CMSO offer a new unifying view on methods like the irrelevant vertex technique and recursive understanding. In this paper we overview the recent advances in algorithmic meta theorems and provide rough sketches for the methods to prove them.

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

Keywords and phrases Algorithmic meta theorems, monadic second-order logic, first-order logic, disjoint paths logic, algorithmic graph structure theory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.2

Category Invited Paper

1 Logics Expressing Computational Problems – Descriptive Complexity

1.1 Logics to Express Computational Problems

Logic provides a universal and machine independent language to formally define computational problems. For example, the k-Colorability problem for any fixed k can be expressed by the following sentence of monadic second-order logic MSO:

$$\varphi_k := \exists M_1 \dots \exists M_k \big(\forall x \bigvee_{1\leq i\leq k} M_i(x) \wedge \bigwedge_{1\leq i\leq k} \forall x \forall y (E(x,y) \rightarrow \neg(M_i(x) \wedge M_i(y)))\big).$$

In this sentence, we existentially quantify k sets of vertices $M_1, \dots, M_k$ representing the k colors. We then state that every element is contained in (at least) one of the M_i, that is, is assigned at least one color. Finally we verify that no two adjacent vertices are assigned the same color. We write $\mathbb{A} \models \varphi$ if a structure $\mathbb{A}$ satisfies a sentence φ, in other words, if the structure is a *model* of φ. In this case, for a graph G, we have $G \models \varphi_k$ if and only if G is k-colorable. The algorithmic problem of testing whether a given structure is a model of a given sentence of a logic $\mathcal{L}$ is called the *model checking problem* for $\mathcal{L}$.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 2; pp. 2:1–2:29
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Second-order logic SO can quantify over relations of arbitrary arity as well as over elements of a structure and is closed under the Boolean connectives $\wedge, \vee, \neg$. Monadic second-order logic MSO is the fragment of SO that allows quantification only over unary relations, also called monadic predicates, and $\exists$SO is the existential fragment of SO.

Logic can express many problems, and in fact, by Fagin's celebrated theorem [49, 50], every NP-property can be expressed by an $\exists$SO sentence. Furthermore, model checking for every fixed $\exists$SO sentence is in NP. Whenever a logic $\mathcal{L}$ satisfies these two conditions for a complexity class $\mathcal{K}$: for every $\mathcal{K}$-property there is an $\mathcal{L}$ sentence defining it, and the model checking problem for every fixed sentence of $\mathcal{L}$ is in $\mathcal{K}$, we say that *$\mathcal{L}$ captures $\mathcal{K}$*. Hence, Fagin's Theorem states that $\exists$SO captures NP. This result was extended by Stockmeyer [114] who observed that full second-order logic captures the polynomial hierarchy. In the area of *descriptive complexity theory* many other logics capturing complexity classes were studied, for example, transitive closure logic captures NLogSpace on ordered structures [79] and least fixed-point logic LFP captures PTime on ordered structures [78, 120]. Hence, the question whether $\mathsf{P} \neq \mathsf{NP}$ becomes equivalent to the purely logical question, independent of machine models, whether on ordered structures every $\exists$SO formula is equivalent to an LFP formula. One of the main open questions in this area is whether there exists a logic $\mathcal{L}$ capturing PTime also on unordered structures [15, 76]. We refer to the the textbook of Immerman [80] for more background.

1.2 Data and Combined Complexity

Note that in the capturing results above we consider the *data complexity* of the model checking problem, that is, the complexity for testing the truth of a fixed formula on a given input structure. In the following we will be interested in the *combined complexity*, where both the structure and the formula are part of the input. Model checking a second-order formula φ quantifying over x d-ary relations and y elements on an structure of size $n \geq 2$ can be done in space $\mathcal{O}(x \cdot n^d + y \cdot \log n) \subseteq \mathcal{O}(|\varphi| \cdot n^{|\varphi|})$, and hence in time $2^{\mathcal{O}(|\varphi| \cdot n^{|\varphi|})}$, by a straight-forward algorithm that recursively iterates over all possible instantiations of the quantified relations and elements. Already the combined complexity of $\exists$SO is complete for NExpTime. For MSO space $\mathcal{O}(x \cdot n + y \cdot \log n) \subseteq \mathcal{O}(|\varphi| \cdot n)$, and hence time $2^{\mathcal{O}(|\varphi| \cdot n)}$ is sufficient. The combined complexity of MSO model checking is PSPACE-complete [115]. We refer to the textbooks [68, 89] for more background.

1.3 The Complexity and Shortcomings of First-Order Logic

First-order logic FO uses only quantification over element variables. It is much weaker than the above second-order logics but still can express many interesting properties, such as the existence of an independent set of a fixed size k, dominating set of fixed size k, and many more. The complexity of FO model checking is highly relevant as FO forms the logical core of the database query language SQL [19], and query evaluation, enumeration and counting are among the most important problems for databases. The existential conjunctive fragment of FO corresponds to conjunctive queries in database theory. The data complexity of FO is in AC^0 [3] (a circuit complexity class representing constant parallel time), while the combined complexity is PSPACE-complete even on structures with only two elements [115, 120]. The combined complxity of conjunctive queries (existential conjunctive FO) is NP-complete on structures with at least two elemements [16]. A first-order formula with y quantifiers on a structure with n elements can be evaluated in space $\mathcal{O}(y \cdot \log n) \subseteq \mathcal{O}(|\varphi| \cdot \log n)$, and hence in time $n^{\mathcal{O}(|\varphi|)}$. Assuming the exponential time hypothesis ETH, this running time cannot be improved to $n^{o(|\varphi|)}$ [17].

First-order logic has two main shortcomings. First, it cannot express general cardinality properties (except fixed hard-coded properties up to a fixed threshold, and this shortcoming is also shared by MSO). This has led to the extension by counting and arithmetic, which are particularly relevant for the database community, see e.g. [4, 48, 74, 80, 87, 88, 110, 118] and the references therein. Second, FO can express only local problems. For example, FO cannot even express the algorithmically extremely simple problem of whether a graph is connected. This shortcoming has classically been addressed by adding transitive-closure or fixed-point operators, leading e.g. to the transitive closure logic FO+TC (which captures NLogSpace on ordered structures [79]), and fixed-point logics (which capture PTime on ordered structures [78, 120]).

2 A Fine-Grained View on the Model Checking Problem – Parameterized Complexity

2.1 Parameterized Complexity and the A- and W-Hierarchy

The distinction between combined complexity and data complexity of the model checking problems gives already some insights on the role of the formula and the structure in the complexity of the problem. An even finer analysis can be given when studying the *parameterized complexity* of the problem. Parameterized complexity theory provides a multi-variate approach that takes into account additional parameters, besides the input size, that allow for a fine-grained analysis of the complexity of problems [27, 34, 35, 53]. A problem is called *fixed-parameter tractable* with respect to a parameter k if it can be solved in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function and n is the input size.

One natural parameter to be considered for the model checking problem is the size of the input formula φ. However, with respect to this parameter already query evaluation for conjunctive queries in W[1]-hard and the counting problem is #W[1]-hard already for acyclic conjunctive queries [5]. The W-hierarchy is a hierarchy of parameterized complexity classes $\mathsf{FPT} \subseteq \mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \ldots \subseteq \mathsf{AW}[\star]$, which is conjectured to be strict, and establishing hardness for one of the levels of the hierarchy is widely accepted as a proof for fixed-parameter intractability [33]. There is a second prominent hierarchy, the A-hierarchy [52], $\mathsf{FPT} \subseteq \mathsf{A}[1] \subseteq \mathsf{A}[2] \subseteq \ldots \subseteq \mathsf{AW}[\star]$. This hierarchy is also conjectured to be strict, and we have $\mathsf{W}[1] = \mathsf{A}[1]$ and $\mathsf{W}[i] \subseteq \mathsf{A}[i]$ for all $i \geq 2$.

2.2 The Need for Structural Parameters

The FO model checking problem yields natural complete problems for the levels of the hierarchy. We need a bit of notation. By Σ_0 and Π_0 denote the set of quantifier-free formulas. For $t \geq 0$ define Σ_{t+1} be the set of formulas $\exists x_1 \ldots \exists x_k\, \varphi$, where $\varphi \in \Pi_t$, and Π_{t+1} the set of all formulas $\forall x_1 \ldots \forall x_k\, \varphi$, where $\varphi \in \Sigma_t$. For $t, u \geq 1$, a Σ_t-formula is furthermore in $\Sigma_{t,u}$ if all quantifier blocks after the leading existential block have length at most u. For all $t \geq 1$, model checking for $\Sigma_{t,1}$-formulas is complete for $\mathsf{W}[t]$. For example, the $\Sigma_{1,1}$-formula $\exists x_1 \ldots \exists x_k \big(\bigwedge_{1 \leq i < j \leq k} (x_i \neq x_j \wedge \neg E(x_i, x_j)) \big)$ expresses the Independent Set problem, the most prominent W[1]-complete problem. Similarly, the $\Sigma_{2,1}$-formula $\exists x_1 \ldots \exists x_k \forall y \big(\bigvee_{1 \leq i \leq k} (y = x_i) \vee \bigvee_{1 \leq i \leq k} E(y, x_i) \big)$ expresses the dominating set problem, which is well-known to be W[2]-complete. For all $t \geq 1$, the model checking problem for Σ_t is complete for A[t]. Note that $\Sigma_{1,1} = \Sigma_1$, which immediately implies W[1] = A[1]. Model checking for full FO is complete for the class AW[$\star$].

The situation for MSO is even worse. Since already 3-Colorability, which is expressed by a fixed MSO-formula, is NP-complete, MSO model checking parameterized by formula length is para-NP-hard.

2.3 MSO and the Parameters Treewidth and Cliquewidth

Hence, the parameter $|\varphi|$ alone still does not yield the desired fine-grained theory of tractability for the model checking problem. For a tangible theory we need to take further structural parameters into account. This is where graph theory enters the stage, which offers a wealth of parameters to classify the complexity of inputs. This classification can be lifted from graphs to general structures by considering their Gaifman graphs or by considering the incidence encoding (see Section 3.4). In a celebrated result Courcelle in 1990 established that every MSO definable property can be tested in linear time on graphs of bounded treewidth [22]. More precisely, he established that the model checking problem for MSO is fixed-parameter tractable with respect to the parameters formula length and treewidth, that is, solvable in time $f(|\varphi|, tw) \cdot n$, where tw is the treewidth of the input graph. This result extends to counting MSO, CMSO [22], and to graphs with bounded cliquewidth [24], that is, CMSO model checking can be solved in time $f(|\varphi|, cw) \cdot n$, where cw is the cliquewidth of the input graph. Furthermore, it was proved that efficient (C)MSO model checking cannot be extended to classes of unbounded treewidth or cliquewidth, assuming further mild closure conditions and the standard complexity theoretic assumptions [26, 63, 84, 85, 86, 93].

2.4 More Parameters for FO

In 1996 Seese [112] established that FO model checking is linear time solvable on graphs with bounded maximum degree, that is, solvable in time $f(\Delta, |\varphi|) \cdot n$, and thereby initiated the fine-grained study of the parameterized complexity of first-order model checking with respect to structural parameters. His result was extended to more and more general classes of sparse graphs [52, 56, 31, 43, 83, 73]. The last result of Grohe, Kreutzer and Siebertz [73] shows that every FO definable property of graphs is decidable in nearly linear time on every nowhere dense class of graphs. Nowhere dense classes of graphs are very general classes of sparse graphs [96] and turn out to be a tractability barrier for FO model checking on monotone classes (that is, classes that are closed under taking subgraphs): if a monotone class of graphs is not nowhere dense, then testing first-order properties for inputs from this class is as hard as for general graphs [43, 83].

2.5 Algorithmic Meta Theorems

Note that the nature of these results is different from those in descriptive complexity theory. They have a much more algorithmic, rather than a complexity theoretic flavor. In particular, the development for the model checking problems parallels that of the development in parameterized complexity, which in the early days was strongly driven by the development of the graph minors theory of Robertson and Seymour [103] and the rise of treewidth as one of the most important structural parameters. The model checking results not only stand by themselves but additionally capture the essence and limits of fundamental algorithmic techniques, such as dynamic programming and compositionality, the locality based method, game based methods. Newer results are strongly based on recursive understanding and the irrelevant vertex technique. In their seminal surveys Martin Grohe and Stephan Kreutzer coined the expression *algorithmic meta theorems* [70, 72, 83].

> *Every computational problem expressible in a logic $\mathcal{L}$ can be solved efficiently on every class $\mathscr{C}$ of structures satisfying certain conditions.*

Since the groundbreaking work of Courcelle [22], algorithmic meta theorems have become an essential tool in the toolbox of parameterized complexity theory. Formalizing a problem in a logic yields a fast and convenient, and yet formal proof for its tractability on certain classes of structures. At this time, the main goal in the area is to find the most general classes of structures that allow for fixed-parameter tractable FO model checking. A recent trend is also to find new logics with expressive power between that of FO and MSO that capture certain algorithmic techniques.

2.6 The Limits of Tractability

As already mentioned, the complexity of CMSO model checking is essentially settled. For plain MSO the question is related to a conjecture of Seese [111], stating that if a class of graphs has a decidable satisfiability problem for MSO, then it has bounded cliquewidth. A stronger version conjectures (formulated in the contrapositive) that if a class $\mathscr{C}$ has unbounded cliquewidth, then MSO can encode all graphs in $\mathscr{C}$, which essentially yields intractable MSO model checking on classes of unbounded cliquewidth. The weaker statement for CMSO was proved by a connection between CMSO and vertex minors [26].

The tractability limit of FO model checking on monotone graph classes is constituted by nowhere dense graph classes. Consequently, attention has shifted to study more general, hereditary classes (that is, classes that are closed under taking induced subgraphs).

2.7 New Directions for FO Model Checking

Recently, there has been very exciting progress on algorithmic meta theorems, which essentially goes into three directions. The first direction is inspired by the classical *interpretation method* (see Section 5). Given a class $\mathscr{C}$ of structures that is well understood, the interpretation method sometimes allows to lift to classes that can be logically interpreted in $\mathscr{C}$. The classical proof of Courcelle's Theorem can be seen as a prime example of the interpretation method: classes with bounded treewidth (and with bounded cliquewidth) can be encoded by MSO in colored trees, which are very easy to handle. When it is possible to encode a class $\mathscr{D}$ in a sparse class $\mathscr{C}$ via an FO transduction we call $\mathscr{D}$ a *structurally sparse class*. Note that it is still a challenge to recover a sparse preimage from a dense but structurally sparse input graph. Results for bounded degree graphs were lifted to classes with structurally bounded degree [57, 58], from bounded expansion classes to classes with structurally bounded expansion [59] and from nowhere dense classes to structurally nowhere dense classes [40].

The second direction was inspired by a result of Adler and Adler [1] who observed that nowhere dense classes are *monadically dependent (NIP)* and *monadically stable*. The notions of dependence and stability are key notions from classical model theory [113] and are defined by forbidden interpretable configurations. This connection to model theory brought new notions of structural tameness and a large toolbox for infinite theories to graph theory. The challenge to develop a combinatorial and algorithmic theory for finite graphs was quickly accepted and led to many strong results, see e.g. [6, 13, 36, 37, 39, 38, 40, 41, 42, 59, 60, 62, 81, 92, 94, 97, 98, 99, 119].

A class is monadically dependent if in colorings of the graphs from the class one cannot interpret all graphs; a natural candidate for the limit of tractability for FO model checking, and soon it was conjectured that monadic dependence constitutes the tractability boundary for FO model checking on hereditary classes [9, 40, 57, 100]. A big step towards this conjecture was taken by Dreier et al. [37, 40] who proved that model checking is tractable on monadically stable classes, which are important subclasses of monadically dependent classes. Furthermore, the hardness part of the conjecture was established in [37].

Some indication for the truth of the conjecture is given by the third direction. Bonnet et al. [12] introduced the new notion of *twinwidth*, which had an immense impact in structural graph theory in the past few years. Bonnet et al. proved that FO model checking is tractable on classes with bounded twinwidth assuming that we are given a contraction sequence, witnessing the boundedness of twinwidth, together with the input. It turns out that a class of ordered graphs has bounded twinwidth if and only if it is monadically dependent [9]. Furthermore, on ordered structures bounded twinwidth contraction sequences can be efficiently computed, and model checking on hereditary ordered classes beyond bounded twinwidth is intractable [9]. Hence, the conjecture is true on ordered graphs The main challenge in the area remains to resolve the conjecture for general hereditary classes.

One consequence of the result on twinwidth that is seldomly mentioned is that it implies tractability of *order-invariant* FO on classes of bounded twinwidth (assuming a contraction sequence is given with the input). An order-invariant formula may use a symbol for a linear order and must satisfy the semantic property that its truth on ordered structures must be independent of the chose order. Note that it is undecidable to decide if a formula is order-invariant [89], order-invariance must be guaranteed for the formulas that are input to the model checking problem. Order-invariant MSO is more expressive than CMSO [66] but the model checking problem is tractable on the same classes [47]. An unpublished result of Gurevich states that the expressive power of order-invariant FO is stronger than that of plain FO (see, e.g., Theorem 5.3 of [89] for a presentation of the result). Model checking for the weaker successor-invariant FO is tractable on classes with bounded expansion [47], for order-invariant FO the limit of tractability is wide open.

2.8 Why do we not study the other classical logics

The fact that first-order logic can only express local problems is classically addressed by adding transitive-closure or fixed-point operators [45, 2, 90]. These logics however, do not have a rich algorithmic theory: Even the model checking problem for the very restricted monadic transitive-closure logic TC^1 is $\mathsf{AW}[\star]$-hard on planar graphs of maximum degree at most 3 [70]. Model checking for full SO is intractable even on colored paths, as PSPACE-computations on strings can be simulated.

2.9 FO plus connectivity, compound logic, and more

This has motivated recent approaches to introduce new logics whose expressive power lies between FO and MSO, which are tailored to express algorithmic graph problems and which are still tractable on interesting graph classes. Examples include *Separator logic* [5, 109], *Disjoint paths logic* [109], *Compound logics for modification problems* [54], and CMSO/tw as the fragment of MSO on graphs obtained by appropriately restricting set quantification [107]. These logics can express many problems that are studied in parameterized complexity, and hence provide very useful meta theorems. Again, these logics are not only relevant for the meta theorems they provide, but they also capture important algorithmic paradigms.

2.10 Advances in Algorithmic Meta Theorems

It has been about 15 years since Grohe and Kreutzer wrote their seminal expositions on algorithmic meta theorems [70, 72, 83]. They identified the following key methods.

1. The Automata Theoretic Method, translating CMSO formulas into automata that can be run on trees.
2. The Reduction or Interpretation Method, allowing to translate tractability results between classes of structures via logical interpretations.
3. The Composition Method, which allow to lift results to structures that are composed of simpler pieces from these pieces.
4. Locality Based Methods for FO, exploiting that FO can express only local properties.
5. Coloring and Quantifier-Elimination, which allows to simplify to quantifier-free formulas by an algorithmic enrichment of the signature.

Of course, the foundational methods have not changed. In this paper we aim to give an overview over new techniques and the recent extensions of the well-established methods.

The automata method has been generalized from trees to trees that are additionally augmented with graph structures [102]. This allows in many cases to directly combine the composition method with the automata method.

The composition method has gained new momentum with the advent of the method of recursive understanding developed in parameterized complexity [18, 82], and its conceptual predecessor, the decomposition into unbreakable parts [28, 29]. The technique has led to the result of Lokshtanov et al. [91], showing that the problem of deciding a CMSO property on general graphs can be reduced to the same problem on unbreakable graphs. It is also the basis for the model checking result for separator logic [102] and disjoint paths logic [108] on classes with excluded topological minors, as well as for model checking of compound logic [54] and CMSO/tw [107] on classes with excluded minors.

The locality based methods have been taken to their limits by combining locality with recursive game based decompositions of local neighborhoods on nowhere dense classes [73] and monadically stable classes [40, 37]. An appropriate extension to monadically dependent classes is one of the main open problems in the area.

The advent of twinwidth has led to a completely new foundational method of dynamic programming along contraction sequences, see e.g. [7, 8, 9, 11, 12].

2.11 Structure of this paper

We take this rapid and exciting development as a motivation to give a brief overview over the new methods for algorithmic meta theorems. We structure the paper as follows. After fixing our notation we first introduce the interpretation method and quantifier elimination as special cases of parameterized reductions between model checking problems. These remaining base cases are the extended automata based method, locality based methods, and methods based on twinwidth.

3 Preliminaries

3.1 Structures

Let us recall some basics from model theory. We refer to the textbooks [2, 46, 77, 89] for extensive background. A *signature* is a collection of relation and function symbols, each with an associated arity. Let σ be a signature. A *σ-structure* $\mathbb{A}$ consists of a non-empty set A, the *universe* of $\mathbb{A}$, together with an interpretation of each k-ary relation symbol $R \in \sigma$ as a k-ary relation $R^{\mathbb{A}} \subseteq A^k$ and an interpretation of each k-ary function symbol $f \in \sigma$ as a k-ary function $f^{\mathbb{A}} : A^k \to A$. In this work we assume that all structures are finite (i.e. have a finite universe and a finite signature). We write $|\mathbb{A}|$ for the size of the universe of $\mathbb{A}$ and $\|\mathbb{A}\|$ for the size of an encoding of $\mathbb{A}$, e.g. in the standard incidence encoding.

In this work we mostly consider the following types of structures:

- *Colored graphs* are σ-structures, where σ consists of a single binary relation E and unary relations, with the property that E is symmetric and anti-reflexive.
- *Guided pointer structures* are σ-structures, where σ consists of a single binary relation E, unary relations, and unary functions, with the property that E is symmetric and anti-reflexive, and that every function $f \in \sigma$ is *guided*, meaning that if $f(u) = v$, then $u = v$ or $uv \in E(G)$.
- *Ordered graphs* are σ-structures, where σ consists of a two binary relations E and $<$, with the property that E is symmetric and anti-reflexive, and $<$ is a linear order.
- *Partial orders* are σ-structures, where σ consists of binary relation $\sqsubseteq$ that is interpreted as a partial order, in many cases a linear order or tree order.

3.2 First-order and monadic second-order logic

We first define first-order logic FO and monadic second-order logic MSO (over the signature σ). The fancier logics will be defined in the sections where they are first discussed. We assume an infinite supply $\textsc{Var}_1$ of first-order variables and an infinite supply $\textsc{Var}_2$ of monadic second-order variables. Every variable is a term, and if $t_1, \ldots, t_k$ are terms and $f \in \sigma$ is a k-ary function symbol, then also $f(t_1, \ldots, t_k)$ is a term. $\mathsf{FO}[\sigma]$ formulas are built from the atomic formulas $t_1 = t_2$, where t_1 and t_2 are terms, and $R(t_1, \ldots, t_k)$, where $R \in \sigma$ is a k-ary relation symbol and $t_1, \ldots, t_k$ are terms, by closing under the Boolean connectives $\neg$, $\wedge$ and $\vee$, and by existential and universal quantification $\exists x$ and $\forall x$.

Monadic second-order formulas are defined as first-order formulas, but further allow the use of monadic quantifiers $\exists X$ and $\forall X$, and of a membership atomic formula $x \in X$, where x is a first-order variable and X a monadic second-order variable. CMSO is the extension of MSO with modulo counting quantifiers $\exists^{a[b]} x \varphi(x)$, expressing the existence of a modulo b elements satisfying φ.

A variable x not in the scope of a quantifier is a *free variable* (we do not consider formulas with free set second-order variables). A formula without free variables is a *sentence*. The *quantifier rank* $\mathrm{qr}(\varphi)$ of a formula φ is the maximum nesting depth of quantifiers in φ. A formula without quantifiers is called *quantifier-free*.

If $\mathbb{A}$ is a σ-structure with universe A, then an *assignment* of the variables in $\mathbb{A}$ is a mapping $\bar{a} : \textsc{Var}_1 \to A$. We use the standard notation $(\mathbb{A}, \bar{a}) \models \varphi(\bar{x})$ or $\mathbb{A} \models \varphi(\bar{a})$ to indicate that φ is satisfied in $\mathbb{A}$ when the free variables $\bar{x}$ of φ have been assigned by $\bar{a}$. We write $\models \varphi$ to express that φ is a valid sentence, that is, φ holds in every structure (of an appropriate signature). For a formula $\varphi(\bar{x})$ we define $\varphi(\mathbb{A}) := \{\bar{a} \in A^{|\bar{x}|} \mid \mathbb{A} \models \varphi(\bar{a})\}$.

Two structures $\mathbb{A}$ and $\mathbb{B}$ are $(\mathcal{L}, q)$-*equivalent*, written $\mathbb{A} \equiv_q^{\mathcal{L}} \mathbb{B}$, if they satisfy the same $\mathcal{L}$-sentences of quantifier rank at most q. The $(\mathcal{L}, q)$-*type* of an element a in $\mathbb{A}$ is the collection of all formulas $\varphi(x)$ of quantifier rank at most q such that $\mathbb{A} \models \varphi(a)$. A set of formulas is a q-*type* if it is the q-type of some element in some structure.

3.3 Interpretations and transductions

Let σ, τ be relational signatures. A *simple $\mathcal{L}$-interpretation* I of σ-structures in τ-structures is a tuple $\mathsf{I} = (\nu, (\rho_R)_{R\in\sigma})$, where $\nu(x)$ and $\rho_R(\bar{x})$ are $\mathcal{L}$-formulas, where $|\bar{x}|$ in $\rho_R(\bar{x})$ matches the arity of R. For every τ-structure $\mathbb{B}$, the σ-structure $\mathbb{A} = \mathsf{I}(\mathbb{B})$ has the universe $\nu(\mathbb{B})$ and each relation $R^{\mathbb{A}}$ is interpreted as $\rho_R(\mathbb{B})$. We say that ν *defines* the vertex set and ρ_R defines the relation R of $\mathsf{I}(\mathbb{B})$. We remark that many of the presented results generalize to interpretations in powers, however, we refrain from presenting them in this generality as interpretations in powers do not preserve graph properties such as bounded treewidth/cliquewidth and monadic dependence/stability.

A *monadic lift* of a τ-structure $\mathbb{B}$ is a τ^+-expansion $\mathbb{B}^+$ of $\mathbb{B}$, where τ^+ is the union of τ and a set of unary relation symbols.

An $\mathcal{L}$-transduction T of σ-structures from τ-structures is a simple $\mathcal{L}$-interpretation of σ-structures in τ^+-structures, where τ^+ is an extension of τ by unary relation symbols as above. We say that a σ-structure $\mathbb{A}$ can be T-transduced from a τ-structure $\mathbb{B}$ if there is a τ^+-lift $\mathbb{B}^+$ of $\mathbb{B}$ such that $\mathbb{A} = \mathsf{T}(\mathbb{B})$. A class $\mathscr{C}$ of τ-structures can be T-transduced from a class $\mathscr{D}$ of σ-structures if for every structure $\mathbb{A} \in \mathscr{C}$ there exists a structure $\mathbb{B} \in \mathscr{D}$ such that $\mathbb{A}$ can be T-transduced from $\mathbb{B}$. We also say that T is a transduction of $\mathscr{C}$ in $\mathscr{D}$ or from $\mathscr{D}$ onto $\mathscr{C}$. A class $\mathscr{D}$ of τ-structures can be transduced from a class $\mathscr{C}$ of σ-structures if it can be T-transduced from $\mathscr{C}$ for some transduction T.

3.4 Gaifman graphs and incidence structures

The *Gaifman graph* of a σ-structure $\mathbb{A}$ with universe A is the graph with vertex set A, where u and v are adjacent if they belong jointly to some relation $R^{\mathbb{A}}$ for $R \in \sigma$, or if one is the image of the other by some function $f^{\mathbb{A}}$ for $f \in \Sigma$.

For a relational signature σ, the *incidence graph* of a σ-structure $\mathbb{A}$ with universe A is the colored graph with vertices A and one vertex for each tuple $\bar{v}$ appearing in a relation $R^{\mathbb{A}}$ (we take multiple copies if a tuple appears in several relations). If $\bar{v} = (v_1, \ldots, v_k)$ and $\bar{v} \in R^{\mathbb{A}}$, then the vertex $\bar{v}$ in the incidence graph is marked with a color R and connected with an edge of color i with the vertex v_i for $1 \le i \le k$. Note that $\mathbb{A}$ is interpretable from its incidence graph, but the converse is in general not true.

4 Reductions Between Algorithmic Meta Theorems

Recall that the model checking problem for $\mathcal{L}$ on a class $\mathscr{C}$ of structures is the problem: given $\mathbb{A} \in \mathscr{C}$ and $\varphi \in \mathcal{L}$, decide whether $\mathbb{A} \models \varphi$. We denote it by $\mathrm{MC}(\mathcal{L}, \mathscr{C})$.

▶ **Definition 1.** *A* non-uniform algorithmic meta theorem *is a result of the form: Let $\mathcal{L}$ be a logic and $\mathscr{C}$ a class of structures. Then for all sentences $\varphi \in \mathcal{L}$ there exists an algorithm that given $\mathbb{A} \in \mathscr{C}$ decides whether $\mathbb{A} \models \varphi$ in time $f(\varphi) \cdot \|\mathbb{A}\|^{\mathcal{O}(1)}$ for some function f.*

▶ **Definition 2.** *An* algorithmic meta theorem *is a result of the form: Let $\mathcal{L}$ be a logic and $\mathscr{C}$ be a class of structures. Then* $\mathrm{MC}(\mathcal{L}, \mathscr{C})$ *is fixed-parameter tractable, that is, there is an algorithm that on input $\mathbb{A} \in \mathscr{C}$ and $\varphi \in \mathcal{L}$ decides whether $\mathbb{A} \models \varphi$ in time $f(\varphi) \cdot \|A\|^{\mathcal{O}(1)}$ for a computable function f.*

We define reductions between algorithmic meta theorems simply as parameterized reductions between the corresponding model checking problems.

▶ **Definition 3.** *Let* $\mathrm{MC}(\mathcal{L}_1, \mathscr{C})$ *and* $\mathrm{MC}(\mathcal{L}_2, \mathscr{D})$ *be model checking problems. A* parameterized reduction *from* $\mathrm{MC}(\mathcal{L}_1, \mathscr{C})$ *to* $\mathrm{MC}(\mathcal{L}_2, \mathscr{D})$ *is a function* R *that maps instances* $(\mathbb{A}, \varphi)$ *of* $\mathrm{MC}(\mathcal{L}_1, \mathscr{C})$ *to instances* $(\mathbb{B}, \psi)$ *of* $\mathrm{MC}(\mathcal{L}_2, \mathscr{D})$ *such that*

1. $\mathbb{A} \models \varphi \Leftrightarrow \mathbb{B} \models \psi$,
2. $|\psi| \leq f(|\varphi|)$ *for a computable function* f*, and*
3. $(\mathbb{B}, \psi)$ *can be computed in time* $g(|\varphi|) \cdot \|(\mathbb{A}, \varphi)\|^{\mathcal{O}(1)}$ *for a computable function* g*.*

We say that $\mathrm{MC}(\mathcal{L}_1, \mathscr{C})$ *reduces to* $\mathrm{MC}(\mathcal{L}_2, \mathscr{D})$*, and write* $\mathrm{MC}(\mathcal{L}_1, \mathscr{C}) \leq_{fpt} \mathrm{MC}(\mathcal{L}_2, \mathscr{D})$*, if there exists a parameterized reduction from* $\mathrm{MC}(\mathcal{L}_1, \mathscr{C})$ *to* $\mathrm{MC}(\mathcal{L}_2, \mathscr{D})$*.*

We now have the standard reduction lemma.

▶ **Lemma 4.** *Let* $\mathrm{MC}(\mathcal{L}, \mathscr{C})$ *and* $\mathrm{MC}(\mathcal{L}', \mathscr{C}')$ *be model checking problems with* $\mathrm{MC}(\mathcal{L}, \mathscr{C}) \leq_{fpt} \mathrm{MC}(\mathcal{L}', \mathscr{C}')$*. If* $\mathrm{MC}(\mathcal{L}', \mathscr{C}')$ *is fixed-parameter tractable, then* $\mathrm{MC}(\mathcal{L}, \mathscr{C})$ *is fixed-parameter tractable.*

4.1 Example: Reducing Separator Logic to FO with MSO Atoms on Augmented Trees

As already discussed above, FO falls short of being able to express algorithmic problems that involve *non-local* properties. For example, FO cannot express the very simple algorithmic question whether two vertices are connected. *Separator logic*, denoted by FO+conn and independently introduced in [5, 109], enriches FO with connectivity predicates that are tailored to express algorithmic graph properties that are commonly studied in parameterized algorithmics. Separator logic is obtained from FO by adding the atomic predicates $\mathsf{conn}_k(x, y, z_1, \ldots, z_k)$ that hold true in a graph if there exists a path between (the valuations of) x and y after (the valuations of) $z_1, \ldots, z_k$ have been deleted. Separator logic can express many interesting problems such as the Feedback Vertex Set problem and Elimination Distance problems to first-order definable classes. It was proved in [102] that model checking for separator logic is fixed-parameter tractable on classes excluding a topological minor, and for subgraph-closed classes, this result cannot be extended to more general classes (assuming a further condition on the efficiency of encoding required for the hardness reduction).

Separator logic yields a first nice example of the reduction method. First, it was observed in [102] that separator logic on highly connected graphs can be reduced to plain FO. Let us formalize the concept of high connectivity.

A *separation* in a graph G is a pair (A, B) of vertex subsets such that $A \cup B = V(G)$ and there are no edges in G between $A \setminus B$ and $B \setminus A$. The *order* of the separation is the cardinality of the *separator* $A \cap B$. A vertex subset X is (q, k)*-unbreakable* in a graph G if for every separation (A, B) in G of order at most k in G, either $|A \cap X| \leq q$ or $|B \cap X| \leq q$. Intuitively, a separation of order k cannot break X in a balanced way: one of the sides must contain at most q vertices of X.

Now observe that for a (q, k)-unbreakable graph G, the query $\mathsf{conn}_k(u, v, x_1, \ldots, x_k)$ can be expressed in plain FO. The query fails if and only if there is a set A of at most q vertices that contains exactly one of the vertices u and v, and such that all neighbors of vertices of A outside of A are contained in $\{x_1, \ldots, x_k\}$. The existence of such a set of q vertices can be expressed using q existential quantifiers followed by a universal quantifier. So every formula that uses only conn_k predicates can be rewritten as a plain FO formula on (q, k)-unbreakable graphs, as long as q is a constant. Denote by FO+conn_k the fragment of FO+conn that uses only conn_k-predicates and denote by $\mathscr{B}_{q,k}$ the class of (q, k)-unbreakable graphs.

▶ **Lemma 5.** $MC(\mathit{FO}{+}\mathit{conn}_k, \mathscr{B}_{q,k}) \leq_{fpt} MC(\mathit{FO}, \mathscr{B}_{q,k})$.

Now, for general graphs, we reduce separator logic to FO with MSO atoms, denoted $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq))$, over augmented trees, using the following decomposition theorem. Let us first define trees and tree decompositions.

A (rooted) tree is an acyclic and connected graph T with a distinguished root vertex r. We write $\mathsf{parent}(x)$ for the parent of a node x of T, and $\mathsf{children}(x)$ is the set of children of x in T. We define $\mathsf{parent}(r) = \bot$. A vertex $x \in V(T)$ is an ancestor of a vertex $y \in V(T)$, written $x \preceq_T y$, or simply $x \preceq y$ if T is clear from the context, if x lies on the unique path between y and the root r. Note that hence every node is an ancestor of itself. For nodes $x, y \in V(T)$, we write $\mathrm{lca}(x, y)$ for the least common ancestor of x and y in T. Note that x is an ancestor of y if and only if $\mathrm{lca}(x, y) = x$.

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \mathsf{bag})$, where T is a rooted tree and $\mathsf{bag}\colon V(T) \to 2^{V(G)}$ is a mapping that assigns to each node x of T a *bag* $\mathsf{bag}(x) \subseteq V(G)$, such that

- for every $u \in V(G)$, the set of nodes $x \in V(T)$ satisfying $u \in \mathsf{bag}(x)$ induces a connected and nonempty subtree of T, and
- for every edge $uv \in E(G)$, there exists a node $x \in V(T)$ such that $\{u, v\} \subseteq \mathsf{bag}(x)$.

The *treewidth* of a graph G is the size of a largest bag of $\mathcal{T}$ minus 1, where $\mathcal{T}$ ranges over all tree decompositions of G.

Let $\mathcal{T} = (T, \mathsf{bag})$ be a tree decomposition and let $x \in V(T)$.

- The *adhesion* of x is

$$\mathsf{adh}(x) \coloneqq \mathsf{bag}(\mathsf{parent}(x)) \cap \mathsf{bag}(x).$$

- The *margin* of x is

$$\mathsf{mrg}(x) \coloneqq \mathsf{bag}(x) \setminus \mathsf{adh}(x).$$

- The *cone at* x is

$$\mathsf{cone}(x) \coloneqq \bigcup_{y \succeq_T x} \mathsf{bag}(y).$$

- The *component at* x is

$$\mathsf{comp}(x) \coloneqq \mathsf{cone}(x) \setminus \mathsf{adh}(x) = \bigcup_{y \succeq_T x} \mathsf{mrg}(y).$$

Observe that the margins $\{\mathsf{mrg}(x)\colon x \in V(T)\}$ are pairwise disjoint and cover the whole vertex set of G. The *adhesion* of a tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ is defined as the largest size of an adhesion, that is, $\max_{x \in V(T)} |\mathsf{adh}(x)|$.

A tree decomposition $\mathcal{T} = (T, \mathsf{bag})$ of a graph G is *strongly* (q, k)*-unbreakable* if for every $x \in V(T)$, $\mathsf{bag}(x)$ is (q, k)-unbreakable in $G[\mathsf{cone}(x)]$.

▶ **Theorem 6** ([30]). *There is a function $q(k) \in 2^{\mathcal{O}(k)}$ such that for every graph G and number k there exists a strongly $(q(k), k)$-unbreakable tree decomposition of G of adhesion at most $q(k)$. Moreover, given G and k, such a tree decomposition can be computed in time* $2^{\mathcal{O}(k^2)} \cdot |G|^2 \cdot \|G\|$.

The theorem opens two directions to approach the model checking problem for separator logic. The first approach is to use compositionality (see Section 7) and do dynamic programming over unbreakable tree decompositions. The second approach, which we follow here, works as follows. We encode the decomposition in a colored tree with additional edges between siblings (the children of some node) in the tree. We call such a structure an *augmented tree.* We will then reduce model checking of separator logic to model checking on a logic over augmented trees. The resulting problem can then be solved via the automata method (see Section 8).

More generally we may want to encode general σ-structures. In this case we represent augmented trees as relational structures equipped with the ancestor relation $\preceq_T$ of a tree T, as well as relations R relating the children of any given node, with which the tree is augmented. We will access the global connectivity of the tree with MSO that speaks only about $\preceq$, as well as over the colors of the tree. We write $\mathsf{MSO}(\preceq, A)$ for the set of MSO formulas over such signatures, where A is a finite set of unary predicates. As we will translate such formulas into tree automata, we will often call A an alphabet. We now consider FO formulas with restricted MSO atoms. For the signature σ and the alphabet A, the logic $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A), \sigma)$ denotes first-order logic over signature σ where one can use formulas of $\mathsf{MSO}(\preceq, A)$ as atomic formulas. Denote by $\mathscr{G}$ the class of all graphs and by $\mathscr{T}_{A,\sigma}$ the class of (A, σ)-augmented trees.

▶ **Theorem 7** ([102]). *$MC(\mathit{FO{+}conn}, \mathscr{G}) \leq_{fpt} MC(\mathit{FO}(\mathit{MSO}(\preccurlyeq, A), \sigma), \mathscr{T}_{A,\sigma})$, where A is a finite alphabet and σ is a signature of colored graphs.*

4.2 Example: Reducing CMSO/tw+dp on Classes with Excluded Minors to CMSO on Classes with Bouunded Treewidth

One of the classic and important problems that separator logic cannot express [109] is the Disjoint Paths problem: *Given a graph G and a set $\{(s_1, t_1), \dots, (s_k, t_k)\}$ of pairs of terminals, the question is whether G contains vertex-disjoint paths joining s_i and t_i for $1 \leq i \leq k$.* A straight-forward approach to deal with this shortcoming is to add a predicate expressing exactly this property. *Disjoint-paths logic* FO+dp is an extension of separator logic that was introduced in [109]. It extends first-order logic (FO) with atomic predicates $\mathsf{DP}_k[(x_1, y_1), \dots, (x_k, y_k)]$ expressing the existence of internally vertex-disjoint paths between x_i and y_i, for $1 \leq i \leq k$. Disjoint paths logic can express many interesting algorithmic problems, such as the disjoint paths problem, minor containment, topological minor containment, $\mathcal{F}$-topological minor deletion, and many more (see [67]). It was shown in [67] that model checking for disjoint-paths logic is fixed-parameter tractable on classes with excluded minors and in [108] that it is fixed-parameter tractable on classes with excluded topological minors.

Another logic recently introduced logic by Sau, Stamoulis and Thilikos [107] is CMSO/tw, a fragment of counting monadic second-order logic that allows only restricted quantification of sets in CMSO formulas. Recall that CMSO with no restriction on set quantification is intractable beyond graphs of bounded cliquewidth (under the standard complexity theoretic assumptions and mild closure conditions). CMSO/tw very elegantly generalizes another extension of FO that was introduced by Fomin et al. [54], the so-called *compound logic*, which is tailored so as to express general families of graph modification problems. CMSO/tw is very expressive, nevertheless, it cannot express the disjoint paths problem. Following the approach of disjoint paths logic, we may simple add an operator to express this property to obtain the logic CMSO/tw+dp.

Let G be a graph and $X \subseteq V(G)$. A graph H is an *X-rooted minor* of G if there is a collection $\mathcal{B} = \{B_x \mid x \in V(H)\}$ of pairwise disjoint connected subsets of $V(G)$, each containing at least one vertex of X, and such that, for every edge $xy \in E(H)$, there are $u \in B_x$

and $v \in B_y$ with $uv \in E(G)$. The set B_x is called the *branch set* of x in G. A graph H is a minor of G if it is a $V(G)$-rooted minor of G. Given a graph G and $X \subseteq V(G)$, the *annotated treewidth* of X in G, denoted $\mathsf{tw}(G, X)$, is the maximum treewidth of an X-rooted minor of G [117].

CMSO/tw is the restriction of CMSO where instead of using the quantifier $\exists X$ (resp. $\forall X$) for a set variable X, we have quantifiers $\exists_k X$ (resp. $\forall_k X$) for some number k, where $\exists_k X$ and $\forall_k X$ mean that the quantification is applied on vertex or edge sets X with $\mathsf{tw}(G, X) \leq k$. For the case of quantification on an edge set $X \subseteq E(G)$, $\mathsf{tw}(G, X) \leq k$ means that the set of the endpoints of the edges in X has annotated treewidth at most k. CMSO/tw+dp is the extension of CMSO/tw with the disjoint paths operators.

▶ **Theorem 8** ([107])**.** *Let $\mathscr{C}$ be a class excluding some minor. Then there exists a class $\mathscr{D}$ of structures with bounded treewidth such that $MC(\mathsf{CMSO/tw+dp}, \mathscr{C}) \leq_{fpt} MC(\mathsf{CMSO}, \mathscr{D})$.*

The theorem is proved using the *irrelevant vertex technique*, which was first introduced in [104]. In particular, it uses the flat wall theorem, in the recent refined formulation of [106]. A similar scheme was also applied in the works [54, 55, 67]. We stress that the irrelevant vertex technique in the proof of Theorem 8 does not simply remove vertices from the input graph. In particular, the theorem does not yield a graph (structure) of bounded treewidth that is logically equivalent to the input graph. This would yield a contradiction with the undecidability of the satisfiability problem of CMSO (and FO). It is crucial that also the input formula is rewritten, depending on the input graph.

The model checking algorithm for FO+dp on classes with excluded topological minors combines the approaches of [102] and [107]. First, the input graph is decomposed into unbreakable parts using Theorem 6. On each part, we distinguish two cases. When a part excludes a minor, we can apply the result of [107] and iteratively remove irrelevant vertices until we arrive at a graph with bounded treewidth. When a part contains large minors, we can prove a generalization of Lemma 5 for disjoint paths logic.

▶ **Lemma 9.** $\mathrm{MC}(\mathsf{FO{+}dp}, \mathscr{C}) \leq_{fpt}$ *$MC(\mathsf{FO}, \mathscr{C})$, where $\mathscr{C}$ is any class of graphs that is (q, k)-unbreakable and contains large clique-minors, where k, q and "large" depends on the input formula.*

The lemma is based on the combinatorics of a variant of the "*generic folio lemma*" proved by Robertson and Seymour in [105], which was used by Grohe et al. [71] in order to show that testing topological minor containment is fixed-parameter tractable. Model checking for disjoint paths logic then proceeds by dynamic programming over the tree decomposition into unbreakable parts, using to compositionality method (Section 7) to combine the solutions of the unbreakable parts into a global solution.

5 The Interpretation Method

The interpretation method is a special case of the reduction method that is fundamental for many model checking algorithms. It is based on encoding structures from a class $\mathscr{C}$ in structures from a class $\mathscr{D}$ such that the structures from $\mathscr{C}$ can be recovered from $\mathscr{D}$ by a logical interpretation. This makes the translation of the formula φ in the reduction particularly elegant and simple. This translation is based on the following interpretation lemma.

▶ **Lemma 10.** *Let I be an $\mathcal{L}$-interpretation of σ-structures in τ-structures. Then for every formula $\varphi \in \mathcal{L}[\sigma]$ there exists a formula $\psi \in \mathcal{L}[\tau]$ with the following property. Let $\mathbb{A}$ be a σ-structure and $\mathbb{B}$ be a τ-structure such that $\mathbb{A} \cong \mathsf{I}(\mathbb{B})$. Then $\mathbb{A} \models \varphi \Leftrightarrow \mathbb{B} \models \psi$. Furthermore, ψ can be efficiently computed from ψ.*

In fact, it is very easy to compute the formula ψ from φ. If $\mathsf{I} = (\nu, (\rho_R)_{R\in\sigma})$, we simply have to replace all occurrences of atoms $R(\bar{x})$ by their defining formulas $\rho_R(\bar{x})$, and restrict quantification to those elements that satisfy the formula ν.

We can now define reductions that are based on interpretations.

▶ **Definition 11.** *A class $\mathscr{C}$ of σ-structures admits* efficient *$\mathcal{L}$-encoding in a class $\mathscr{D}$ of τ-structures if there exists an $\mathcal{L}$-interpretation I of σ-structures in τ-structures and an algorithm that given $\mathbb{A} \in \mathscr{C}$ and $\varphi \in \mathcal{L}$ computes a structure $\mathbb{B} \in \mathscr{D}$ with $\mathsf{I}(\mathbb{B}) \cong \mathbb{A}$ in time$f(|\varphi|) \cdot \|\mathbb{A}\|^{\mathcal{O}(1)}$ for some computable function f.*

The next lemma is immediate by Lemma 10 and is the key to the interpretation method.

▶ **Lemma 12.** *If a class $\mathscr{C}$ of σ-structures admits* efficient *$\mathcal{L}$-encoding in a class $\mathscr{D}$ of τ-structures, then $\mathrm{MC}(\mathcal{L}, \mathscr{C}) \leq_{fpt} \mathrm{MC}(\mathcal{L}, \mathscr{D})$. Hence, if $\mathrm{MC}(\mathcal{L}, \mathscr{D})$ is fixed-parameter tractable, then so is $\mathrm{MC}(\mathcal{L}, \mathscr{C})$.*

The interpretation method combines well with other methods. In many cases the signature of $\mathscr{D}$ is chosen such that the logic we are interested in collapses to a simpler logic, for example, the signature may allow for quantifier elimination, such that model checking becomes trivial. Also the single steps in the game based approaches on nowhere dense and monadically stable classes can be seen as simplifications of the model checking problem via the interpretation method.

5.1 Example: The CMSO-Encoding on Classes with Bounded Cliquewidth in Colored Trees

We recall the classical reduction of CMSO on classes with bounded cliquewidth to CMSO on colored trees. We define clique expressions with respect to more general base classes, as the methods for $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A), \sigma)$ on augmented trees extends to these more general graph classes. We remark however, that while cliquewidth and clique decompositions can be efficiently computed [101], this is not clear for clique decompositions over more general classes and we may have to require such decompositions to be given with the input.

A k-colored graph is a graph with each vertex assigned a color from $[k]$. On k-colored graphs we define the following operations with respcet to a base class $\mathscr{C}$.

- *Create* a k-colored graph G from $\mathscr{C}$.
- For a function $c : [k] \to [k]$ *recolor* the vertices of the input graph according to c.
- For a set S of 2-element subsets of $[k]$, *join* a family of k-colored graphs by taking their disjoint union and for each $\{i, j\} \in S$ (possibly $i = j$), add an edge between every pair of vertices that have colors i and j, respectively, and originate from different input graphs.

A *width-k* clique decomposition is a (rooted) tree T where the nodes are labeled by the above operation names in an arity preserving way, that is, all constants are leaves and all recolor operations have exactly one child. Note that the join operation is a commutative operation, so the tree does not need to have an order on siblings. A clique decomposition defines the k-colored graph obtained by evaluating the operations in the decomposition.

The *cliquewidth* of a graph is the minimum number k for which there is a width-k clique decomposition whose result is (some coloring of) the graph over $\mathscr{C}_1$, which contains only the single-vertex graph.

We remark that this definition of cliquewidth is different from the original definition [25], however, it is within factor 2 of that definition. Also note that every class of graphs with bounded treewidth also has bounded cliquewidth.

▶ **Lemma 13.** *Let $\mathscr{C}$ be a class with bounded cliquewidth. Then there exists a class $\mathscr{T}$ of colored trees such that $\mathscr{C}$ can be MSO-encoded in $\mathscr{T}$. In particular,* $\mathrm{MC}(\mathsf{CMSO}, \mathscr{C}) \leq_{fpt} \mathrm{MC}(\mathsf{CMSO}, \mathscr{T})$.

The idea is to use the tree T from the clique decomposition as the host graph. The vertices of G are the leaves of T. In order to decide whether two vertices u, v are connected by an edge we have to find the color of the vertices at the least common ancestor $x = \mathrm{lca}(u, v)$ and check whether vertices of these colores are connected by the join operation at x. MSO can keep track of how colors change through the tree interpret the connections accordingly.

5.2 Example: Structurally sparse graph classes

With the good understanding of sparse graph classes it is a natural question to extend these results to classes that interpret or transduce in sparse classes. For example, the concepts of treedepth and treewidth as well as their dense analogs shrubdepth and cliquewidth can be defined in terms of transductions. We write $\mathscr{T}_d$ for the class of trees of depth at most d and $\mathscr{T}$ for the class of all trees.

- A class $\mathscr{C}$ of graphs has bounded treedepth if and only if the class of incidence graphs of graphs from $\mathscr{C}$ can be FO- or MSO-transduced from $\mathscr{T}_d$ for some $d \geq 1$ (this follows from the work of Ganian et al. [64, 65]).
- A class $\mathscr{C}$ of graphs has bounded treewidth if and only if the class of incidence graphs of graphs from $\mathscr{C}$ can be MSO-transduced from $\mathscr{T}$ by a classical result of Courcelle [23].
- A class $\mathscr{C}$ of graphs has bounded treewidth if and only if the class of incidence graphs of graphs from $\mathscr{C}$ can be FO-transduced from the class of all (finite) tree orders as shown by Colcombet [20].

The dense analogs are obtained as follows.

- A class $\mathscr{C}$ of graphs has bounded shrubdepth if and only if $\mathscr{C}$ can be FO- or MSO-transduced from $\mathscr{T}_d$ for some $d \geq 1$ as shown by Ganian et al. [64, 65].
- A class $\mathscr{C}$ of graphs has bounded cliquewidth if and only if $\mathscr{C}$ can be MSO-transduced from $\mathscr{T}$ as proved by Courcelle [23].
- A class $\mathscr{C}$ of graphs has bounded cliquewidth if and only if $\mathscr{C}$ can be FO-transduced from the class of all (finite) tree orders as shown by Colcombet [20].

When applied to the classical notions of sparsity we obtain the following notions of *structural sparsity*, see [59, 95]

- A class $\mathscr{D}$ has *structurally bounded degree* if there exists a class $\mathscr{C}$ of bounded degree such that $\mathscr{D}$ can be transduced from $\mathscr{C}$ [57, 58].
- A class $\mathscr{D}$ has *structurally bounded expansion* if there exists a class $\mathscr{C}$ of bounded expansion such that $\mathscr{D}$ can be transduced from $\mathscr{C}$ [59].
- A class $\mathscr{D}$ is *structurally nowhere dense* if there exists a nowhere dense class $\mathscr{C}$ such that $\mathscr{D}$ can be transduced from $\mathscr{C}$ [40, 59].

Structurally sparse classes have a rich combinatorial theory, however, the difficulty to obtain algorithmic meta theorems for these classes lies in the problem to find the sparse pre-images of graphs when given only the dense input graph from $\mathscr{D}$. Only for classes with structurally bounded degree we have a meta theorem whose proof is based on the interpretation method of Lemma 12.

▶ **Theorem 14** ([57]). *Let $\mathscr{D}$ be a class with structurally bounded degree. Then there exists a class $\mathscr{C}$ with bounded degree such that $\mathscr{D}$ can be FO-encoded in $\mathscr{C}$. In particular,* $\mathrm{MC}(\mathsf{FO}, \mathscr{D}) \leq_{fpt} \mathrm{MC}(\mathsf{FO}, \mathscr{C})$*.*

Even though we know how to solve the FO model checking even on monadically stable classes [37], efficient sparsification of classes with structurally bounded expansion and structurally nowhere dense classes (both of these are monadically stable) is still an important open problem.

6 Quantifier elimination

Quantifier elimination is a classical technique from model theory to demonstrate the tameness of theories. In the algorithmic context of finite structures we can first enrich the signature, and then eliminate quantifiers, making quantifier elimination a special case of a reduction. Model checking of quantifier-free formulas is then trivial, as we just have to check the atomic type of an input tuple.

6.1 Example: Quantifier Elimination on Classes with Bounded Expansion

We demonstrate the principle with the by now classical example of quantifier elimination on classes with bounded expansion. The first building block is quantifier elimination on classes with bounded treedepth. To generalize to classes with bounded expansion we need a slightly stronger statement than just the quantifier elimination result. Recall from the preliminaries that a guided pointer structures is a σ-structures, where σ consists of a single binary relation E, unary relations, and unary functions, with the property that E is symmetric and anti-reflexive, and that every function $f \in \sigma$ is *guided*, meaning that if $f(u) = v$, then $u = v$ or $uv \in E(G)$.

▶ **Theorem 15** ([44]). *For every FO-formula $\varphi(\bar{x})$ and every class $\mathscr{C}$ of colored graphs with bounded treedepth there exists a quantifier-free formula $\tilde{\varphi}(\bar{x})$ and a linear time computable map Y such that, for every $G \in \mathscr{C}$, $Y(G)$ is a guided expansion of G such that for all tuples of vertices $\bar{v}$ we have $G \models \varphi(\bar{v}) \Leftrightarrow Y(G) \models \tilde{\varphi}(\bar{v})$.*

This quantifier elimination result lifts to classes with bounded expansion by so-called low treedepth colorings. For a positive integer p, a *p-treedepth coloring* of a graph G is a vertex coloring of G such that the subgraph induced by any $i \leq p$ color classes has treedepth at most i.

▶ **Lemma 16** ([121]). *A class $\mathscr{C}$ of graphs has bounded expansion if and only if for every p there exists $c(p)$ such that every $G \in \mathscr{C}$ admits a p-treedepth coloring with at most $c(p)$ colors. Furthermore, given G and p, such a coloring is efficiently computable.*

Using low treedepth decompositions, one can now iteratively eliminate quantifiers in bounded expansion classes. Note that because the computed expansions are guided by the original graph structure, the class (of Gaifman graphs) does not loose the property of having bounded expansion in each quantifier elimination step. This leads to the following theorem.

▶ **Theorem 17** ([44, 59, 72]). *Let $\mathscr{C}$ be a class with bounded expansion. There exists a class $\mathscr{D}$ of guided pointer structures such that for every $G \in \mathscr{C}$ and FO-formula $\varphi(\bar{x})$ one can efficiently compute a quantifier-free formula $\tilde{\varphi}(\bar{x})$ and a guided expansion $Y(G)$ of G such that for all tuples of vertices $\bar{v}$*

$$G \models \varphi(\bar{v}) \quad \Leftrightarrow \quad Y(G) \models \tilde{\varphi}(\bar{v}).$$

Consequently, $\mathrm{MC}(\mathsf{FO}, \mathscr{C}) \leq_{fpt} \mathrm{MC}(\mathsf{QF}, \mathscr{D})$, and in particular, as $\mathrm{MC}(\mathsf{QF}, \mathscr{D})$ is fixed-parameter tractable, $\mathrm{MC}(\mathsf{FO}, \mathscr{C})$ is fixed-parameter tractable.

Nowhere dense classes can be characterized by low treedepth colorings similarly as bounded expansion classes. A class $\mathscr{C}$ of graphs is nowhere dense if and only if for every p and every $\varepsilon > 0$ there exists $c(p, \varepsilon)$ such that every n-vertex graph $H \subseteq G \in \mathscr{C}$ admits a p-treedepth coloring with at most $c(p, \varepsilon) \cdot n^{\varepsilon}$ colors. Note that these are too many colors to also obtain quantifier elimination on nowhere dense classes. In fact, it was proved that quantifier elimination as for bounded expansion classes cannot exist for nowhere dense classes [69].

Classes with structurally bounded expansion are characterized by the existence of p-shrubdepth colorings with $c(p)$ colors for each fixed p [59] (here we do not require that the shrubdepth is at most p (note that this is not even defined) but only that it is bounded for every p). Monadically stable classes are characterized by the existence of p-shrubdepth colorings with $c(p, \varepsilon) \cdot n^{\varepsilon}$ colors for each p [14].

7 The composition method

The composition method is a very classical method for FO and MSO, so we only give a very rough sketch and refer to the literature [72, 70, 83]. The method allows to deduce the truth of formulas in structures that are composed of simpler parts, e.g. via tree decompositions with small adhesion. The most classical composition theorems goes back to Feferman and Vaught [51]. This key lemma is easily proved using Ehrenfeucht-Fraïssé games (in particular it extends to CMSO).

▶ **Theorem 18** (Feferman and Vaught [51]). *Let $\mathbb{A}, \mathbb{B}$ be σ-structures and let $\bar{c}$ be constant symbols naming all elements of $V(\mathbb{A}) \cap V(\mathbb{B})$. Then the q-type (for FO or MSO) of $\mathbb{A} \cup \mathbb{B}$ is determined by the q-type of $(\mathbb{A}, \bar{c})$ and the q-type of $(\mathbb{B}, \bar{c})$.*

By dynamic programming along tree decompositions one obtains for example efficient CMSO model checking on classes with bounded treewidth, Courcelle's famous theorem.

▶ **Theorem 19** (Courcelle's Theorem [21]). *Let $\mathscr{C}$ be a class of bounded treewidth. Then $\mathrm{MC}(\mathsf{CMSO}, \mathscr{C})$ is fixed-parameter tractable.*

We obtain as an immediate corollary of Theorem 8 that $MC(\mathsf{CMSO/tw{+}dp}, \mathscr{C})$ is fixed-parameter tractable on classes with excluded minors.

▶ **Corollary 20** ([107]). *Let $\mathscr{C}$ be a class excluding some minor. Then $\mathrm{MC}(\mathsf{CMSO/tw{+}dp}, \mathscr{C})$ is fixed-parameter tractable.*

Another recent application of the composition method is the reduction of CMSO model checking to unbreakable graphs by the recursive understanding technique [91].

▶ **Theorem 21** ([91]). *Let φ be a CMSO sentence. For all k there exists q such that if there exists an algorithm that solves the model checking problem for φ on (q,k)-unbreakable graphs in time $\mathcal{O}(n^d)$ for some $d > 4$, then there exists an algorithm that solves the model checking problem for φ on general graphs in time $\mathcal{O}(n^d)$.*

Note that the result is for individual CMSO properties. Efficient model checking for a single sentence φ on unbreakable graphs is a much weaker requirement than the requirement that CMSO types (up to some quantifier rank) have to be efficiently computable on unbreakable graphs. The theorem is proved by recursively replacing large unbreakable parts of the graph that are glued by small separators by small parts that have the same type just with respect to φ. This is possible for each fixed formula φ, as representatives for equivalence classes can be hardcoded in the algorithm. As a consequence, the theorem only yields non-uniform fpt algorithms.

Uniform algorithms via dynamic programming exist for many special cases, in particular model checking for disjoint paths logic on classes with excluded topological minors is fixed-parameter tractable and this result is proved using the composition method.

▶ **Theorem 22** ([108]). *Let $\mathscr{C}$ be a class with excluded topological minors. Then* $\mathrm{MC}(\mathsf{FO}+\mathsf{dp},\mathscr{C})$ *is fixed-parameter tractable.*

8 The automata based method

Also the automata theoretic method is a classical method for model checking. Every CMSO property of colored trees can be translated into an equivalent automaton, which can simply be run on the tree to evaluate [32, 116]. As a consequence, we immediately get the following theorem.

▶ **Theorem 23** ([32, 116]). *Let $\mathscr{T}$ be a class of colored trees. Then* $\mathrm{MC}(\mathit{CMSO},\mathscr{T})$ *is fixed-parameter tractable.*

In order to conclude the fixed parameter tractability of model checking for separator logic on classes with excluded topological minors via the reduction of Theorem 7, we now consider automata for $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A),\sigma)$ that run on augmented trees. Recall that in augmented trees we have a tree T represented by the ancestor relation $\preccurlyeq$ and additionally labeled with letters from an alphabet A, as well as σ-structures on the children of inner nodes of T. In the following we will call the structure below a node x the *whorl* of x. The whorls are used to represent the bags of tree decompositions, for which we want to do first-order model checking (there is a small caveat here that we will comment on below). We define automata that read augmented trees in the standard bottom-up fashion and whose transition function is defined by first-order formulas. At any node x, to determine the state of the automaton at x, we consider the graph induced on the children of x in the augmented tree, vertex-labeled by the states already determined in the bottom-up computation. Then the state at x is determined by evaluating a fixed collection of first-order sentences on this labeled graph. Finally, the automaton accepts the augmented tree depending on the state at the root. As a result we get that $\mathsf{FO}(\mathsf{MSO}(\preccurlyeq, A),\sigma)$ model checking is fixed parameter tractable on trees that are augmented with structures on which FO model checking is fixed-parameter tractable.

Formally, an *automaton $\mathcal{A}$ on (A,σ)-augmented trees* consists of:

- an input *alphabet* A: the automaton will process augmented trees over the alphabet A,
- a finite set of *states* Q,
- a set of *accepting states* $F \subset Q$,

- for each state $q \in Q$ and letter $a \in A$, a first-order sentence $\delta_{q,a}$ in the signature $\Sigma \cup Q$, where each $q \in Q$ is viewed as a unary relation symbol, such that for every fixed $a \in A$, the sentences $(\delta_{q,a})_{q\in Q}$ are mutually inconsistent and their disjunction $\bigvee_{q\in Q} \delta_{q,a}$ is equivalent to true. The sentences $\delta_{q,a}$ are called the *transition sentences* of $\mathcal{A}$.

A *run* of $\mathcal{A}$ on an augmented tree T is a labeling $\rho : V(T) \to Q$ such that for every node v with letter a and whorl $\mathbb{A}_v$, the state $q = \rho(v)$ is the unique state such that $\delta_{q,a}$ holds in the Q-labeled structure $\mathbb{A}_v$ with labeling ρ restricted to $\mathbb{A}_v$. Formally, $\mathbb{A}_v$ is viewed as a structure over the signature $\Sigma \cup Q$, where a predicate $q \in Q$ holds on a vertex v if and only if $\rho(v) = q$. The run is *accepting* if it labels the root with an accepting state. The automaton $\mathcal{A}$ *accepts* an augmented tree T if it has an accepting run on it.

▶ **Theorem 24** ([102]). *For every formula $\varphi(\bar{x})$ of FO(MSO($\preccurlyeq, A$), σ) there is an automaton $\mathcal{A}$ on augmented trees that is equivalent to φ. Moreover, $\mathcal{A}$ is computable from φ.*

We call the model checking problem for classes of structures that are additionally labeled with unary predicates the *labeled model checking problem.*

▶ **Lemma 25** ([102]). *Let $\mathscr{C}$ be a class of σ-structures such that labeled model checking for FO is fixed-parameter tractable on $\mathscr{C}$. Then automata evaluation on $\mathscr{C}$-augmented trees is efficient.*

As a corollary we get the following theorem.

▶ **Theorem 26** ([102]). *Let $\mathscr{C}$ be a class of σ-structures such that labeled model checking for FO is fixed-parameter tractable on $\mathscr{C}$. Then model-checking for FO(MSO($\preccurlyeq, A$), σ) is efficient on $\mathscr{C}$-augmented trees with labels from a finite alphabet A.*

As a final corollary we obtain the following theorem.

▶ **Theorem 27** ([109]). *Let $\mathscr{C}$ be a class excluded a topological minor. Then* MC(*FO+conn*, $\mathscr{C}$) *is fixed-parameter tractable.*

Let us comment on why we do not obtain fixed-parameter tractability on nowhere dense classes. The reason is that we were slightly imprecise when we stated that used the parts of the decomposition into unbreakable parts as the whorls in the augmentations. In fact, we need to toke the so-called *bag graphs*, which are obtained from the subgraphs induced by a bag by turning all adhesions towards children into cliques. If the original graph excludes some topological minor, then the bag graphs also exclude some (larger) topological minor, hence, FO model checking is tractable on the bag graphs. If however, the original graph comes from a nowhere dense class, the bag graphs are possibly no longer nowhere dense.

9 Locality and Game-based Decompositions of Local Neighborhoods

We now sketch the methods for FO model checking on nowhere dense and monadically stable graph classes. As discussed in the introduction, monadically dependent classes are conjectured to be the most general hereditary classes of graphs with tractable FO model checking. Monadically stable classes are important special cases of monadically dependent classes. On monotone classes the notions of monadic dependence, monadic stability and nowhere denseness coincide [1] and for monotone classes nowhere denseness constitutes the limit of tractability [73, 43, 83]. The basic method applied on nowhere dense classes could be generalized to monadically stable classes [40, 37]. We present both results in parallel.

The key property of FO that is exploited for efficient model checking is its *locality.* This is formalized for example by Gaifman's Locality Theorem, which states that every first-order formula $\varphi(\bar{x})$ is equivalent to a Boolean combination of

1. *local formulas* $\psi^{(r)}(\bar{x})$ and
2. *basic local formulas* $\exists x_1 \dots \exists x_k \big(\bigwedge_{i \neq j} \operatorname{dist}(x_i, x_j) > 2r \wedge \chi^{(r)}(x_i) \big)$.

Here, the notation $\psi^{(r)}(\bar{x})$ means that for every graph G and every tuple $\bar{v} \in V(G)^{|\bar{x}|}$ we have $G \models \psi^{(r)}(\bar{v})$ if and only if $G[N_r(\bar{v})] \models \psi^{(r)}(\bar{v})$, where $G[N_r(\bar{v})]$ denotes the subgraph of G induced by the r-neighborhood $N_r(\bar{v})$ of $\bar{v}$. The numbers r and k in the formulas above depend only on the formula φ, and furthermore, the Gaifman normal form of any formula φ is computable from φ.

This translates the model-checking problem to the following algorithmic problem. To decide for a graph G and tuple $\bar{v} \in V(G)^{|\bar{x}|}$ whether $G \models \varphi(\bar{v})$,

1. decide whether $\bar{v}$ has the local properties described by $\psi^{(r)}(\bar{x})$;
2. decide for each $v \in V(G)$ whether $G \models \chi^{(r)}(v)$;
3. solve each *generalized independent set* problem described by the basic local formulas $\exists x_1 \dots \exists x_k \big(\bigwedge_{i \neq j} \operatorname{dist}(x_i, x_j) > 2r \wedge \chi^{(r)}(x_i) \big)$, and finally
4. evaluate the Boolean combination of these statements that is equivalent to φ.

Note that the generalized independent set problem is also a local problem by the following argument. If a graph has many vertices satisfying $\chi^{(r)}$ that are far away from each other, then we can greedily pick elements for the independent set. Otherwise, all elements satisfying $\chi^{(r)}$ are close to one of the greedily picked vertices, so that testing if there is a different solution becomes a local property.

Local formulas can be evaluated in bounded-radius neighborhoods of the graph, where the radius depends only on φ. Hence, whenever the local neighborhoods in graphs from a class $\mathscr{C}$ admit efficient model checking, then one immediately obtains an efficient model checking algorithm for $\mathscr{C}$. This technique was first employed in [56] and is also the basis of the model checking algorithm on nowhere dense graph classes and monadicallly stable classes. It immediately yields efficient model checking e.g. on planar graphs, as they have locally bounded treewidth, or on classes with locally bounded cliquewidth.

However, on nowhere dense or monadically stable classes we cannot immediately apply one of the above presented meta theorems. Instead, we recursively apply simplifications based on the interpretation method in the local neighborhoods. In nowhere dense classes we simplify by deleting a single vertex and in monadically stable classes we simplify by *flipping* the edges between two sets of vertices. Obviously, the original edge set can be recovered by an interpretation in a coloring of the graph in both cases. By an appropriately chosen vertex deletions or flips, we have made the formula to be checked more complicated, but the graph to be tested has become simpler. We then recurse with localizing the formulas, considering local neighborhoods and deleting a vertex or flipping, until finally we arrive at a single vertex graph, on which the model checking problem is trivial. The formal treatment of the recursive simplification of the graph is captured by games, by the *Splitter game* on nowhere dense classes and the *Flipper game* on monadically stable classes.

In the radius-r Splitter game, two players called *Connector* and *Splitter*, engage on a graph and thereby recursively decompose local neighborhoods. Starting with the input graph G, in each of the following rounds, Connector chooses a subgraph of the current game graph of radius at most r and Splitter deletes a single vertex from this graph. The game continues with the resulting graph and terminates when the single-vertex graph is reached. A class of graphs is nowhere dense if and only if for every r there exists ℓ such that Splitter can win the radius-r Splitter game in ℓ rounds [73]. Furthermore, a strategy for splitter can be efficiently computed. Hence, the recursive decomposition can be efficiently computed and terminates after a bounded number of rounds.

Similarly, in the radius-r Flipper game, two players called *Connector* and *Flipper*, engage on a graph and thereby recursively decompose local neighborhoods. Starting with the input graph G, in each of the following rounds, Connector chooses a subgraph of the current game graph of radius at most r and Flipper chooses two sets of vertices A and B and flips the adjacency between the vertices of these two sets. The game continues with the resulting graph. Flipper wins once a graph consisting of a single vertex is reached. A class of graphs is monadically stable if and only if for every r there exists $\ell(r)$ such that Flipper can win the radius-r Flipper game in $\ell(r)$ rounds [60]. Furthermore, a strategy for flipper can be efficiently computed. Again, we get an efficient recursive decomposition of bounded depth.

A straight-forward implementation of this recursive algorithm however leads to a branching degree of n and a running time of $\Omega(n^\ell)$, where the depth ℓ of the recursion grows with φ. To avoid this, [73] cluster nearby neighborhoods and handle them together in one recursive call using so-called *sparse r-neighborhood covers*. An r-neighborhood cover with *degree d* and *radius s* of a graph G is a family $\mathcal{X}$ of subsets of $V(G)$, called *clusters*, such that the r-neighborhood of every vertex is contained in some cluster, every cluster has radius at most s, and every vertex appears in at most d clusters. It was shown in [73] that nowhere dense classes admit r-neighborhood covers with radius $2r$ and degree $\mathcal{O}(n^\varepsilon)$ for any $\varepsilon > 0$. Similarly, it was shown in [37] that monadically stable classes admit r-neighborhood covers with radius $4r$ and degree $\mathcal{O}(n^\varepsilon)$ for any $\varepsilon > 0$. By scaling ε appropriately the recursive data structure can be constructed in the desired fpt running time.

Some care has to be taken that the quantifier rank of the formulas does not grow when localizing in each recursive step. This technical problem was addressed by considering a rank-preserving normal form in [73]. This approach could be much simplified by a nice game based argument in [40]. We obtain the following theorems.

▶ **Theorem 28** ([73]). *Let $\mathscr{C}$ be a nowhere dense class. Then* MC(*FO*, $\mathscr{C}$) *is fixed-parameter tractable.*

▶ **Theorem 29** ([40, 37]). *Let $\mathscr{C}$ be a monadically stable class. Then* MC(*FO*, $\mathscr{C}$) *is fixed-parameter tractable.*

As mentioned in the introduction, it is one of the main questions in the area whether these results can be extended to monadically dependent classes. If based on the same approach as for nowhere dense and monadically stable classes, one would have to devise a game that not necessarily has bounded length, but that, combined with sparse neighborhood coves, if they exist for monadically dependent classes, leads to an fpt bounded size recursive data structure.

10 Twinwidth

Twinwidth was recently introduced by Bonnet, Kim, Thomassé and Watrigant [12] as a generalization of an invariant for classes of permutations defined by Guillemot and Marx [75]. The definition of twinwidth is based on *contraction sequences*.

Let G be a graph on n vertices. A *contraction sequence* for G is a sequence $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of partitions of the vertex set of G such that:

- $\mathcal{P}_1$ is the partition into singletons;
- $\mathcal{P}_n$ is the partition with one part;
- for each $t \in [n]$, $t > 1$, $\mathcal{P}_t$ is obtained from $\mathcal{P}_{t-1}$ by taking some two parts $A, B \in \mathcal{P}_{t-1}$ and *contracting* them: replacing them with a single part $A \cup B \in \mathcal{P}_t$.

A pair of disjoint vertex subsets $A, B \subset V(G)$ is *complete* if every vertex of A is adjacent to every vertex of B, and *anti-complete* if there is no edge with one endpoint in A and the other one in B. The pair A, B is *pure* if it is complete or anti-complete, and *impure* otherwise.

A *trigraph* is a structure with two types of undirected edges, *red* and *black edges*. Given a partition of $\mathcal{P}$, we define the *quotient trigraph* $G/\mathcal{P}$ as the trigraph on vertex set $\mathcal{P}$, where two parts $A, B \in \mathcal{P}$ are connected by a black edge if the pair A, B is complete in G, non-adjacent if the pair is anti-complete, and connected by a red edge if A, B is impure. For a trigraph H, its *impurity graph* is the graph on vertex set H where two vertices $u, v \in V(H)$ are considered adjacent if they are impure towards each other in H, that is, the subgraph on $V(H)$ induced by the red edges of H.

The *width* of the contraction sequence $\mathcal{P}_1, \dots, \mathcal{P}_n$ is the maximum degree in the impurity graphs of $G/\mathcal{P}_t$ over all times $t \in [n]$. The *twin-width* of G is the minimum possible width of a contraction sequence of G.

Many well-studied classes of graphs have bounded twinwidth, e.g. planar graphs, and more generally, any class of graphs excluding a fixed minor, cographs, and more generally, any class of bounded clique-width, and many more. Most imporantly in our context, the property of having bounded twinwidth is preserved unter FO-transductions, hence, classes with bounded twinwidth are monadically dependent. In fact, a class of ordered structures has bounded twinwidth if and only if it is monadically dependent [9]. FO model checking is fixed-parameter tractable on every class of bounded twinwidth assuming a contraction sequence is given together with the input [12]. On ordered structures a contraction sequence can be computed efficiently, leading to efficient model checking on ordered classes of bounded twinwidth. We give a brief sketch of model checking on classes with bounded twinwidth, following the presentation of [61].

The model checking result can be presented using the notion of *local types*. Intuitively, the k-local type of a part refers to all formulas (up to quantifier rank k) that the $f(k)$-neighborhood of the part satisfies. Here, we consider distances in the impurity graphs.

Initially, the first partition of the sequence only contains singletons and no part is impure with another. So the local types (for a fixed value of k) can be computed in constant time for each part.

Then, and since the provided sequence has width at most d, the k-neighborhood of a part only contains d^k other parts. Furthermore, when two parts are merged according to the sequence, this only impacts the k-neighborhood of a constant number of parts. These types can be recomputed efficiently.

Finally, the local type of the last partition, which only contains one part: the entire graph, provides the type of the graph and informs us whether the original desired property is satisfied.

10.1 Around twinwidth

The method described above, gives rise to various questions. Are there other suitable width parameters for the contraction sequence? Do we need the sequence to end on a single part, or could there be other stopping points?

Some of these directions have been considered by Bonnet et al. [10]. First, by changing the notion of width for the contraction sequence, one can capture the notions of rank width and linear rank width by bounding the size of connected components in the impurity graph, or the total number of impure connections.

On the direction of contraction sequences that stop before the single part partition, Bonnet et al. [10] also proved that if the connections (both pure and impure) of a partition results in a graph of bounded degree, or a graph of bounded expansion, one could still derive information on the type of the original graph. Proving fixed parameter tractability of FO model checking on classes of graphs that can contract to a class with bounded degree, and fixed parameter tractability of ∃FO model checking on classes of graphs that can contract to a class with bounded expansion. Where ∃FO is the existential fragment of first order logic, which prohibits the use of universal quantification, as well as negations.

References

1 Hans Adler and Isolde Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics*, 36:322–330, 2014. doi:10.1016/J.EJC.2013.06.048.

2 Albert Atserias. E. grädel, p. kolaitis, l. libkin, m. marx, i. spencer, m. vardi, y. venema and s. weinstein , finite model theory and its applications, springer-verlag (2007). *Comput. Sci. Rev.*, 2(1):55–59, 2008. doi:10.1016/J.COSREV.2008.01.001.

3 David A Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within nc1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

4 Michael Benedikt and H Jerome Keisler. Expressive power of unary counters. *Structures in Logic and Computer Science: A Selection of Essays in Honor of A. Ehrenfeucht*, pages 34–50, 2005.

5 Mikolaj Bojanczyk. Separator logic and star-free expressions for graphs. *arXiv preprint*, 2021. arXiv:2107.13953.

6 Édouard Bonnet, Jan Dreier, Jakub Gajarskỳ, Stephan Kreutzer, Nikolas Mählmann, Pierre Simon, and Szymon Toruńczyk. Model checking on interpretations of classes of bounded local cliquewidth. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, 2022.

7 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. SIAM, 2021. doi:10.1137/1.9781611976465.118.

8 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.

9 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Toruńczyk. Twin-width IV: ordered graphs and matrices. *Journal of the ACM*, 71(3):1–45, 2024.

10 Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, and Stéphan Thomassé. Twin-width VI: the lens of contraction sequences. In *SODA*, pages 1036–1056. SIAM, 2022. doi:10.1137/1.9781611977073.45.

11 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.

12 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable fo model checking. *ACM Journal of the ACM (JACM)*, 69(1):1–46, 2021. doi:10.1145/3486655.

13 Samuel Braunfeld, Jaroslav Nešetřil, Patrice Ossona de Mendez, and Sebastian Siebertz. Decomposition horizons: from graph sparsity to model-theoretic dividing lines. *arXiv preprint*, 2022. arXiv:2209.11229.

14 Samuel Braunfeld, Jaroslav Nešetřil, Patrice Ossona de Mendez, and Sebastian Siebertz. Decomposition horizons and a characterization of stable hereditary classes of graphs. *arXiv preprint*, 2024. arXiv:2209.11229.

15 Ashok Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and system Sciences*, 25(1):99–128, 1982. doi:10.1016/0022-0000(82)90012-5.

16 Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, 1977. doi:10.1145/800105.803397.

17 Jianer Chen, Xiuzhen Huang, Iyad A Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/J.JCSS.2006.04.007.

18 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing fpt algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016. doi:10.1137/15M1032077.

19 Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. doi:10.1145/362384.362685.

20 Thomas Colcombet. A combinatorial theorem for trees: applications to monadic logic and infinite structures. In *Automata, Languages and Programming: 34th International Colloquium, ICALP 2007, Wrocław, Poland, July 9-13, 2007. Proceedings 34*, pages 901–912. Springer, 2007.

21 Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Formal models and semantics*, pages 193–242. Elsevier, 1990. doi:10.1016/B978-0-444-88074-1.50010-X.

22 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.

23 Bruno Courcelle. The monadic second-order logic of graphs VII: Graphs as relational structures. *Theoretical Computer Science*, 101(1):3–33, 1992. doi:10.1016/0304-3975(92)90148-9.

24 Bruno Courcelle, Johann A Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/S002249910009.

25 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

26 Bruno Courcelle and Sang-il Oum. Vertex-minors, monadic second-order logic, and a conjecture by seese. *Journal of Combinatorial Theory, Series B*, 97(1):91–126, 2007. doi:10.1016/J.JCTB.2006.04.003.

27 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5(4). Springer, 2015. doi:10.1007/978-3-319-21275-3.

28 Marek Cygan, Paweł Komosa, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Saket Saurabh, and Magnus Wahlström. Randomized contractions meet lean decompositions. *ACM Transactions on Algorithms (TALG)*, 17(1):1–30, 2020. doi:10.1145/3426738.

29 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 323–332, 2014. doi:10.1145/2591796.2591852.

30 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019. doi:10.1137/140988553.

31 Anuj Dawar, Martin Grohe, and Stephan Kreutzer. Locally excluding a minor. In *LICS 2007*, pages 270–279. IEEE, 2007. doi:10.1109/LICS.2007.31.

32 John Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, 1970. doi:10.1016/S0022-0000(70)80041-1.

33 Rod G Downey and Michael R Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on computing*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

34 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999. `doi:10.1007/978-1-4612-0515-9`.

35 Rodney G Downey, Michael R Fellows, et al. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

36 Jan Dreier. Lacon-and shrub-decompositions: A new characterization of first-order transductions of bounded expansion classes. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021. `doi:10.1109/LICS52264.2021.9470680`.

37 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. `arXiv:2311.18740`.

38 Jan Dreier, Jakub Gajarskỳ, Sandra Kiefer, Michał Pilipczuk, and Szymon Toruńczyk. Treelike decompositions for transductions of sparse graphs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–14, 2022.

39 Jan Dreier, Nikolas Mählmann, Amer E. Mouawad, Sebastian Siebertz, and Alexandre Vigny. Combinatorial and algorithmic aspects of monadic stability. In *33rd International Symposium on Algorithms and Computation, ISAAC 2022*, volume 248 of *LIPIcs*, pages 11:1–11:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPICS.ISAAC.2022.11`.

40 Jan Dreier, Nikolas Mählmann, and Sebastian Siebertz. First-order model checking on structurally sparse graph classes. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 567–580, 2023. `doi:10.1145/3564246.3585186`.

41 Jan Dreier, Nikolas Mählmann, Sebastian Siebertz, and Szymon Toruńczyk. Indiscernibles and flatness in monadically stable and monadically nip classes. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

42 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. `doi:10.1145/3618260.3649739`.

43 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Deciding first-order properties for sparse graphs. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 133–142. IEEE, 2010.

44 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM (JACM)*, 60(5):36, 2013.

45 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory.* Perspectives in Mathematical Logic. Springer, 1995.

46 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical logic (2. ed.).* Undergraduate texts in mathematics. Springer, 1994.

47 Kord Eickmeyer, Jan van den Heuvel, Ken-ichi Kawarabayashi, Stephan Kreutzer, Patrice Ossona De Mendez, Michał Pilipczuk, Daniel A Quiroz, Roman Rabinovich, and Sebastian Siebertz. Model-checking on ordered structures. *ACM Transactions on Computational Logic (TOCL)*, 21(2):1–28, 2020. `doi:10.1145/3360011`.

48 Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, 1997. `doi:10.1006/JCSS.1997.1485`.

49 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of computation*, 7:43–73, 1974.

50 Ronald Fagin. Monadic generalized spectra. *Math. Log. Q.*, 21(1):89–96, 1975. `doi:10.1002/MALQ.19750210112`.

51 Solomon Feferman and Robert L Vaught. The first order properties of products of algebraic systems. *Journal of Symbolic Logic*, 32(2), 1967.

52 Jörg Flum and Martin Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM Journal on Computing*, 31(1):113–145, 2001. `doi:10.1137/S0097539799360768`.

53 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. `doi:10.1007/3-540-29953-X`.

54 Fedor V Fomin, Petr A Golovach, Ignasi Sau, Giannos Stamoulis, and Dimitrios M Thilikos. Compound logics for modification problems. *ACM Transactions on Computational Logic*, 2023.

55 Fedor V Fomin, Petr A Golovach, Giannos Stamoulis, and Dimitrios M Thilikos. An algorithmic meta-theorem for graph modification to planarity and fol. *ACM Transactions on Computation Theory*, 14(3-4):1–29, 2023. doi:10.1145/3571278.

56 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM (JACM)*, 48(6):1184–1206, 2001. doi:10.1145/504794.504798.

57 Jakub Gajarskỳ, Petr Hliněnỳ, Jan Obdržálek, Daniel Lokshtanov, and M Sridharan Ramanujan. A new perspective on fo model checking of dense graph classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–23, 2020. doi:10.1145/3383206.

58 Jakub Gajarsky and Daniel Král'. Recovering sparse graphs. *Leibniz International Proceedings in Informatics (LIPIcs)*, 117:29, 2018.

59 Jakub Gajarskỳ, Stephan Kreutzer, Jaroslav Nešetřil, Patrice Ossona De Mendez, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–41, 2020. doi:10.1145/3382093.

60 Jakub Gajarský, Nikolas Mählmann, Rose McCarty, Pierre Ohlmann, Michal Pilipczuk, Wojciech Przybyszewski, Sebastian Siebertz, Marek Sokolowski, and Szymon Torunczyk. Flipper games for monadically stable graph classes. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023*, volume 261 of *LIPIcs*, pages 128:1–128:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.128.

61 Jakub Gajarský, Michal Pilipczuk, Wojciech Przybyszewski, and Szymon Torunczyk. Twin-width and types. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 123:1–123:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.123.

62 Jakub Gajarskỳ, 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*, pages 1–12, 2022.

63 Robert Ganian, Petr Hliněnỳ, Alexander Langer, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Lower bounds on the complexity of mso1 model-checking. *Journal of Computer and System Sciences*, 80(1):180–194, 2014. doi:10.1016/J.JCSS.2013.07.005.

64 Robert Ganian, Petr Hliněnỳ, Jaroslav Nešetřil, Jan Obdržálek, and Patrice Ossona De Mendez. Shrub-depth: Capturing height of dense graphs. *Logical Methods in Computer Science*, 15, 2019.

65 Robert Ganian, Petr Hliněnỳ, Jaroslav Nešetřil, Jan Obdržálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast mso 1. In *Mathematical Foundations of Computer Science 2012: 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings 37*, pages 419–430. Springer, 2012. doi:10.1007/978-3-642-32589-2_38.

66 Tobias Ganzow and Sasha Rubin. Order-invariant MSO is stronger than counting MSO in the finite. In Susanne Albers and Pascal Weil, editors, *25th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2008*, volume 1 of *LIPIcs*, pages 313–324. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. doi:10.4230/LIPICS.STACS.2008.1353.

67 Petr A Golovach, Giannos Stamoulis, and Dimitrios M Thilikos. Model-checking for first-order logic with disjoint paths predicates in proper minor-closed graph classes. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3684–3699. SIAM, 2023. doi:10.1137/1.9781611977554.CH141.

68 Erich Grädel, Phokion G Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y Vardi, Yde Venema, Scott Weinstein, et al. *Finite Model Theory and its applications.* Springer, 2007.

69 Mario Grobler, Yiting Jiang, Patrice Ossona de Mendez, Sebastian Siebertz, and Alexandre Vigny. Discrepancy and sparsity. *J. Comb. Theory B*, 169:96–133, 2024. doi:10.1016/J.JCTB.2024.06.001.

70 Martin Grohe. Logic, graphs, and algorithms. *Logic and automata*, 2:357–422, 2008.

71 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 479–488, 2011. doi:10.1145/1993636.1993700.

72 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. *AMS-ASL Joint Special Session*, 558:181–206, 2009.

73 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.

74 Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 253–266, 2018. doi:10.1145/3196959.3196970.

75 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

76 Yuri Gurevich. Logic and the challenge of computer science. Technical report, University of Michigan, 1985.

77 Wilfrid Hodges. *Model theory*, volume 42 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1993.

78 Neil Immerman. Upper and lower bounds for first order expressibility. *Journal of Computer and System Sciences*, 25(1):76–98, 1982. doi:10.1016/0022-0000(82)90011-3.

79 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987. doi:10.1137/0216051.

80 Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 1998.

81 Yiting Jiang, Jaroslav Nešetřil, Patrice Ossona de Mendez, and Sebastian Siebertz. Regular partitions of gentle graphs. *Acta Mathematica Hungarica*, 161(2):719–755, 2020.

82 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k-way cut of bounded size is fixed-parameter tractable. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 160–169. IEEE, 2011. doi:10.1109/FOCS.2011.53.

83 Stephan Kreutzer. Algorithmic meta-theorems. *Finite and algorithmic model theory*, 379:177–270, 2011.

84 Stephan Kreutzer. On the parameterized intractability of monadic second-order logic. *Logical Methods in Computer Science*, 8, 2012. doi:10.2168/LMCS-8(1:27)2012.

85 Stephan Kreutzer and Siamak Tazari. Lower bounds for the complexity of monadic second-order logic. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 189–198. IEEE, 2010. doi:10.1109/LICS.2010.39.

86 Stephan Kreutzer and Siamak Tazari. On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 354–364. SIAM, 2010. doi:10.1137/1.9781611973075.30.

87 Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005133.

88 Dietrich Kuske and Nicole Schweikardt. Gaifman normal forms for counting extensions of first-order logic. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

89 Leonid Libkin. *Elements of finite model theory*, volume 41. Springer, 2004. doi:10.1007/978-3-662-07003-1.

90 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

91 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.135.

92 Nikolas Mählmann. *Monadically stable and monadically dependent graph classes: characterizations and algorithmic meta-theorems*. PhD thesis, Universität Bremen, 2024.

93 Johann A Makowsky and JP Marino. Tree-width and the monadic quantifier hierarchy. *Theoretical computer science*, 303(1):157–170, 2003. doi:10.1016/S0304-3975(02)00449-8.

94 Maryanthe Malliaris and Saharon Shelah. Regularity lemmas for stable graphs. *Transactions of the American Mathematical Society*, 366(3):1551–1585, 2014.

95 Jaroslav Nešetřil and P Ossona de Mendez. Structural sparsity. *Russian Mathematical Surveys*, 71(1):79, 2016.

96 Jaroslav Nešetřil and Patrice Ossona De Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011. doi:10.1016/J.EJC.2011.01.006.

97 Jaroslav Nešetřil, Patrice Ossona de Mendez, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Rankwidth meets stability. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2014–2033. SIAM, 2021.

98 Jaroslav Nešetřil, Roman Rabinovich, Patrice Ossona de Mendez, and Sebastian Siebertz. Linear rankwidth meets stability. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1180–1199. SIAM, 2020.

99 Pierre Ohlmann, Michał Pilipczul, Szymon Toruńczyk, and Wojciech Przybyszewski. Canonical decompositions in monadically stable and bounded shrubdepth graph classes. *arXiv preprint*, 2023. arXiv:2303.01473.

100 Open problems from the workshop on algorithms, logic and structure, university of warwick, 2016. URL: https://warwick.ac.uk/fac/sci/maths/people/staff/daniel_kral/alglogstr/openproblems.pdf.

101 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. doi:10.1016/J.JCTB.2005.10.006.

102 Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. Algorithms and data structures for first-order logic with connectivity under vertex failures. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 102:1–102:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.102.

103 Neil Robertson and Paul D Seymour. Graph minors I – XXIII, 1983 – 2010.

104 Neil Robertson and Paul 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.

105 Neil Robertson and Paul 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.

106 Ignasi Sau, Giannos Stamoulis, and Dimitrios M Thilikos. A more accurate view of the flat wall theorem. *Journal of Graph Theory*, 107(2):263–297, 2024.

107 Ignasi Sau, Giannos Stamoulis, and Dimitrios M Thilikos. Parameterizing the quantification of cmso: model checking on minor-closed graph classes. *arXiv preprint arXiv:2406.18465*, 2024. doi:10.48550/arXiv.2406.18465.

108 Nicole Schirrmacher, Sebastian Siebertz, Giannos Stamoulis, Dimitrios M Thilikos, and Alexandre Vigny. Model checking disjoint-paths logic on topological-minor-free graph classes. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12, 2024. doi:10.1145/3661814.3662089.

109 Nicole Schirrmacher, Sebastian Siebertz, and Alexandre Vigny. First-order logic with connectivity operators. *ACM Transactions on Computational Logic*, 24(4):1–23, 2023. doi:10.1145/3595922.

110 Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Transactions on Computational Logic (TOCL)*, 6(3):634–671, 2005. doi:10.1145/1071596.1071602.

111 Detlef Seese. The structure of the models of decidable monadic theories of graphs. *Annals of pure and applied logic*, 53(2):169–195, 1991. doi:10.1016/0168-0072(91)90054-P.

112 Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996. doi:10.1017/S0960129500070079.

113 Saharon Shelah. *Classification theory: and the number of non-isomorphic models*. Elsevier, 1990.

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

115 Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology, 1974.

116 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical systems theory*, 2(1):57–81, 1968. doi:10.1007/BF01691346.

117 Dimitrios M Thilikos and Sebastian Wiederrecht. Excluding surfaces as minors in graphs. *arXiv preprint*, 2023. arXiv:2306.01724.

118 Szymon Toruńczyk. Aggregate queries on sparse databases. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 427–443, 2020.

119 Szymon Toruńczyk. Flip-width: Cops and robber on dense graphs. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 663–700. IEEE, 2023.

120 Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146, 1982.

121 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Mathematics*, 309(18):5562–5568, 2009. doi:10.1016/J.DISC.2008.03.024.

Execution-Time Opacity Problems in One-Clock Parametric Timed Automata

Étienne André
Université Sorbonne Paris Nord, LIPN, CNRS UMR 7030, F-93430 Villetaneuse, France
Institut Universitaire de France (IUF), Paris, France

Johan Arcile
IBISC, Univ Evry, Université Paris-Saclay, 91025 Evry, France

Engel Lefaucheux
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

Parametric timed automata (PTAs) extend the concept of timed automata, by allowing timing delays not only specified by concrete values but also by parameters, allowing the analysis of systems with uncertainty regarding timing behaviors. The full execution-time opacity is defined as the problem in which an attacker must never be able to deduce whether some private location was visited, by only observing the execution time. The problem of full ET-opacity emptiness (i.e., the emptiness over the parameter valuations for which full execution-time opacity is satisfied) is known to be undecidable for general PTAs. We therefore focus here on one-clock PTAs with integer-valued parameters over dense time. We show that the full ET-opacity emptiness is undecidable for a sufficiently large number of parameters, but is decidable for a single parameter, and exact synthesis can be effectively achieved. Our proofs rely on a novel construction as well as on variants of Presburger arithmetics. We finally prove an additional decidability result on an existential variant of execution-time opacity.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Security and privacy → Logic and verification

Keywords and phrases Timed opacity, Parametric timed automata, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.3

Related Version *Full Version*: `https://arxiv.org/abs/2410.01659`

Funding This work is partially supported by the ANR-NRF French-Singaporean research program ProMiS (ANR-19-CE25-0015 / 2019 ANR NRF 0092) and by ANR BisoUS (ANR-22-CE48-0012).

1 Introduction

As surveyed in [13], for some systems, private information may be deduced simply by observation of public information. For example, it may be possible to infer the content of some memory space from the access times of a cryptographic module.

The notion of *opacity* [28, 15] concerns information leaks from a system to an attacker; that is, it expresses the power of the attacker to deduce some secret information based on some publicly observable behaviors. If an attacker observing a subset of the actions cannot deduce whether a given sequence of actions has been performed, then the system is opaque. Time particularly influences the deductive capabilities of the attacker. It has been shown in [19] that it is possible for models that are opaque when timing constraints are omitted, to be non-opaque when those constraints are added to the models.

For this reason, the notion is extended to *timed* opacity in [17], where the attacker can also observe time. The input model is timed automata (TAs) [1], a formalism extending finite-state automata with real-time variables called *clocks*. It is proved in [17] that this version of timed opacity is undecidable for TAs.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 3; pp. 3:1–3:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In [9], a less powerful version of opacity is proposed, where the attacker has access only to the system execution time and aims at deducing whether a private location was visited during the system execution. This version of timed opacity is called *execution-time opacity (ET-opacity)*. Two main problems are considered in [9]: 1) the existence of at least one execution time for which the system is ET-opaque (∃-*ET-opacity*), and 2) whether *all* execution times are such that the system is ET-opaque (called *full ET-opacity*). These two notions of opacity are proved to be decidable for TAs [7]. In the same works, the authors then extend ET-opacity to parametric timed automata (PTAs) [2]. PTAs are an extension of TAs where timed constraints can be expressed with timing parameters instead of integer constants, allowing to model uncertainty or lack of knowledge. The two problems come with two flavors: 1) *emptiness* problems: whether the set of parameter valuations guaranteeing a given version of opacity (∃-ET-opacity or full ET-opacity) is empty or not, and 2) *synthesis* problems: synthesize all parameter valuations for which a given version of opacity holds. Both emptiness problems ∃OE (∃-ET-opacity emptiness) and FOE (full-ET-opacity emptiness) have been shown to be undecidable for PTAs, while decidable subclasses are exhibited [9, 7]. A semi-algorithm (i.e., that may not terminate, but is correct if it does) is provided to solve ∃-ET-opacity synthesis (hereafter ∃OS) in [9].

1.1 Contributions

We address here full-ET-opacity emptiness (FOE) and synthesis (FOS), and ∃-ET-opacity emptiness (∃OE) and synthesis (∃OS), for PTAs with integer-valued parameters over dense time with the following main theoretical contributions:

1. We prove that **FOE is undecidable** (Corollary 27) for PTAs with a single clock and a sufficiently large number of parameters.
2. We prove in contrast that **FOE is decidable** (Corollary 28) for PTAs with a single clock and a single parameter.
3. We prove that **∃OE is decidable** (Theorem 29) for PTAs with a single clock and arbitrarily many parameters. We also exhibit a better complexity for a single parameter over discrete time (Theorem 31).

We focus on one-clock PTAs, as virtually all problems are undecidable for 3 clocks [5], and the 2-clock case is an extremely difficult problem, already for reachability [21]. Our contributions are summarized in Table 1. In order to prove these results, we improve on the semi-algorithm from [9] for ∃OS and provide one for FOS. These solutions are based on the novel notion of *parametric execution times* (PET). The PET of a PTA is the total elapsed time and associated parameter valuations on all paths between two given locations. We provide a semi-algorithm for the computation of PET, that builds upon reachability synthesis (i.e., the synthesis of parameter valuations for which a set of locations are reachable) for which a semi-algorithm already exists [24]. We then show how to resolve ∃OS and FOS problems by performing set operations on PET of two complementary subsets of the PTA where we respectively consider only private paths and only non-private paths.

We then solve the full ET-opacity emptiness (FOE) problem for PTAs with 1 clock and 1 parameter, by rewriting the problems in a parametric variant of Presburger arithmetic. This is done by 1) providing a sound and complete method for encoding infinite PET for PTAs with 1 clock and arbitrarily many parameters over dense time; and 2) translating them into parametric semi-linear sets, a formalism defined and studied in [27]. With these ingredients, we notably prove that: 1) FOE is undecidable in general for PTAs with 1 clock and sufficiently many parameters. This is done by reducing a known undecidable problem of parametric Presburger arithmetic (whose undecidability comes from Hilbert's 10th problem) to the FOE problem in this context. 2) ∃OE is decidable for PTAs with 1 clock and arbitrarily many parameters. This is done by reducing ∃OE to the existential fragment of Presburger arithmetic with divisibility, known to be decidable.

1.2 Related works

The undecidability of timed opacity proved in [17] leaves hope for decidability only by modifying the problem (as in [9, 7]), or by restraining the model. In [31, 32], (initial state) opacity is shown to be decidable on a restricted subclass of TAs called real-time automata [18]. In [3], a notion of *timed bounded opacity*, where the secret has an expiration date, and over a time-bounded framework, is proved decidable. Opacity over subclasses of TAs (such as one-clock or one-actions TAs) is considered in [6, 4] and over discrete time in [25].

In [9], ∃-ET-opacity synthesis (∃OS) is solved using a semi-algorithm. The method is based on a self-composition of the PTA with m parameters and n clocks, where the resulting model consists of $m+1$ parameters and $2n+1$ clocks. The method terminates if the symbolic state space of this self-composition is finite. Our work proposes in contrast an approach based on set operations on parametric execution times (PET) of both complementary subsets of the PTA where we respectively consider only private paths and only non-private paths. Those submodels are each composed of $m+1$ parameters and $n+1$ clocks. Our new method terminates if the symbolic state spaces of both submodels are finite. Another improvement is that the method described here also supports full timed opacity synthesis (FOS).

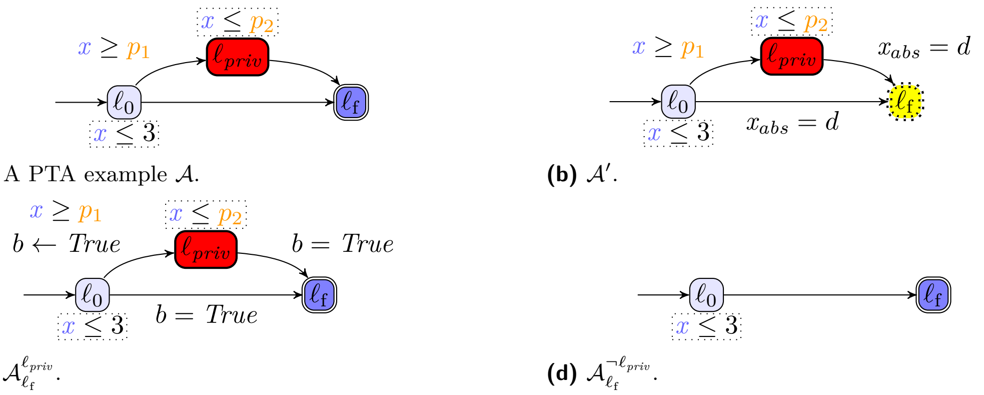
The reachability emptiness problem (i.e., the emptiness over the valuations set for which a given target location is reachable) is known to be undecidable in general since [2]. The rare decidable settings require a look at the number of parametric clocks (i.e., compared at least once in a guard or invariant to a parameter), non-parametric clocks and parameters; throughout this paper, we denote these 3 numbers using a triple (pc, npc, p). Reachability emptiness is decidable for $(1, *, *)$-PTAs ("$*$" denotes "arbitrarily many" for decidable cases, and "sufficiently many" for undecidable cases) over discrete time [2] or dense time with integer-valued parameters [12], for $(1, 0, *)$-PTAs over dense time over rational-valued parameters [10], and for $(2, *, 1)$-PTAs over discrete time [16, 21]; and it is undecidable for $(3, *, 1)$-PTAs over discrete or dense time [12], and for $(1, 3, 1)$-PTAs over dense time only for rational-valued parameters [29]. See [5] for a complete survey as of 2019.

Section 2 recalls the necessary preliminaries. Section 3 introduces one of our main technical proof ingredients, i.e., the definition of PET, and PET-based semi-algorithms for ∃OS and FOS. Section 4 considers the FOE problem over $(1, 0, *)$-PTAs (undecidable) and $(1, 0, 1)$-PTAs (decidable). Section 5 proves decidability of ∃OE for $(1, 0, *)$-PTAs. We also give a better complexity for $(1, 0, 1)$-PTAs over discrete time. Section 6 concludes.

2 Preliminaries

We let $\mathbb{T}$ be the domain of the time, which will be either non-negative reals $\mathbb{R}_{\geq 0}$ (continuous-time semantics) or naturals $\mathbb{N}$ (discrete-time semantics). Unless otherwise specified, we assume $\mathbb{T} = \mathbb{R}_{\geq 0}$.

Clocks are real-valued variables that all evolve over time at the same rate. We assume a set $\mathbb{X} = \{x_1, \dots, x_H\}$ of *clocks*. A *clock valuation* is a function $\mu : \mathbb{X} \to \mathbb{T}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given a constant $\gamma \in \mathbb{T}$, $\mu + \gamma$ denotes the valuation s.t. $(\mu + \gamma)(x) = \mu(x) + \gamma$, for all $x \in \mathbb{X}$. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation μ, denoted by $[\mu]_R$, as follows: $[\mu]_R(x) = 0$ if $x \in R$, and $[\mu]_R(x) = \mu(x)$ otherwise.



(a) A PTA example $\mathcal{A}$.

(b) $\mathcal{A}'$.

(c) $\mathcal{A}^{\ell_{priv}}_{\ell_f}$.

(d) $\mathcal{A}^{\neg\ell_{priv}}_{\ell_f}$.

Figure 1 A PTA example and its transformed versions. The yellow dotted location is urgent.

A *(timing) parameter* is an unknown integer-valued constant of a model. We assume a set $\mathbb{P} = \{p_1, \dots, p_M\}$ of *parameters*. A *parameter valuation* v is a function $v : \mathbb{P} \to \mathbb{N}$.

We assume $\bowtie \in \{<, \leq, =, \geq, >\}$. A *clock guard* C is a conjunction of inequalities over $\mathbb{X} \cup \mathbb{P}$ of the form $x \bowtie \sum_{1 \leq i \leq M} \alpha_i p_i + \gamma$, with $x \in \mathbb{X}$, $p_i \in \mathbb{P}$, and $\alpha_i, \gamma \in \mathbb{Z}$. Given C, we write $\mu \models v(C)$ if the expression obtained by replacing each x with $\mu(x)$ and each p with $v(p)$ in C evaluates to true.

2.1 Parametric timed automata

Parametric timed automata (PTAs) extend TAs with parameters within guards and invariants in place of integer constants [2]. We also add to the standard definition of PTAs a special private location, which will be used to define our subsequent opacity concepts.

▶ **Definition 1** (PTA [2]). *A parametric timed automaton (PTA) [2] $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$, where: 1) Σ is a finite set of actions; 2) L is a finite set of locations; 3) $\ell_0 \in L$ is the initial location; 4) $\ell_{priv} \in L$ is a special private location; 5) $\ell_f \in L$ is the final location; 6) $\mathbb{X}$ is a finite set of clocks; 7) $\mathbb{P}$ is a finite set of parameters; 8) I is the invariant, assigning to every $\ell \in L$ a clock guard $I(\ell)$ (called* invariant*); 9) E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathbb{X}$ is a set of clocks to be reset, and g is a clock guard.*

Given a parameter valuation v, we denote by $v(\mathcal{A})$ the non-parametric structure where all occurrences of a parameter p_i have been replaced by $v(p_i)$.

▶ **Definition 2** (Reset-free PTA). *A* reset-free PTA *$\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$ is a PTA where $\forall\ (\ell, g, a, R, \ell') \in E,\ R = \emptyset$.*

▶ **Example 3.** Consider the PTA $\mathcal{A}$ in Figure 1a. It has three locations, one clock and two parameters (actions are omitted). "$x \leq p_2$" is the invariant of ℓ_{priv}, and the transition from ℓ_0 to ℓ_{priv} has guard "$x \geq p_1$". In this example, x is never reset, and therefore $\mathcal{A}$ happens to be reset-free.

▶ **Definition 4** (Semantics of a timed automaton (TA) [1]). *Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_f, \mathbb{X}, \mathbb{P}, I, E)$ and a parameter valuation v, the semantics of the TA $v(\mathcal{A})$ is given by the timed transition system (TTS) [22] $\mathfrak{T}_{v(\mathcal{A})} = (\mathfrak{S}, \mathfrak{s}_0, \Sigma \cup \mathbb{R}_{\geq 0}, \to)$, with*

1. $\mathfrak{S} = \{(\ell, \mu) \in L \times \mathbb{R}^H_{\geq 0} \mid \mu \models v(I(\ell))\}$, $\mathfrak{s}_0 = (\ell_0, \vec{0})$,
2. $\to$ *consists of the discrete and (continuous) delay transition relations:*

a. *discrete transitions:* $(\ell, \mu) \overset{e}{\mapsto} (\ell', \mu')$, *if* $(\ell, \mu), (\ell', \mu') \in \mathfrak{S}$, *and there exists* $e = (\ell, g, a, R, \ell') \in E$, *such that* $\mu' = [\mu]_R$, *and* $\mu \models v(g)$.

b. *delay transitions:* $(\ell, \mu) \overset{\gamma}{\mapsto} (\ell, \mu + \gamma)$, *with* $\gamma \in \mathbb{R}_{\geq 0}$, *if* $\forall \gamma' \in [0, \gamma], (\ell, \mu + \gamma') \in \mathfrak{S}$.

Moreover we write $(\ell, \mu) \xrightarrow{(\gamma, e)} (\ell', \mu')$ for a combination of a delay and discrete transition if $\exists \mu'' : (\ell, \mu) \overset{\gamma}{\mapsto} (\ell, \mu'') \overset{e}{\mapsto} (\ell', \mu')$.

Given a TA $v(\mathcal{A})$ with concrete semantics $(\mathfrak{S}, \mathfrak{s}_0, \Sigma \cup \mathbb{R}_{\geq 0}, \rightarrow)$, we refer to the states of $\mathfrak{S}$ as the *concrete states* of $v(\mathcal{A})$. A *run* of $v(\mathcal{A})$ is an alternating sequence of concrete states of $v(\mathcal{A})$ and pairs of edges and delays starting from the initial state $\mathfrak{s}_0$ of the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots$ with $i = 0, 1, \dots$, $e_i \in E$, $d_i \in \mathbb{R}_{\geq 0}$ and $(\ell_\mathrm{i}, \mu_i) \xrightarrow{(d_i, e_i)} (\ell_{\mathrm{i}+1}, \mu_{i+1})$.

Given a state $\mathfrak{s} = (\ell, \mu)$, we say that $\mathfrak{s}$ is *reachable* in $v(\mathcal{A})$ if $\mathfrak{s}$ appears in a run of $v(\mathcal{A})$. By extension, we say that ℓ is reachable in $v(\mathcal{A})$; and by extension again, given a set L_{target} of locations, we say that L_{target} is reachable in $v(\mathcal{A})$ if there exists $\ell \in L_{target}$ such that ℓ is reachable in $v(\mathcal{A})$.

Given a finite run $\rho : (\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (d_{i-1}, e_{i-1}), (\ell_\mathrm{n}, \mu_n)$, the *duration* of ρ is $dur(\rho) = \sum_{0 \leq i \leq n-1} d_i$. We also say that ℓ_n is reachable in time $dur(\rho)$.

Let us now recall the symbolic semantics of PTAs (see e.g., [23]). We first define operations on constraints. A *linear term* over $\mathbb{X} \cup \mathbb{P}$ is of the form $\sum_{1 \leq i \leq H} \alpha_i x_i + \sum_{1 \leq j \leq M} \beta_j p_j + \gamma$, with $x_i \in \mathbb{X}$, $p_j \in \mathbb{P}$, and $\alpha_i, \beta_j, \gamma \in \mathbb{Z}$. A *constraint* $\mathbf{C}$ (i.e., a convex polyhedron) over $\mathbb{X} \cup \mathbb{P}$ is a conjunction of inequalities of the form $lt \bowtie 0$, where lt is a linear term. Given a parameter valuation v, $v(\mathbf{C})$ denotes the constraint over $\mathbb{X}$ obtained by replacing each parameter p in $\mathbf{C}$ with $v(p)$. Likewise, given a clock valuation μ, $\mu(v(\mathbf{C}))$ denotes the expression obtained by replacing each clock x in $v(\mathbf{C})$ with $\mu(x)$. We write $\mu \models v(\mathbf{C})$ whenever $\mu(v(\mathbf{C}))$ evaluates to true. We say that v *satisfies* $\mathbf{C}$, denoted by $v \models \mathbf{C}$, if the set of clock valuations satisfying $v(\mathbf{C})$ is nonempty. We say that $\mathbf{C}$ is *satisfiable* if $\exists \mu, v$ s.t. $\mu \models v(\mathbf{C})$. We define the *time elapsing* of $\mathbf{C}$, denoted by $\mathbf{C}^{\nearrow}$, as the constraint over $\mathbb{X}$ and $\mathbb{P}$ obtained from $\mathbf{C}$ by delaying all clocks by an arbitrary amount of time. That is, $\mu' \models v(\mathbf{C}^{\nearrow})$ if $\exists \mu : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0}, \exists \gamma \in \mathbb{R}_{\geq 0}$ s.t. $\mu \models v(\mathbf{C}) \wedge \mu' = \mu + \gamma$. Given $R \subseteq \mathbb{X}$, we define the *reset* of $\mathbf{C}$, denoted by $[\mathbf{C}]_R$, as the constraint obtained from $\mathbf{C}$ by resetting the clocks in R to 0, and keeping the other clocks unchanged. That is,

$$\mu' \models v([\mathbf{C}]_R) \text{ if } \exists \mu : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0} \text{ s.t. } \mu \models v(\mathbf{C}) \wedge \forall x \in \mathbb{X} \begin{cases} \mu'(x) = 0 & \text{if } x \in R \\ \mu'(x) = \mu(x) & \text{otherwise.} \end{cases}$$

We denote by $\mathbf{C}{\downarrow_{\mathbb{P}}}$ the projection of $\mathbf{C}$ onto $\mathbb{P}$, i.e., obtained by eliminating the variables not in $\mathbb{P}$ (e.g., using Fourier-Motzkin [30]).

▶ **Definition 5** (Symbolic state). *A symbolic state is a pair $(\ell, \mathbf{C})$ where $\ell \in L$ is a location, and $\mathbf{C}$ its associated parametric zone.*

▶ **Definition 6** (Symbolic semantics). *Given a PTA $\mathcal{A} = (\Sigma, L, \ell_0, \ell_{priv}, \ell_\mathrm{f}, \mathbb{X}, \mathbb{P}, I, E)$, the symbolic semantics of $\mathcal{A}$ is the labeled transition system called* parametric zone graph $\mathbf{PZG}(\mathcal{A}) = (E, \mathbf{S}, \mathbf{s}_0, \Rightarrow)$*, with*

- $\mathbf{S} = \{(\ell, \mathbf{C}) \mid \mathbf{C} \subseteq I(\ell)\}$, $\mathbf{s}_0 = \big(\ell_0, (\bigwedge_{1 \leq i \leq H} x_i = 0)^{\nearrow} \wedge I(\ell_0)\big)$*, and*
- $\big((\ell, \mathbf{C}), e, (\ell', \mathbf{C}')\big) \in \Rightarrow$ *if* $e = (\ell, g, a, R, \ell') \in E$ *and*

$$\mathbf{C}' = \big([(\mathbf{C} \wedge g)]_R \wedge I(\ell')\big)^{\nearrow} \wedge I(\ell') \textit{ with } \mathbf{C}' \textit{ satisfiable.}$$

That is, in the parametric zone graph, nodes are symbolic states, and arcs are labeled by *edges* of the original PTA.

2.2 Reachability synthesis

We use reachability synthesis to solve the problems defined in Section 2.3. This procedure, called EFsynth, takes as input a PTA $\mathcal{A}$ and a set of target locations L_{target}, and attempts to synthesize all parameter valuations v for which L_{target} is reachable in $v(\mathcal{A})$. $\mathsf{EFsynth}(\mathcal{A}, L_{target})$ was formalized in e.g., [24] and is a procedure that may not terminate, but that computes an exact result (sound and complete) if it terminates.

2.3 Execution-time opacity problems

We recall here the notion of execution-time opacity [9, 7]. This form of opacity is such that the observation is limited to the time to reach a given location. This section recalls relevant definitions from [9, 7].

Given a TA $v(\mathcal{A})$ and a run ρ, we say that ℓ_{priv} is *visited on the way to* ℓ_f *in* ρ if ρ is of the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_\mathrm{m}, \mu_m), (d_m, e_m), \cdots (\ell_\mathrm{n}, \mu_n)$
for some $m, n \in \mathbb{N}$ such that $\ell_\mathrm{m} = \ell_{priv}$, $\ell_\mathrm{n} = \ell_\mathrm{f}$ and $\forall 0 \leq i \leq n-1, \ell_\mathrm{i} \neq \ell_\mathrm{f}$. We denote by $\mathit{Visit}^{priv}(v(\mathcal{A}))$ the set of those runs, and refer to them as *private* runs. We denote by $\mathit{DVisit}^{priv}(v(\mathcal{A}))$ the set of all the durations of these runs.

Conversely, we say that ℓ_{priv} is *avoided on the way to* ℓ_f *in* ρ if ρ is of the form $(\ell_0, \mu_0), (d_0, e_0), (\ell_1, \mu_1), \cdots, (\ell_\mathrm{n}, \mu_n)$
with $\ell_\mathrm{n} = \ell_\mathrm{f}$ and $\forall 0 \leq i < n, \ell_\mathrm{i} \notin \{\ell_{priv}, \ell_\mathrm{f}\}$. We denote the set of those runs by $\mathit{Visit}^{\overline{priv}}(v(\mathcal{A}))$, referring to them as *public* runs, and by $\mathit{DVisit}^{\overline{priv}}(v(\mathcal{A}))$ the set of all the durations of these public runs. Therefore, $\mathit{DVisit}^{priv}(v(\mathcal{A}))$ (resp. $\mathit{DVisit}^{\overline{priv}}(v(\mathcal{A}))$) is the set of all the durations of the runs for which ℓ_{priv} is visited (resp. avoided) on the way to ℓ_f. These concepts can be seen as the set of execution times from the initial location ℓ_0 to the final location ℓ_f while visiting (resp. not visiting) a private location ℓ_{priv}. Observe that, from the definition of the duration of a run, this "execution time" does not include the time spent in ℓ_f.

We now recall formally the concept of "execution-time opacity (ET-opacity) for a set of durations (or execution times) D": a system is *ET-opaque for execution times* D whenever, for any duration in D, it is not possible to deduce whether the system visited ℓ_{priv} or not.

▶ **Definition 7** (Execution-time opacity (ET-opacity) for D). *Given a TA* $v(\mathcal{A})$ *and a set of execution times* D*, we say that* $v(\mathcal{A})$ *is* execution-time opaque (ET-opaque) for execution times D *if* $D \subseteq (\mathit{DVisit}^{priv}(v(\mathcal{A})) \cap \mathit{DVisit}^{\overline{priv}}(v(\mathcal{A})))$.

In the following, we will be interested in the *existence* of such an execution time. We say that a TA is ∃-ET-opaque if it is ET-opaque for a non-empty set of execution times.

▶ **Definition 8** (∃-ET-opacity). *A TA* $v(\mathcal{A})$ *is* ∃-ET-opaque *if* $(\mathit{DVisit}^{priv}(v(\mathcal{A})) \cap \mathit{DVisit}^{\overline{priv}}(v(\mathcal{A}))) \neq \emptyset$.

In addition, a system is *fully ET-opaque* if, for any possible measured execution time, an attacker is not able to deduce whether ℓ_{priv} was visited or not.

▶ **Definition 9** (full ET-opacity). *A TA* $v(\mathcal{A})$ *is* fully ET-opaque *if* $\mathit{DVisit}^{priv}(v(\mathcal{A})) = \mathit{DVisit}^{\overline{priv}}(v(\mathcal{A}))$.

▶ **Example 10.** Consider again the PTA $\mathcal{A}$ in Figure 1a. Let v s.t. $v(p_1) = 1$ and $v(p_2) = 4$. Then $v(\mathcal{A})$ is ∃-ET-opaque since there is at least one execution time for which $v(\mathcal{A})$ is ET-opaque. Here, $v(\mathcal{A})$ is ET-opaque for execution times $[1, 3]$. However, $v(\mathcal{A})$ is not fully

ET-opaque since there is at least one execution time for which $v(\mathcal{A})$ is not ET-opaque. Here, $v(\mathcal{A})$ is not ET-opaque for execution times $[0, 1)$ (which can only occur on a public run) and for execution times $(3, 4]$ (which can only occur on a private run).

Let us consider the following decision problems:

∃-ET-opacity p emptiness problem (∃OE):
INPUT: A PTA $\mathcal{A}$
PROBLEM: Decide the emptiness of the set of valuations v s.t. $v(\mathcal{A})$ is ∃-ET-opaque.

Full ET-opacity p emptiness problem (FOE):
INPUT: A PTA $\mathcal{A}$
PROBLEM: Decide the emptiness of the set of valuations v s.t. $v(\mathcal{A})$ is fully ET-opaque.

The synthesis counterpart allows for a higher-level problem aiming at synthesizing (ideally the entire set of) parameter valuations v for which $v(\mathcal{A})$ is ∃-ET-opaque or fully ET-opaque.

∃-ET-opacity p synthesis problem (∃OS):
INPUT: A PTA $\mathcal{A}$
PROBLEM: Synthesize the set of all valuations v s.t. $v(\mathcal{A})$ is ∃-ET-opaque.

Full ET-opacity p synthesis problem (FOS):
INPUT: A PTA $\mathcal{A}$
PROBLEM: Synthesize the set of all valuations v s.t. $v(\mathcal{A})$ is fully ET-opaque.

3 A parametric execution times-based semi-algorithm for ∃OS and FOS

One of our main results is the proof that both ∃OS and FOS can be deduced from set operations on two sets representing respectively all the durations and parameter valuations of the runs for which ℓ_{priv} is reached (resp. avoided) on the way to ℓ_f. Those sets can be seen as a parametrized version of $DVisit^{priv}(v(\mathcal{A}))$ and $DVisit^{\overline{priv}}(v(\mathcal{A}))$. In order to compute such sets, we propose here the novel notion of parametric execution times. (Note that our partial solution for PET construction and semi-algorithms for ∃OS and FOS work perfectly for *rational*-valued parameters too, and that they are not restricted to 1-clock PTAs.)

3.1 Parametric execution times

The parametric execution times (PET) are the parameter valuations and execution times of the runs to ℓ_f.

▶ **Definition 11.** *Given a PTA $\mathcal{A}$ with final location ℓ_f, the* parametric execution times *of $\mathcal{A}$ are defined as $PET(\mathcal{A}) = \{(v, d) \mid \exists \rho$ in $v(\mathcal{A})$ such that $d = dur(\rho) \wedge \rho$ is of the form $(\ell_0, \mu_0), (d_0, e_0), \cdots, (\ell_n, \mu_n)$ for some $n \in \mathbb{N}$ such that $\ell_n = \ell_f$ and $\forall 0 \leq i \leq n-1, \ell_i \neq \ell_f\}$.*

By definition, we only consider paths up to the point where ℓ_f is reached, meaning that execution times do not include the time elapsed in ℓ_f, and that runs that reach ℓ_f more than once are only considered up to their first visit of ℓ_f.

▶ **Example 12.** Consider again the PTA $\mathcal{A}$ in Figure 1a. Then $PET(\mathcal{A})$ is $(d \leq 3 \wedge p_1 \geq 0 \wedge p_2 \geq 0) \vee (0 \leq p_1 \leq 3 \wedge p_1 \leq d \leq p_2)$.

Partial solution

Synthesizing parametric execution times is in fact equivalent to a reachability synthesis where the PTA is enriched (in particular by adding a clock measuring the total execution time).

▶ **Proposition 13.** *Let $\mathcal{A}$ be a PTA, and ℓ_f the final location of $\mathcal{A}$.*
Let $\mathcal{A}'$ be a copy of $\mathcal{A}$ s.t.:

- *a clock x_{abs} is added and initialized at* 0 *(it does not occur in any guard or reset);*
- *a parameter d is added;*
- *ℓ_f is made* urgent *(i.e., time is not allowed to pass in ℓ_f), all outgoing edges from ℓ_f are pruned and a guard $x_{abs} = d$ is added to all incoming edges to ℓ_f.*

Then, $PET(\mathcal{A}) = \mathsf{EFsynth}(\mathcal{A}', \{\ell_\mathrm{f}\})$.

Proof. By having ℓ_f being urgent and removing its outgoing edges, we ensure that the runs that reach ℓ_f in $\mathcal{A}'$ are all of the form $(\ell_0, \mu_0), (d_0, e_0), \cdots, (\ell_n, \mu_n)$ for some $n \in \mathbb{N}$ such that $\ell_n = \ell'$ and $\forall 0 \leq i \leq n-1, \ell_i \neq \ell'$. By having a clock x_{abs} that is never reset and ℓ_f being urgent, we ensure that for any run ρ that reaches ℓ_f in $\mathcal{A}'$, the value of x_{abs} in the final state if equals to $dur(\rho)$. By having a guard $x_{abs} = d$ on all incoming edges to ℓ_f, we ensure that $d = dur(\rho)$ on any run ρ that reaches ℓ_f.

Therefore, $\mathsf{EFsynth}(\mathcal{A}', \{\ell_\mathrm{f}\})$ contains all parameter valuations of the runs to ℓ_f in $\mathcal{A}$ that stop once ℓ_f is reached, along with the duration of those runs contained in d. ◀

▶ **Example 14.** Consider again the PTA $\mathcal{A}$ in Figure 1a. Then $\mathcal{A}'$ is given in Figure 1b.

As per Lemma 33 in Section A, there exist semi-algorithms for reachability synthesis, and hence for the PET synthesis problem – although they do not guarantee termination.

3.2 ∃OS and FOS problems

Now, we detail how the PET can be used to compute the solution to both ∃OS and FOS. To do so, we will go through a (larger) intermediate problem: the synthesis of both parameter valuations v *and* execution times for which $v(\mathcal{A})$ is ET-opaque.

∃-ET-opacity p-d synthesis problem (d-∃OS):
Input: A PTA $\mathcal{A}$
Problem: Synthesize the set of parameter valuations v and execution times d s.t. $v(\mathcal{A})$ is ∃-ET-opaque and $v(\mathcal{A})$ is ET-opaque for execution time d.

Full ET-opacity p-d synthesis problem (d-FOS):
Input: A PTA $\mathcal{A}$
Problem: Synthesize the set of parameter valuations v and execution times d s.t. $v(\mathcal{A})$ is fully ET-opaque and d is the set of durations of all runs in $v(\mathcal{A})$.

First, given a PTA $\mathcal{A}$ and two locations ℓ_f and ℓ_{priv} of $\mathcal{A}$, let us formally define both sets representing respectively all the durations and parameter valuations of the runs for which ℓ_{priv} is reached (resp. avoided) on the way to ℓ_f.

Let $\mathcal{A}^{\ell_{priv}}_{\ell_\mathrm{f}}$ be a copy of $\mathcal{A}$ s.t.: 1) a Boolean variable[1] b is added and initialized to *False*, 2) b is set to *True* on all incoming edges to ℓ_{priv}, 3) a guard $b =$ *True* is added to all incoming edges to ℓ_f. The PTA $\mathcal{A}^{\ell_{priv}}_{\ell_\mathrm{f}}$ contains all runs of $\mathcal{A}$ for which ℓ_{priv} is reached on the way to ℓ_f, and $PET(\mathcal{A}^{\ell_{priv}}_{\ell_\mathrm{f}})$ contains the durations and parameter valuations of those runs.

[1] Which is a convenient syntactic sugar for doubling the number of locations.

Let $\mathcal{A}_{\ell_f}^{\neg \ell_{priv}}$ be a copy of $\mathcal{A}$ s.t. all incoming and outgoing edges to and from ℓ_{priv} are pruned. The PTA $\mathcal{A}_{\ell_f}^{\neg \ell_{priv}}$ contains all runs of $\mathcal{A}$ for which ℓ_{priv} is avoided on the way to ℓ_f, and $PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})$ contains the durations and parameter valuations of those runs.

▶ **Example 15.** Consider again the PTA $\mathcal{A}$ in Figure 1a. Then $\mathcal{A}_{\ell_f}^{\ell_{priv}}$ is given in Figure 1c, and $\mathcal{A}_{\ell_f}^{\neg \ell_{priv}}$ is given in Figure 1d.

▶ **Proposition 16.** *Given a PTA $\mathcal{A}$, we have: $\texttt{d-}\exists\texttt{OS}(\mathcal{A}) = PET(\mathcal{A}_{\ell_f}^{\ell_{priv}}) \cap PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})$.*

Proof. By definition, $\texttt{d-}\exists\texttt{OS}(\mathcal{A})$ is the synthesis of parameter valuations v and execution times D_v such that $v(\mathcal{A})$ is opaque w.r.t. ℓ_{priv} on the way to ℓ_f for these execution times D_v. This means that $\texttt{d-}\exists\texttt{OS}(\mathcal{A})$ contains exactly all parameter valuations and execution times for which there exist both at least one run in $\mathcal{A}_{\ell_f}^{\ell_{priv}}$ and at least one run in $\mathcal{A}_{\ell_f}^{\neg \ell_{priv}}$. Since PET are the synthesis of the parameter valuations and execution times up to the final location, $\texttt{d-}\exists\texttt{OS}(\mathcal{A})$ is equivalent to the intersection of the $PET(\mathcal{A}_{\ell_f}^{\ell_{priv}})$ and $PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})$. ◀

▶ **Example 17.** Consider again the PTA $\mathcal{A}$ in Figure 1a. Then $PET(\mathcal{A}_{\ell_f}^{\ell_{priv}})$ is $p_1 \leq d \leq p_2 \wedge 0 \leq p_1 \leq 3$. Moreover, $PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})$ is $0 \leq d \leq 3 \wedge p_1 \geq 0 \wedge p_2 \geq 0$. Hence, $\texttt{d-}\exists\texttt{OS}(\mathcal{A})$ is $0 \leq p_1 \leq d \leq p_2 \wedge d \leq 3$.

In order to compute $\texttt{d-FOS}(\mathcal{A})$, we need to remove from $\texttt{d-}\exists\texttt{OS}(\mathcal{A})$ all parameter valuations v s.t. there is at least one run to ℓ_f in $v(\mathcal{A})$ whose duration is not in the set of execution times for which $v(\mathcal{A})$ is ET-opaque. Parameter valuations and durations of such runs are included in $PET(\mathcal{A}) \setminus \texttt{d-}\exists\texttt{OS}(\mathcal{A})$, which is also the difference between $PET(\mathcal{A}_{\ell_f}^{\ell_{priv}})$ and $PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})$. We note that difference as

$$\mathit{Diff}(\mathcal{A}) = \big(PET(\mathcal{A}_{\ell_f}^{\ell_{priv}}) \cup PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})\big) \setminus \big(PET(\mathcal{A}_{\ell_f}^{\ell_{priv}}) \cap PET(\mathcal{A}_{\ell_f}^{\neg \ell_{priv}})\big)$$

$\mathit{Diff}(\mathcal{A})$ is made of a union of convex polyhedra $\mathbf{C}$ over $\mathbb{P}$ (i.e., the parameters of $\mathcal{A}$) and d, which is the duration of runs. The parameter values in those polyhedra are the ones we do not want to see in $\texttt{d-FOS}(\mathcal{A})$. Our solution thus consists in removing from $\texttt{d-}\exists\texttt{OS}(\mathcal{A})$ the values of $\mathbb{P}$ in $\mathit{Diff}(\mathcal{A})$.

▶ **Proposition 18.** *Given a PTA $\mathcal{A}$ with parameter set $\mathbb{P}$: $\texttt{d-FOS}(\mathcal{A}) = \texttt{d-}\exists\texttt{OS}(\mathcal{A}) \setminus \mathit{Diff}(\mathcal{A})\downarrow_{\mathbb{P}}$.*

Proof. See Section B.1. ◀

▶ **Example 19.** Consider again the PTA $\mathcal{A}$ in Figure 1a. We have $\mathit{Diff}(\mathcal{A})$ is $(0 \leq p_1 \leq 3 < d \leq p_2) \vee (0 \leq d \leq 3 \wedge d < p_1 \wedge p_2 \geq 0) \vee (0 \leq p_2 < d \leq 3 \wedge p_1 \geq 0)$. Then $\mathit{Diff}(\mathcal{A})\downarrow_{\mathbb{P}}$ is $(0 \leq p_1 \leq 3 < p_2) \vee (0 < p_1 \wedge p_2 \geq 0) \vee (0 \leq p_2 < 3 \wedge p_1 \geq 0)$. Hence, $\texttt{d-FOS}(\mathcal{A})$ is $p_1 = 0 \leq d \leq p_2 = 3$.

Finally, obtaining $\exists\texttt{OS}(\mathcal{A})$ and $\texttt{FOS}(\mathcal{A})$ is trivial since, by definition, $\exists\texttt{OS}(\mathcal{A}) = (\texttt{d-}\exists\texttt{OS}(\mathcal{A}))\downarrow_{\mathbb{P}}$ and $\texttt{FOS}(\mathcal{A}) = (\texttt{d-FOS}(\mathcal{A}))\downarrow_{\mathbb{P}}$.

▶ **Example 20.** Consider again the PTA $\mathcal{A}$ in Figure 1a. Then $\exists\texttt{OS}(\mathcal{A})$ is $0 \leq p_1 \leq p_2 \wedge p_1 \leq 3$. And $\texttt{FOS}(\mathcal{A})$ is $p_1 = 0 \wedge p_2 = 3$.

On correctness and termination

We described here a method for computing $\exists\texttt{OS}(\mathcal{A})$ and $\texttt{FOS}(\mathcal{A})$ for a PTA, that produces an exact (sound and complete) result if it terminates. It relies on the PET of two subsets of the PTA, the computation of which requires enrichment with one clock and one parameter. If they can be computed, those PET take the form of a finite union of convex polyhedra, on which are then applied the union, intersection, difference and projection set operations – that are known to be decidable in this context. Thus the actual termination of the whole semi-algorithm relies on the reachability synthesis of two $(n+1, 0, m+1)$-PTAs. Reachability synthesis is known to be effectively computable for $(1, 0, m)$-PTAs [10], and cannot be achieved for PTAs with 3 parametric clocks or more due to the undecidability of the reachability emptiness problem [2]. For the semi-algorithm we proposed here for $\exists$OS and FOS problems, we therefore do not have any guarantees of termination, even with only one parametric clock (due to the additional clock x_{abs}), although this might change depending on future results regarding the decidability of reachability synthesis for PTAs with 2 parametric clocks (a first decidability result for the emptiness only was proved for $(2, *, 1)$-PTAs over discrete time [21]).

4 Decidability and undecidability of FOE for 1-clock-PTAs

In this section, we:

1. propose a method to compute potentially infinite PET on $(1, 0, *)$-PTAs, i.e., PTAs with 1 parametric clock and arbitrarily many parameters (Section 4.1);
2. prove decidability of the FOE problem for $(1, 0, 1)$-PTAs, by rewriting infinite PET in a variant of Presburger arithmetic (Section 4.2);
3. prove undecidability of the FOE problem for $(1, 0, *)$-PTAs (Section 4.2).

4.1 Encoding infinite PET for $(1, 0, *)$-PTAs

Given a PTA $\mathcal{A}$ with exactly 1 clock, the goal of the method described here is to guarantee termination of the computation of $PET(\mathcal{A})$ with an exact result. If the partial solution given in Section 3.1 is applied, it amounts to a reachability synthesis on a PTA with 2 clocks, without guarantee of termination. The gist of this method is a form of divide and conquer, where we solve sub-problems, specifically reachability synthesis on sub-parts of $\mathcal{A}$ without adding an additional clock. The first step consists of building some reset-free PTAs, each representing a meaningful subset of the paths joining two given locations in $\mathcal{A}$. $PET(\mathcal{A})$ is then obtained by combining the results of reachability synthesis performed on those reset-free PTAs. The result is encoded in a (finite) regular expression that represents an infinite union of convex polyhedra. Note that this method works perfectly for rational-valued parameters.

4.1.1 Defining the set of reset-free PTAs

Each of the PTAs we build describes parts of the behavior between two locations. More precisely, they represent all the possible paths such that clock resets may occur only on the last transition of the path. We first define the set of locations that we may need based on whether they are initial, final, or reached by a transition associated to a reset.

▶ **Definition 21** (Final-reset paths $FrP(\mathcal{A}, \ell_f)$). *Let $\mathcal{A}$ be a 1-clock PTA, ℓ_0 its initial location and ℓ_f a location of $\mathcal{A}$. We define as $FrP(\mathcal{A}, \ell_f)$ the set of pairs of locations s.t. $\forall(\ell_i, \ell_j) \in FrP(\mathcal{A}, \ell_f)$*

- *$\ell_i = \ell_0$, or $\ell_i \neq \ell_f$ and there is a clock reset on an incoming edge to ℓ_i,*
- *$\ell_j = \ell_f$, or there is a clock reset on an incoming edge to ℓ_j.*

For each pair of states (ℓ_i, ℓ_j) as defined above, we build a reset-free PTA. If the target state ℓ_j is not final (which is a special case), the reset-free PTA models every path going from ℓ_i to ℓ_j and that ends with a reset on its last step. In particular, this ensures that ℓ_j is reached with clock valuation 0.

▶ **Definition 22** (Reset-free PTA $\mathcal{A}(\ell_i, \ell_j)$). *Let $\mathcal{A}$ be a 1-clock PTA, x its unique clock, and ℓ_i, ℓ_j two locations in $\mathcal{A}$. We define as $\mathcal{A}(\ell_i, \ell_j)$ the reset-free PTA obtained from a copy of $\mathcal{A}$ by:*

1. *creating a duplicate ℓ'_j of ℓ_j;*
2. *for all incoming edges (ℓ, g, a, R, ℓ_j) where $R = \emptyset$, removing (ℓ, g, a, R, ℓ_j) and adding an incoming edge (ℓ, g, a, R, ℓ'_j);*
3. *if $\ell_j \neq \ell_\mathrm{f}$, then for all outgoing edges (ℓ_j, g, a, R, ℓ), removing (ℓ_j, g, a, R, ℓ) and adding an outgoing edge (ℓ'_j, g, a, R, ℓ),*
else, making ℓ'_j urgent and adding an edge $(\ell'_j, \mathit{True}, \epsilon, \emptyset, \ell_j)$;
4. *removing any upper bound invariant on ℓ_j and making it urgent;*
5. *if $\ell_i \neq \ell_j$, setting ℓ_i as the initial location,*
else, setting ℓ'_j as the initial location;
6. *removing any clock reset on incoming edges to ℓ_j and pruning all other edges featuring a clock reset, and all outgoing edges from ℓ_f;*
7. *adding a parameter d, and a guard $x = d$ to all incoming edges to ℓ_j;*

We will show next how the reachability synthesis of those reset-free PTAs corresponds to fragments of the runs that are considered in $\mathit{PET}(\mathcal{A})$. For simplification, given $\mathcal{A}$ a 1-clock PTA, and ℓ_i, ℓ_j two locations of $\mathcal{A}$, we now note $Z_{\ell_i,\ell_j} = \mathsf{EFsynth}(\mathcal{A}(\ell_i, \ell_j), \{\ell_j\})$.

4.1.2 Reconstruction of PET from the reachability synthesis of the reset-free PTAs

Given $\mathcal{A}$ a 1-clock PTA, and ℓ_f a location of $\mathcal{A}$, for all $(\ell_i, \ell_j) \in \mathit{FrP}(\mathcal{A}, \ell_\mathrm{f})$ we may compute the parametric zone Z_{ℓ_i,ℓ_j} with guarantee of termination, since the reachability synthesis is decidable on 1-clock PTAs. Those parametric zones may be used to build the (potentially infinite) PET of $\mathcal{A}$. To do so, we first define a (non-parametric, untimed) finite automaton where the states are the locations of $\mathcal{A}$, and the arc between the states ℓ_i and ℓ_j is labeled by Z_{ℓ_i,ℓ_j}. We refer to this automaton as the *automaton of the zones* of $\mathcal{A}$.

▶ **Definition 23** (Automaton of the zones). *Let $\mathcal{A}$ be a 1-clock PTA, ℓ_0 its initial location and ℓ_f a location of $\mathcal{A}$. We define as $\hat{A}$ the finite automaton such that:*

- *The states of $\hat{A}$ are exactly the locations of $\mathcal{A}$;*
- *ℓ_0 is initial and ℓ_f is final;*
- *$\forall(\ell_i, \ell_j) \in \mathit{FrP}(\mathcal{A}, \ell_\mathrm{f})$, there is a transition from ℓ_i to ℓ_j labeled by Z_{ℓ_i,ℓ_j}.*

We claim that the language $\hat{L}$ of $\hat{A}$ is a representation of the times (along with parameter constraints) to go from ℓ_0 to ℓ_f in $\mathcal{A}$. As $\hat{A}$ is a finite automaton, $\hat{L}$ can be represented as a regular expression with three operators: the concatenation (.), the alteration (+), and the Kleene star (*). $\mathit{PET}(\mathcal{A})$ can thus be expressed by redefining those operators with operations on the parametric zones that label edges of $\hat{L}$.

Any parametric zone $Z_{a,b}$ labeling an edge of $\hat{A}$ is of the form $\bigcup_i \mathbf{C}_i$ with $1 \leq i \leq n$ and $\mathbf{C}_i$ a convex polyhedra. As per Definition 6, $\mathbf{C}_i$ is a conjunction of inequalities, each of the form $\alpha d + \sum_{1 \leq i \leq M} \beta_i p_i + \gamma \bowtie 0$, with $p_i \in \mathbb{P}$, and $\alpha, \beta_i, \gamma \in \mathbb{Z}$. Note that x has been replaced by execution times d, as per Definition 11. In the following, we denote by $\mathbf{C}_i^d$ all inequalities such that $\alpha \neq 0$ (i.e., inequalities over d and possibly some parameters in $\mathbb{P}$), and by $\mathbf{C}_i^{\mathbb{P}}$ all inequalities such that $\alpha = 0$ (i.e., inequalities strictly over $\mathbb{P}$). This means that $\mathbf{C}_i = \mathbf{C}_i^d \wedge \mathbf{C}_i^{\mathbb{P}}$. For simplification of what follows, we write inequalities in $\mathbf{C}_i^d$ as $d \bowtie c$ where $c = \frac{\sum_{1 \leq i \leq M} \beta_i p_i + \gamma}{-\alpha}$.

Given $Z_{a,b} = \bigcup_i \mathbf{C}_i$ and $Z_{c,d} = \bigcup_j \mathbf{C}_j$, we define the operators $\bar{.}$, $\bar{*}$ and $\bar{+}$.

Operator $\bar{.}$ is the addition of the time durations and intersection of parameter constraints between two parametric zones. Formally, $Z_{a,b} \;\bar{.}\; Z_{c,d} = \bigcup_{i*j} \mathbf{C}_{i,j}^d \cap \mathbf{C}_{i,j}^{\mathbb{P}}$ such that $\mathbf{C}_{i,j}^{\mathbb{P}} = \mathbf{C}_i^{\mathbb{P}} \wedge \mathbf{C}_j^{\mathbb{P}}$, and for all $d \bowtie c_i \in \mathbf{C}_i^d$ and $d \bowtie' c_j \in \mathbf{C}_j^d$, if $\bowtie, \bowtie' \in \{<, \leq, =\}$ or $\bowtie, \bowtie' \in \{>, \geq, =\}$, then $d \bowtie'' c_i + c_j \in \mathbf{C}_{i,j}^d$ with $\bowtie''$ being in the same direction as $\bowtie$ and $\bowtie'$ and is

- a strict inequality if either $\bowtie$ or $\bowtie'$ is a strict inequality;
- an equality if both $\bowtie$ and $\bowtie'$ are equalities;
- a non-strict inequality otherwise.

Operator $\bar{*}$ is the recursive application of $\bar{.}$ on a parametric zone. Formally, ${Z_{a,b}}^{\bar{*}} = \bigcup_{K \in \mathbb{N}} \{d = 0\}(\bar{.}Z_{a,b})^K$ where $(\bar{.}Z_{a,b})$ is repeated K times, with K being any value in $\mathbb{N}$. Note that $\{d = 0\}$ corresponds to the case where the loop is never taken, and that it is neutral for the $\bar{.}$ operator: $\{d = 0\}\bar{.}Z_{a,b} = Z_{a,b}$. Also note that, in practice, $a = b$ whenever we use this operator.

Operator $\bar{+}$ is the union of two parametric zones. Formally, $Z_{a,b} \bar{+} Z_{c,d} = Z_{a,b} \cup Z_{c,d}$.

Note that the result of any of those operations is a union of convex polyhedra of the form $\bigcup_i \mathbf{C}_i$, meaning that these operators can be nested. Also, this union is infinite whenever operator $\bar{*}$ is present.

▶ **Proposition 24.** *Let $\mathcal{A}$ be a 1-clock PTA and ℓ_{f} a location of $\mathcal{A}$. Let $\hat{L}$ be the language of the automaton of the zones $\hat{\mathcal{A}}$, and e a regular expression describing $\hat{L}$. Let $\bar{e}$ be the expression obtained by replacing the $.$, $+$ and * operators in e respectively by $\bar{.}$, $\bar{+}$ and $^{\bar{*}}$. We have $\bar{e} = PET(\mathcal{A})$.*

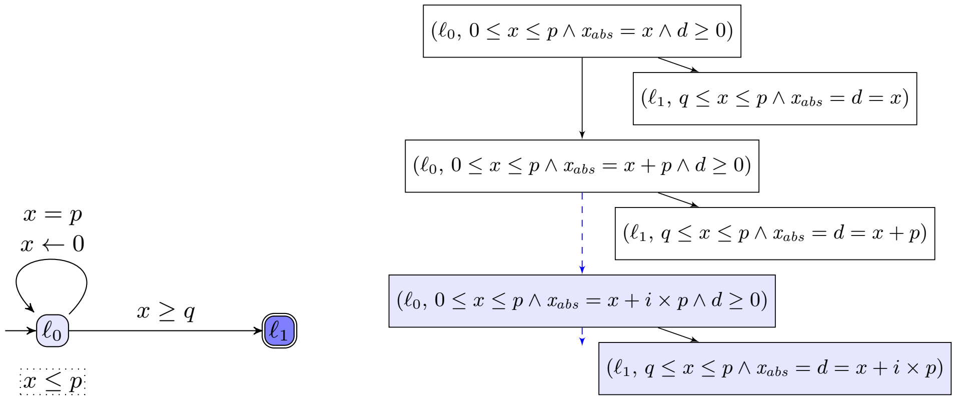
Proof. See Section B.4. ◀

4.1.3 Summary and illustration of the encoding

Given a PTA $\mathcal{A}$ with exactly 1 clock, and given a location ℓ_{f} of $\mathcal{A}$, we compute with an exact result an encoding of $PET(\mathcal{A})$, through the following steps:

1. compute $FrP(\mathcal{A}, \ell_{\mathrm{f}})$, the pairs of locations (ℓ_i, ℓ_j) such that on some run from initial location to ℓ_{f} there might exists a sub-path from ℓ_i to ℓ_j, such that the clock is reset when entering both locations, but never in between;
2. for each of those pairs, compute the reset-free PTA $\mathcal{A}(\ell_i, \ell_j)$, for which reachability synthesis, noted Z_{ℓ_i,ℓ_j} corresponds to the aforementioned sub-paths;
3. generate the automaton of the zones $\hat{\mathcal{A}}$, on which each pair of locations (ℓ_i, ℓ_j) is connected by a transition labeled with Z_{ℓ_i,ℓ_j};
4. compute a regular expression for $\hat{\mathcal{A}}$, which we proved to be equivalent to $PET(\mathcal{A})$. Note that computing a regular expression from a finite automaton is decidable and there exists numerous efficient methods for this [20].

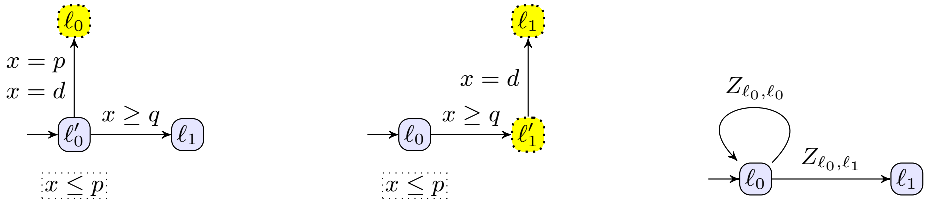
Before discussing how this regular expression can be used to answer the Full ET-opacity p emptiness problem, let us illustrate how it is obtained on a simple example. Figure 2a depicts a 1-clock PTA $\mathcal{A}$ with a clock x and two parameters p and q. We are interested in solving $PET(\mathcal{A})$ where we assume here that ℓ_{f} is ℓ_1. Applying the semi-algorithm from Section 3.1, suppose the addition of a clock x_{abs} and parameter d to the PTA, followed by the computation of the reachability synthesis to ℓ_1. In this case, the algorithm does not terminate though, and as shown in Figure 2b.


$x = p$
$x \leftarrow 0$
$x \geq q$
ℓ_0
ℓ_1
$x \leq p$
$(\ell_0,\, 0 \leq x \leq p \wedge x_{abs} = x \wedge d \geq 0)$
$(\ell_1,\, q \leq x \leq p \wedge x_{abs} = d = x)$
$(\ell_0,\, 0 \leq x \leq p \wedge x_{abs} = x + p \wedge d \geq 0)$
$(\ell_1,\, q \leq x \leq p \wedge x_{abs} = d = x + p)$
$(\ell_0,\, 0 \leq x \leq p \wedge x_{abs} = x + i \times p \wedge d \geq 0)$
$(\ell_1,\, q \leq x \leq p \wedge x_{abs} = d = x + i \times p)$

(a) 1-clock PTA $\mathcal{A}$.

(b) Symbolic (infinite) state space when computing PET in a naive way.

Figure 2 A 1-clock PTA and the PET problem.



(a) $\mathcal{A}(\ell_0, \ell_0)$.

(b) $\mathcal{A}(\ell_0, \ell_1)$.

(c) Automaton of the zones $\hat{\mathcal{A}}$.

Figure 3 Reset-free automata of $\mathcal{A}$ (from Figure 2a) and automaton of the zones $\hat{\mathcal{A}}$.

Following the steps of our method, we have $FrP(\mathcal{A}, \ell_1) = \{(\ell_0, \ell_0), (\ell_0, \ell_1)\}$. Figures 3a and 3b depict the corresponding reset-free automata while Figure 3c gives the automaton of the zones. Urgent locations are colored in yellow.

Reachability synthesis of the reset-free automata gives $Z_{\ell_0,\ell_0} = \{d = p\}$ and $Z_{\ell_0,\ell_1} = \{q \leq d \leq p\}$. As per Proposition 24, the expression $\bar{e}$ (obtained by replacing operators in the regular expression of the language of $\hat{\mathcal{A}}$) is equivalent to $PET(\mathcal{A})$ (again taking ℓ_1 as final location). That expression can be easily obtained (for example with a state elimination method) and gives $\bar{e} = (Z_{\ell_0,\ell_0})^{\bar{*}} \,\bar{.}\, Z_{\ell_0,\ell_1}$. We may then develop operations on $\bar{e}$ and obtain the following infinite disjunction of parametric zones.

$$\begin{aligned} PET(\mathcal{A}) &= (Z_{\ell_0,\ell_0})^{\bar{*}} \,\bar{.}\, Z_{\ell_0,\ell_1} \\ &= \{d = p\}^{\bar{*}} \bar{.}\, \{q \leq d \leq p\} \\ &= \{d = 0 \vee d = p \vee d = 2p \vee \ldots\}.\{q \leq d \leq p\} \\ &= \{q \leq d \leq p \vee q + p \leq d \leq 2p \vee q + 2p \leq d \leq 3p \vee \ldots\} \end{aligned} \tag{1}$$

4.2 Solving the FOE problem through a translation of PET to parametric Presburger arithmetic

Presburger arithmetic is the first order theory of the integers with addition. It is a useful tool that can represent and manipulate sets of integers called semi-linear sets. Those sets are particularly meaningful to study TAs, as the set of durations of runs reaching the final location can be described by a semi-linear set [14]. Presburger arithmetic is however not expressive enough to represent durations of runs in PTAs due to the presence of parameters. In [27], a parametric extension of Presburger arithmetic was considered, introducing linear parametric semi-linear sets (LpSl sets) which are functions associating to a parameter valuation v a (traditional) semi-linear set of the following form:

$$\begin{aligned} S(v) = \Big\{ x \in \mathbb{N}^m \mid \bigvee_{i \in I} \exists x_0, \dots x_{n_i} \in \mathbb{N}^m, k_1, \dots k_{n_i} \in \mathbb{N}, x = \sum_{j=0}^{n_i} x_j \\ \wedge b_0^i(v) \leq x_0 \leq c_0^i(v) \wedge \bigwedge_{j=1}^{n_i} k_j b_j^i(v) \leq x_j \leq k_j c_j^i(v) \Big\} \end{aligned} \tag{2}$$

where I is a finite set and the b_j^i and c_j^i are affine functions with coefficients in $\mathbb{N}$. A 1-LpSl set is an LpSl set defined over a single parameter. Given two LpSl (resp. 1-LpSl) sets S_1 and S_2, the LpSl (resp. 1-LpSl) equality problem consists in deciding whether there exists a parameter valuation v such that $S_1(v) = S_2(v)$.

▶ **Theorem 25** ([27]). *The LpSl equality problem is undecidable.*

The 1-LpSl equality problem is decidable. Moreover, the set of valuations achieving equality can be computed.

The main goal of this subsection is to relate the expressions computed in Section 4.1 to LpSl sets in order to tackle ET-opacity problems. Since Presburger arithmetic is a theory of integers, we have to restrict PTAs to integer parameters; this is what prevents our results to be extended to rational-valued parameters in a straightforward manner. Moreover, we need to focus on time durations of runs with integer values. This second restriction however is without loss of generality. Indeed, in [8, Theorem 5], a trick is provided (which consists mainly in doubling every term of the system so that any run duration that used to be a rational of the form $\frac{q}{2}$ is now an integer to ensure that if a set is non-empty, it contains an integer. This transformation also allows one to consider only non-strict constraints, and thus we assume every constraint is non-strict in the following.

▶ **Theorem 26.** *The LpSl equality problem reduces to the* `FOE` *problem for* $(1, 0, *)$*-PTAs.*

Moreover, the `FOE` *problem for* $(1, 0, 1)$*-PTAs reduces to the 1-LpSl equality problem.*

Sketch of proof. From Equation (2) one can see that an LpSl set parametrically defines integers that are the sum of two types of elements: x_0 belongs to an interval, while the x_j represent a sum of integers, each coming from the interval $[b_j^i; c_j^i]$. Intuitively, we separate a run into its elementary path until the final state and its loops. We use x_0 to represent the duration of the elementary path, and the x_j adds the duration of loops. Each occurrence of the same loop within a run being independent (as they include a reset of the clock), their durations all belong to the same interval.

Formally, given a PTA $\mathcal{A}$, using Section 3.2, we build the PTAs $\mathcal{A}_{\ell_f}^{\ell_{priv}}$ and $\mathcal{A}_{\ell_f}^{\neg\ell_{priv}}$ separating the private and public runs of $\mathcal{A}$. Then with Section 4.1, we obtain expressions $\bar{e}_{\ell_{priv}}$ and $\bar{e}_{\neg\ell_{priv}}$ such that (Proposition 24) $\bar{e}_{\ell_{priv}} = PET(\mathcal{A}_{\ell_f}^{\ell_{priv}})$ and $\bar{e}_{\neg\ell_{priv}} = PET(\mathcal{A}_{\ell_f}^{\neg\ell_{priv}})$. We then develop and simplify these expressions until we can build LpSl sets representing the integers accepted by each expression. We can then show the inter-reduction as the full ET-opacity is directly equivalent to the equality of the two sets. Note that one direction of the reduction is stronger, allowing multiple parameters. This is due to constraints over the parameters which may appear in our expressions, but cannot be transferred to LpSl sets. However, when there is a single parameter, one can easily resolve these constraints beforehand. See Section B.5 for a complete proof. ◀

Combining Theorems 25 and 26 directly gives us:

▶ **Corollary 27.** *FOE is undecidable for* $(1, 0, *)$*-PTAs.*

▶ **Corollary 28.** *FOE is decidable for* $(1, 0, 1)$*-PTAs and FOS can be solved.*

5 Decidability of ∃OE for $(1, 0, *)$-PTAs for integer-valued parameters

We prove here the decidability of ∃OE for $(1, 0, *)$-PTAs with integer parameters over dense time (Section 5.1); we also prove that the same problem is in EXPSPACE for $(1, *, 1)$-PTAs over discrete time (Section 5.2).

5.1 General case

Adding the divisibility predicate (denoted "|") to Presburger arithmetic produces an undecidable theory, whose purely existential fragment is known to be decidable [26]. The FOE problem can be encoded in this logic, but requires a single quantifier alternation, which goes beyond the aforementioned decidability result, leading us to rely on [27]. The ∃OE problem however can be encoded in the purely existential fragment.

▶ **Theorem 29.** *The ∃OE problem is decidable.*

Sketch of proof. As for Theorem 26, we start by building and simplifying expressions representing the private and public durations of the PTA. Instead of translating the expression into LpSl set however, we now use Presburger with divisibility.

Again, a run can be decomposed in the run without loops, and its looping parts. The duration of the former is defined directly by conjunction of inequalities, which can be formulated in a Presburger arithmetic formula. The latter requires the divisibility operator to represent the arbitrary number of loops. Hence, we can build a formula accepting exactly the integers satisfying our expressions. Deciding the ∃OE problem can be achieved by testing the existence of an integer satisfying the formulas produced from both expressions, which can be stated in a purely existential formula. See [11] for a complete proof. ◀

▶ Remark 30 (complexity). Let us quickly discuss the complexity of this algorithm. The expressions produced by Proposition 24 can, in the worst case, be exponential in the size of the PTA. This formula was then simplified within the proof of Theorem 26, in part by developing it, which could lead to an exponential blow-up. Finally, the existential fragment of Presburger arithmetic with divisibility can be solved in NEXPTIME [26]. As a consequence, our algorithm lies in 3NEXPTIME.

5.2 Discrete time case

There are clear ways to improve the complexity of this algorithm. In particular, we finally prove an alternative version of Theorem 29 in a more restricted setting ($\mathbb{T} = \mathbb{N}$), but with a significantly lower complexity upper bound and using completely different proof ingredients [21].

▶ **Theorem 31.** *∃OE is decidable in* EXPSPACE *for* $(1, *, 1)$*-PTAs over discrete time.*

Proof. See [11]. ◀

▶ Remark 32. The fact that we can handle arbitrarily many non-parametric clocks in Theorem 31 does not improve Theorem 29: over discrete time, it is well-known that non-parametric clocks can be eliminated using a technique from [2], and hence come "for free".

Table 1 Execution-time opacity problems for PTAs: contributions and some open cases.

Time	(pc, npc, p)	∃OE emptiness	∃OE synthesis
dense	$(1, 0, *)$	✓ Th. 29	?
dense	$(1, *, *)$	?	?
dense	$(2, 0, 1)$	?	?
dense	$(3, 0, 1)$	× [9, Th.6.1]	×
discrete	$(1, *, 1)$	✓EXPSPACE Th. 31	?

Time	(pc, npc, p)	FOE emptiness	FOE synthesis
dense	$(1, 0, 1)$	✓ Corol. 28	✓ Corol. 28
dense	$(1, 0, [2, M])$	?	?
dense	$(1, 0, M)$	× Corol. 27	×
dense	$([2, 3], 0, 1)$	?	?
dense	$(4, 0, 2)$	× [9, Th. 7.1]	×

6 Conclusion and perspectives

In this paper, we addressed the ET-opacity for 1-clock PTAs with integer-valued parameters over dense time. We proved that 1) FOE is undecidable for a sufficiently large number of parameters, 2) FOE becomes decidable for a single parameter, and 3) ∃OE is decidable, in 3NEXPTIME over dense time and in EXPSPACE over discrete time. These results rely on a novel construction of PET, for which a sound and complete computation method is provided. In the general case, we provided semi-algorithms for the computation of PET, ∃OS and FOS.

The undecidability result reduces from a problem in parametric Presburger arithmetic, itself reducing from Hilbert's tenth problem. The latter is known to be undecidable for various classes of polynomials (with degree 4 and 58 variables for instance). The number of parameters used in the undecidability of the parametric Presburger arithmetic problem is not direct from their proof but we can estimate that at least 200 parameters are needed. Closing the gap through this approach would require important developments in Diophantine analysis. The opacity problem hence remains open for many cases of low number of parameters.

Our PET constructions and all PET-related results work perfectly for rational-valued parameters. It remains however unclear how to extend our (un)decidability results to rational-valued parameters, as our other proof ingredients (notably using the Presburger arithmetics) heavily rely on integer-valued parameters.

It remains also unclear whether synthesis can be achieved using techniques from [21], explaining the "open" cell in the "discrete time" row of Table 1. Also, a number of problems remain open in Table 1, notably the 2-clock case, already notoriously difficult for reachability emptiness [2, 21].

Finally, exploring *weak* ET-opacity [7] (which allows the attacker to deduce that the private location was *not* visited) is also on our agenda.

References

1 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994. doi:10.1016/0304-3975(94)90010-8.

2 Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *STOC*, pages 592–601, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167242.

3 Ikhlass Ammar, Yamen El Touati, Moez Yeddes, and John Mullins. Bounded opacity for timed systems. *Journal of Information Security and Applications*, 61:1–13, September 2021. doi:10.1016/j.jisa.2021.102926.

4 Jie An, Qiang Gao, Lingtai Wang, Naijun Zhan, and Ichiro Hasuo. The opacity of timed automata. In André Platzer, Kristin-Yvonne Rozier, Matteo Pradella, and Matteo Rossi, editors, *FM*, volume 14933 of *Lecture Notes in Computer Science*, pages 620–637. Springer, 2024. doi:10.1007/978-3-031-71162-6_32.

5 Étienne André. What's decidable about parametric timed automata? *International Journal on Software Tools for Technology Transfer*, 21(2):203–219, April 2019. doi:10.1007/s10009-017-0467-0.

6 Étienne André, Sarah Dépernet, and Engel Lefaucheux. The bright side of timed opacity. In Kazuhiro Ogata, Meng Sun, and Dominique Méry, editors, *ICFEM*, 2024. To appear.

7 Étienne André, Engel Lefaucheux, Didier Lime, Dylan Marinho, and Jun Sun. Configuring timing parameters to ensure execution-time opacity in timed automata. In Maurice H. ter Beek and Clemens Dubslaff, editors, *TiCSA*, Electronic Proceedings in Theoretical Computer Science. Springer, 2023. Invited paper.

8 Étienne André, Engel Lefaucheux, and Dylan Marinho. Expiring opacity problems in parametric timed automata. In Yamine Ait-Ameur and Ferhat Khendek, editors, *ICECCS*, pages 89–98, 2023. doi:10.1109/ICECCS59891.2023.00020.

9 Étienne André, Didier Lime, Dylan Marinho, and Jun Sun. Guaranteeing timed opacity using parametric timed model checking. *ACM Transactions on Software Engineering and Methodology*, 31(4):1–36, October 2022. doi:10.1145/3502851.

10 Étienne André, Didier Lime, and Nicolas Markey. Language preservation problems in parametric timed automata. *Logical Methods in Computer Science*, 16(1), January 2020. doi:10.23638/LMCS-16(1:5)2020.

11 Étienne André, Johan Arcile, and Engel Lefaucheux. Execution-time opacity problems in one-clock parametric timed automata (extended version). Technical Report abs/2410.01659, arXiv, October 2024. arXiv:2410.01659.

12 Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. Language emptiness of continuous-time parametric timed automata. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 69–81. Springer, July 2015. doi:10.1007/978-3-662-47666-6_6.

13 Arnab Kumar Biswas, Dipak Ghosal, and Shishir Nagaraja. A survey of timing channels and countermeasures. *ACM Computing Surveys*, 50(1):6:1–6:39, 2017. doi:10.1145/3023872.

14 Véronique Bruyère, Emmanuel Dall'Olio, and Jean-Francois Raskin. Durations and parametric model-checking in timed automata. *ACM Transactions on Computational Logic*, 9(2):12:1–12:23, 2008. doi:10.1145/1342991.1342996.

15 Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y. A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008. doi:10.1007/s10207-008-0058-x.

16 Daniel Bundala and Joël Ouaknine. On parametric timed automata and one-counter machines. *Information and Computation*, 253:272–303, 2017. doi:10.1016/j.ic.2016.07.011.

17 Franck Cassez. The dark side of timed opacity. In Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-Hoon Kim, and Sang-Soo Yeo, editors, *ISA*, volume 5576 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2009. doi:10.1007/978-3-642-02617-1_3.

18 Catalin Dima. Real-time automata. *Journal of Automata, Languages and Combinatorics*, 6(1):3–23, 2001. doi:10.25596/jalc-2001-003.

19 Guillaume Gardey, John Mullins, and Olivier H. Roux. Non-interference control synthesis for security timed automata. *Electronic Notes in Theoretical Computer Science*, 180(1):35–53, 2007. doi:10.1016/j.entcs.2005.05.046.

20 Hermann Gruber and Markus Holzer. From finite automata to regular expressions and back - A summary on descriptional complexity. *International Journal of Foundations of Computer Science*, 26(8):1009–1040, 2015. doi:10.1142/S0129054115400110.

21 Stefan Göller and Mathieu Hilaire. Reachability in two-parametric timed automata with one parameter is EXPSPACE-complete. In Markus Bläser and Benjamin Monmege, editors, *STACS*, volume 187 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.36.

22 Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer, 1992. doi:10.1007/BFb0031995.

23 Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002. doi:10.1016/S1567-8326(02)00037-1.

24 Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for real-time systems. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015. doi:10.1109/TSE.2014.2357445.

25 Julian Klein, Paul Kogel, and Sabine Glesner. Verifying opacity of discrete-timed automata. In Nico Plat, Stefania Gnesi, Carlo A. Furia, and Antónia Lopes, editors, *FormaliSE*, pages 55–65. ACM, 2024. doi:10.1145/3644033.3644376.

26 Antonia Lechner, Joël Ouaknine, and James Worrell. On the complexity of linear arithmetic with divisibility. In *LICS*, pages 667–676. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.67.

27 Engel Lefaucheux. When are two parametric semi-linear sets equal? Technical Report hal-04172593, HAL, 2024. URL: https://inria.hal.science/hal-04172593.

28 Laurent Mazaré. Using unification for opacity properties. In Peter Ryan, editor, *WITS*, pages 165–176, April 2004.

29 Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In Nancy A. Lynch and Bruce H. Krogh, editors, *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000. doi:10.1007/3-540-46430-1_26.

30 Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

31 Lingtai Wang and Naijun Zhan. Decidability of the initial-state opacity of real-time automata. In Cliff B. Jones, Ji Wang, and Naijun Zhan, editors, *Symposium on Real-Time and Hybrid Systems - Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday*, volume 11180 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2018. doi:10.1007/978-3-030-01461-2_3.

32 Lingtai Wang, Naijun Zhan, and Jie An. The opacity of real-time automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2845–2856, 2018. doi:10.1109/TCAD.2018.2857363.

A Recalling the correctness of EFsynth

▶ **Lemma 33** ([24]). *Let $\mathcal{A}$ be a PTA, and let L_{target} be a subset of the locations of $\mathcal{A}$. Assume EFsynth$(\mathcal{A}, L_{target})$ terminates with result K. Then $v \models K$ iff L_{target} is reachable in $v(\mathcal{A})$.*

B Proof of results

B.1 Proof of Proposition 18

▶ **Proposition 18.** *Given a PTA $\mathcal{A}$ with parameter set $\mathbb{P}$: $\mathtt{d\text{-}FOS}(\mathcal{A}) = \mathtt{d\text{-}\exists OS}(\mathcal{A}) \setminus \mathit{Diff}(\mathcal{A}){\downarrow_{\mathbb{P}}}$.*

Proof. By definition, $\mathtt{d\text{-}FOS}(\mathcal{A})$ is the synthesis of parameter valuations v (and execution times of their runs) s.t. $v(\mathcal{A})$ is fully opaque w.r.t. ℓ_{priv} on the way to ℓ_f. By definition, $\mathit{Diff}(\mathcal{A}){\downarrow_{\mathbb{P}}}$ is the set of parameter valuations s.t. for any valuation $v \in \mathit{Diff}(\mathcal{A}){\downarrow_{\mathbb{P}}}$, there is at least one run where ℓ_{priv} is reached (resp. avoided) on the way to ℓ_f in $v(\mathcal{A})$ whose duration time is different from those of any run where ℓ_{priv} is avoided (resp. reached) on the way to ℓ_f in $v(\mathcal{A})$. By removing this set of parameters from $\mathtt{d\text{-}\exists OS}(\mathcal{A})$, we are left with parameter valuations (and execution times of their runs) s.t. for any v, any run ρ where ℓ_{priv} is reached (resp. avoided) on the way to ℓ_f in $v(\mathcal{A})$, there is a run ρ' where ℓ_{priv} is avoided (resp. reached) on the way to ℓ_f in $v(\mathcal{A})$ and $dur(\rho) = dur(\rho')$. This is equivalent to our definition of full opacity. ◀

B.2 Proposition 34

▶ **Proposition 34.** *Let $\mathcal{A}$ be a 1-clock PTA, and $(\ell_i, \ell_j) \in FrP(\mathcal{A}, \ell_\mathrm{f})$ such that $\ell_j \neq \ell_\mathrm{f}$. Then Z_{ℓ_i,ℓ_j} is equivalent to the synthesis of parameter valuations v and execution times D_v such that $D_v = \{d \mid \exists\rho$ from $(\ell_i, \{x = 0\})$ to ℓ_j in $v(\mathcal{A})$ such that $d = dur(\rho)$, ℓ_f is never reached, and x is reset on the last edge of ρ and on this edge only $\}$.*

Proof. Let us first consider the case where $\ell_i \neq \ell_j$. Steps 1 to 3 in Definition 22 imply that whenever ℓ_j occurs either as a source or target location in an edge, it is replaced by the duplicate locality ℓ'_j, except when ℓ_j is the target location and x is reset on the edge. At this stage, for any path between ℓ_i and ℓ_j in $\mathcal{A}$, where no incoming edge to ℓ_j featuring a clock reset is present, there is an equivalent path in $\mathcal{A}(\ell_i, \ell_j)$ with ℓ_j being replaced by ℓ'_j. Step 4 implies that whenever ℓ_j is reached in $\mathcal{A}(\ell_i, \ell_j)$ no delay is allowed. As there are no outgoings edges from ℓ_j anymore, and only incoming edges featuring a clock reset, only runs ending with such edges are accepted by the reachability synthesis on ℓ_j. Since the clock value when entering in ℓ_j through such an edge is always 0, removing the upper bound of the invariant does not impact the availability of transitions. Because of our assumption that $\ell_i \neq \ell_j$, Step 5 does not change the initial location. Step 6 ensures that, in any run from ℓ_i to ℓ_j :

- no clock reset is performed before the last edge of the run;
- the clock is not reset when entering ℓ_j, and is therefore equals to the duration of the run;
- ℓ_f is not reached.

Step 7 ensures that d is equal to the value of the clock when entering ℓ_f.

Let us now consider the case where $\ell_i = \ell_j$. In this case, Step 5 changes the initial locality to ℓ'_j. Because of Steps 1 to 3, runs from ℓ'_j to ℓ_j in $\mathcal{A}(\ell_i, \ell_j)$ are identical to runs looping from ℓ_i to ℓ_i in $\mathcal{A}$ where x is reset on the last edge of the run and on this edge only. Restrictions obtained by Steps 4, 6 and 7 are unchanged.

Therefore, Z_{ℓ_i,ℓ_j} is equivalent to the synthesis of parameter valuations v and execution times D_v such that $D_v = \{d \mid \exists\rho$ from $(\ell_i, \{x = 0\})$ to ℓ_j in $v(\mathcal{A})$ such that $d = dur(\rho)$, ℓ_f is never reached, and x is reset on the last edge of ρ and on this edge only. ◀

B.3 Proposition 35

▶ **Proposition 35.** *Let $\mathcal{A}$ be a 1-clock PTA, and $(\ell_i, \ell_j) \in FrP(\mathcal{A}, \ell_\mathrm{f})$ such that $\ell_j = \ell_\mathrm{f}$. Then Z_{ℓ_i,ℓ_j} is equivalent to the synthesis of parameter valuations v and execution times D_v such that $D_v = \{d \mid \exists\rho$ from $(\ell_i, \{x = 0\})$ to ℓ_f in $v(\mathcal{A})$ such that $d = dur(\rho)$, ℓ_f is reached only on the last state of ρ, and x may only be reset on the last edge of ρ $\}$.*

Proof. By Definition 21, we know that $\ell_i \neq \ell_\mathrm{f}$.

Steps 1 to 3 in Definition 22 imply that:

- whenever ℓ_f is the target location of an edge, it is replaced by the duplicate locality ℓ'_j, except when x is reset on the edge;
- once ℓ'_j is reached, no delay is allowed and the only available transition consists in reaching ℓ_f through an empty action ϵ.

At this stage, the only difference between path from ℓ_i to ℓ_f in $\mathcal{A}(\ell_i, \ell_j)$ and $\mathcal{A}$ is that incoming edges to ℓ_f where x is not reset now leads to ℓ'_j, and then to ℓ_f without any added elapsed time. Step 4 implies that whenever ℓ_f is reached in $\mathcal{A}(\ell_i, \ell_j)$ no delay is allowed. As ℓ_f is either entered by the immediate transition from ℓ'_j or feature a clock reset, removing the upper bound of the invariant does not impact the availability of transitions. As $\ell_i \neq \ell_\mathrm{f}$, Step 5 does not change the initial location. Step 6 ensures that, in any run from ℓ_i to ℓ_j :

- no clock reset is performed before the last edge of the run (not counting the ϵ edge from ℓ'_j to ℓ_f);
- the clock value is not reset when entering ℓ_f, and is therefore equal to the duration of the run;
- no action can be taken after reaching ℓ_f.

Step 7 ensures that d is equal to the value of the clock when entering ℓ_f.

Therefore, Z_{ℓ_i,ℓ_j} is equivalent to the synthesis of parameter valuations v and execution times D_v such that $D_v = \{d \mid \exists \rho$ from $(\ell_i, \{x = 0\})$ to ℓ_f in $v(\mathcal{A})$ such that $d = dur(\rho)$, ℓ_f is reached only on the last state of ρ, and x may only be reset on the last edge of ρ. ◀

B.4 Proof of Proposition 24

▶ **Proposition 24.** *Let $\mathcal{A}$ be a 1-clock PTA and ℓ_f a location of $\mathcal{A}$. Let $\hat{L}$ be the language of the automaton of the zones $\hat{\mathcal{A}}$, and e a regular expression describing $\hat{L}$. Let $\bar{e}$ be the expression obtained by replacing the $.$, $+$ and * operators in e respectively by $\bar{.}$, $\bar{+}$ and $\bar{*}$. We have $\bar{e} = PET(\mathcal{A})$.*

Proof. Let us first show that $\bar{e}$ contains $PET(\mathcal{A})$. Let ρ be a path whose time duration and parameter constraints are in $PET(\mathcal{A})$. By definition, ρ starts at time 0 in the initial locality and ends in ℓ_f, with only one occurrence of ℓ_f in the whole path. Let us consider that the clock is reset n times before the last transition, then ρ can be decomposed as $\rho_0 \dots \rho_n$ such that:

- $\forall\, 0 \leq i < n$, sub-path ρ_i starts in ℓ_i at time valuation 0, ends in ℓ_{i+1}, contains a single reset positioned on the last transition (thus ending with time valuation 0) and does not contain any occurrence of ℓ_f;
- sub-path ρ_n starts in ℓ_n at time valuation 0, ends in ℓ_f, may only contain a reset on its last transition, and contains exactly one occurrence of ℓ_f.

By Definition 21, $\forall\, 0 \leq i < n$, $(\ell_i, \ell_{i+1}) \in FrP(\mathcal{A}, \ell_f)$ and by Proposition 34, $Z_{\ell_i,\ell_{i+1}}$ is the synthesis of parameter valuations and execution times of that sub-path. By Definition 21, $(\ell_n, \ell_f) \in FrP(\mathcal{A}, \ell_f)$ and by Proposition 35, Z_{ℓ_n,ℓ_f} is the synthesis of parameter and valuation times of that sub-path. By Definition 23, there is a sequence of transitions $Z_{\ell_0,\ell_1}, \dots, Z_{\ell_i,\ell_{i+1}}, \dots, Z_{\ell_n,\ell_f}$ in the automaton of the zones $\hat{\mathcal{A}}$. By application of operators $\bar{+}$ and $\bar{*}$, that sequence thus exists in $\bar{e}$ as $Z_{\ell_0,\ell_1}\bar{.}\dots\bar{.}Z_{\ell_i,\ell_{i+1}}\bar{.}\dots\bar{.}Z_{\ell_n,\ell_f}$. By definition of operator $\bar{.}$, this expression is the intersection of all parameter constraints and the addition of all valuation times, which is equivalent to $PET(\mathcal{A})$.

Let us now show that $PET(\mathcal{A})$ contains $\bar{e}$. By application of operators $\bar{+}$ and $\bar{*}$, any word in $\bar{e}$ can be expressed as a sequence of concatenation operations $\bar{.}$. By Definition 23, given a word $Z_{\ell_0,\ell_1}\bar{.}\dots\bar{.}Z_{\ell_i,\ell_{i+1}}\bar{.}\dots\bar{.}Z_{\ell_n,\ell n+1} \in \bar{e}$, we know that ℓ_0 is the initial location of $\mathcal{A}$, $\ell_{n+1} = \ell_f$ and $\forall\, 0 \leq i \leq n, \ell_i \neq \ell_f$. By Proposition 34, $\forall\, 0 \leq i < n$, $Z_{\ell_i,\ell_{i+1}}$ is the synthesis of parameter valuations and execution times of paths between ℓ_i and ℓ_{i+1} in $\mathcal{A}$ such that ℓ_f is never reached, and x is reset on the last edge of the path and on this edge only. And by Proposition 35, Z_{ℓ_n,ℓ_f} is the synthesis of parameter valuations and execution times of paths between ℓ_n and ℓ_f in $\mathcal{A}$ such that ℓ_f is reached only on the last state of ρ, and x may only be reset on the last edge of ρ.

Let us assume there exists a path ρ whose time duration and parameter constraints are in $PET(\mathcal{A})$ such that $\rho = \rho_0 \dots \rho_n$ and:

- $\forall\, 0 \leq i < n$, sub-path ρ_i starts in ℓ_i at time valuation 0, ends in ℓ_{i+1}, contains a single reset positioned on the last transition (thus ending with time valuation 0) and does not contain any occurrence of ℓ_f;
- sub-path ρ_n starts in ℓ_n at time valuation 0, ends in ℓ_f, may only contain a reset on its last transition, and contains exactly one occurrence of ℓ_f.

Then $Z_{\ell_0,\ell_1}\bar{\cdot}\dots\bar{\cdot}Z_{\ell_i,\ell_{i+1}}\bar{\cdot}\dots\bar{\cdot}Z_{\ell_n,\ell n+1} \in PET(\mathcal{A})$. On the other hand, if there does not exist such a path, then there exist $0 \leq i \leq n$ such that $Z_{\ell_i,\ell_{i+1}} = \emptyset$. By recursive applications of operator $\bar{\cdot}$, the whole sequence is evaluated as $\emptyset$ and thus contained in $PET(\mathcal{A})$. ◀

B.5 Proof of Theorem 26

▶ **Theorem 26.** *The LpSl equality problem reduces to the* `FOE` *problem for* $(1, 0, *)$*-PTAs. Moreover, the* `FOE` *problem for* $(1, 0, 1)$*-PTAs reduces to the 1-LpSl equality problem.*

Proof. Given a PTA $\mathcal{A}$, we showed in Section 3.2 how to compute two PTAs $\mathcal{A}_{\ell_f}^{\ell_{priv}}$ and $\mathcal{A}_{\ell_f}^{\neg\ell_{priv}}$ separating the private and public runs of $\mathcal{A}$. Then in Section 4.1, we showed how to build expressions $\bar{e}_{\ell_{priv}}$ and $\bar{e}_{\neg\ell_{priv}}$ such that (Proposition 24) $\bar{e}_{\ell_{priv}} = PET(\mathcal{A}_{\ell_f}^{\ell_{priv}})$ and $\bar{e}_{\neg\ell_{priv}} = PET(\mathcal{A}_{\ell_f}^{\neg\ell_{priv}})$.

Note that the operators $\bar{\cdot}$, $\bar{*}$ and $\bar{+}$ are associative and commutative; moreover, each term Z occurring in the expressions $\bar{e}_{\ell_{priv}}$ and $\bar{e}_{\neg\ell_{priv}}$ is a union of constraints $Z = \bigcup_{i'} \mathbf{C}_{i'} = \overline{\boldsymbol{+}}_{i'}\mathbf{C}_{i'}$. As a consequence, we can thus develop the entire expression to the form

$$\overline{\boldsymbol{+}}_i\ (\mathbf{C}_1^i\bar{\cdot}\mathbf{C}_2^i\bar{\cdot}\cdots\bar{\cdot}\mathbf{C}_{n_i}^i)\bar{\cdot}(\mathbf{C}_{n_i+1}^i)^{\bar{*}}\bar{\cdot}(\mathbf{C}_{n_i+2}^i)^{\bar{*}}\bar{\cdot}\cdots\bar{\cdot}(\mathbf{C}_{n_i+m_i}^i)^{\bar{*}}.$$

where we put all $\bar{+}$ outside of the expression. For example, the expression $Z_1\bar{\cdot}(Z_2)^{\bar{*}}$ where $Z_1 = \mathbf{C}_1 \cup \mathbf{C}_2$ and $Z_2 = \mathbf{C}_3 \cup \mathbf{C}_4$ is developed into $\mathbf{C}_1\bar{\cdot}(\mathbf{C}_3)^{\bar{*}}\bar{\cdot}(\mathbf{C}_4)^{\bar{*}}\bar{+}\mathbf{C}_2\bar{\cdot}(\mathbf{C}_3)^{\bar{*}}\bar{\cdot}(\mathbf{C}_4)^{\bar{*}}$.

As $\mathbf{C}^{\bar{*}} = \{d = 0\}\bar{+}\mathbf{C}\bar{\cdot}\mathbf{C}^{\bar{*}}$, for each $\mathbf{C}_{n_i+j}^i$ we can w.l.o.g. express term i as the union of two terms: one where $(\mathbf{C}_{n_i+j}^i)^{\bar{*}}$ is removed (i.e., this loop is never taken), and one where $\mathbf{C}_{n_i+j}^i$ is concatenated to the term (i.e., the loop is taken at least once). This means that each term, is turned into 2^{m_i} terms, where we can assume w.l.o.g. that for each $j > 0$, $\mathbf{C}_{n_i+j}^i = \mathbf{C}_j^i$.

Given an expression of the above form, by definition of $\bar{\cdot}$, the product $\mathbf{C}_1^i\bar{\cdot}\mathbf{C}_2^i\bar{\cdot}\cdots\bar{\cdot}\mathbf{C}_{n_i}^i$ is also a conjunction of inequalities and thus can be expressed as $\mathbf{C}_i^d \wedge \mathbf{C}_i^{\mathbb{P}}$ where $\mathbf{C}_i^{\mathbb{P}}$ is obtained by the constraints that do not involve d while $\mathbf{C}_i^d$ contains the constraints that involve d and potentially some parameters in $\mathbb{P}$. Note also that by the assumption that for each $j > 0$, $\mathbf{C}_{n_i+j}^i = \mathbf{C}_j^i$, any constraint that does not involve d can be removed from $\mathbf{C}_{n_i+j}^i$ without modifying the set. Therefore, the expression can now be rewritten as

$$\overline{\boldsymbol{+}}_i(\mathbf{C}_i^d \wedge \mathbf{C}_i^{\mathbb{P}})\bar{\cdot}(\mathbf{C}_1^i)^{\bar{*}}\bar{\cdot}(\mathbf{C}_2^i)^{\bar{*}}\bar{\cdot}\cdots\bar{\cdot}(\mathbf{C}_{m_i}^i)^{\bar{*}}.$$

where every inequality in $\mathbf{C}_j^i$ involves d.

- Assume the expressions involve a single parameter p. Let us show that the `FOE` problem for PTAs over a single parameter reduces to the 1-LpSl equality problem.

  Every constraint on p is of the form $p \bowtie c$ with $c \in \mathbb{N}$ and $\bowtie \in \{\leq, \geq\}$. Therefore, there exists a constant M such that for all i, either the constraint $\mathbf{C}_i^{\mathbb{P}}$ is satisfied for all $p \geq M$, or it is satisfied by none.

  For any fixed valuation v, full ET-opacity of $v(\mathcal{A})$ is decidable by [7]. We thus assume that we consider only valuations of p greater than M. This can be represented by replacing every occurrence of p in the expressions by $M + p$. This can be done without loss of generality as we can independently test whether the PTA is fully ET-opaque for the finitely many integer values of p smaller than M. When solving the `FOS` problem, we thus need to include the valuations of p smaller than M that achieved equality to the valuations provided by the reduction.

The terms $\mathbf{C}_i^{\mathbb{P}}$ being either always or never valid, one can either remove this constraint from the expression, or the term containing it producing an expression of the form

$$\overline{+}_i \mathbf{C}_0^i \bar{\cdot} (\mathbf{C}_1^i)^{\bar{*}} \bar{\cdot} (\mathbf{C}_2^i)^{\bar{*}} \bar{\cdot} \cdots \bar{\cdot} (\mathbf{C}_{m_i}^i)^{\bar{*}}.$$

where every constraint involves x.
Once again, assuming p is large enough, the constraint $\mathbf{C}_j^i$ can be assumed to be of the form $\alpha_j^i p + \beta_j^i \leq x \leq \gamma_j^i p + \delta_j^i$ where $\alpha_j^i, \beta_j^i, \gamma_j^i, \delta_j^i \in \mathbb{N}$.
For both expressions $\bar{e}_{\ell_{priv}}$ and $\bar{e}_{\neg\ell_{priv}}$, now in the simplified form described above, we build the 1-LpSl sets $S_{\bar{e}_{\ell_{priv}}}$ and $S_{\bar{e}_{\neg\ell_{priv}}}$ where, taking the notations from Equation (2), I is the set $\overline{+}$ ranges over, for $0 \leq j \leq m_i, b_j^i = \alpha_j^i p + \beta_j^i$ and $c_j^i = \gamma_j^i p + \delta_j^i$.
For a valuation v of p, we have that $S_{\bar{e}_{\ell_{priv}}}(v)$ contains exactly the integers that satisfy $v(\bar{e}_{\ell_{priv}})$ (and similarly for $S_{\bar{e}_{\neg\ell_{priv}}}(v)$ and $v(\bar{e}_{\neg\ell_{priv}})$). Therefore, there exists a valuation such that $\mathcal{A}$ if fully opaque w.r.t. ℓ_{priv} on the way to ℓ_f iff there exists a parameter valuation v such that $S_{\bar{e}_{\ell_{priv}}}(v) = S_{\bar{e}_{\neg\ell_{priv}}}(v)$, establishing the reduction.

- We now wish to show that the LpSl equality problem reduces to the FOE problem.

To do so, we fix two LpSl sets S_1 and S_2, then build two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $S_i(v)$ contains exactly the integers that satisfy $v(PET(\mathcal{A}_i))$, for all valuation v, for $i \in \{1, 2\}$.
Let us focus on S_1 and assume it is of the form given by Equation (2). We build $\mathcal{A}_1$ so that from the initial location ℓ_0 it can take multiple transitions (one for each $i \in I$), the ith transition being allowed if the clock lies between b_0^i and c_0^i, reset the clock and reach a state ℓ_i. From ℓ_i, there are n_i loops, and the jth loop can be taken if the clock lies between b_j^i and c_j^i and resets the clock. Moreover, a transition can be taken from ℓ_i to ℓ_f if $x = 0$.
Formally, $\mathcal{A}_1 = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, I, E)$ where $\Sigma = \{\epsilon\}$, $L = \{\ell_0, \ell_f\} \cup \{\ell_i \mid i \in I\}$, $\mathbb{X} = \{x\}$, $\mathbb{P}$ is the set of parameters appearing in S_1, I does not restrict the PTA (i.e., it associates $\mathbb{R}_{\geq 0}$ to every location), and finally

$$\begin{aligned} E =& \big\{(\ell_0, (b_0^i \leq x \leq c_0^i), \epsilon, \{x\}, \ell_i \mid i \in I\big\} \\ & \cup \big\{(\ell_i, (b_j^i \leq x \leq c_j^i), \epsilon, \{x\}, \ell_i \mid i \in I, 1 \leq j \leq n_i\big\} \\ & \cup \big\{(\ell_i, (x = 0), \epsilon, \emptyset, \ell_f \mid i \in I\big\}. \end{aligned}$$

Thus, a run reaching ℓ_f can be decomposed into final-reset paths. In other words, there is a run reaching ℓ_f with duration d iff d can be written as a sum $d = \sum_{j=0}^{n_i} d_j$ where $b_0^i \leq d_0 \leq c_0^i$ and for all $j > 0$, $k_j b_j^i \leq d_j \leq k_j c_j^i$ where k_j is the number of times the jth loop is taken in the PTA. As a consequence, the set of durations of runs reaching ℓ_f is exactly S_1.
We build $\mathcal{A}_2$ similarly. We now build the PTA $\mathcal{A}$ which can either immediately (with $x = 0$) go to the initial state of $\mathcal{A}_1$ or go immediately to a private location ℓ_{priv} before immediately reaching the initial state of $\mathcal{A}_2$. The final location of $\mathcal{A}_1$ and $\mathcal{A}_2$ are then fused in a single location ℓ_f. We thus have that, the set of runs reaching ℓ_{priv} on the way to ℓ_f are exactly the ones reaching ℓ_f in $\mathcal{A}_2$ (with a prefix of duration 0). And similarly, the set of runs avoiding ℓ_{priv} on the way to ℓ_f are exactly the ones reaching ℓ_f in $\mathcal{A}_1$ (with a prefix of duration 0). Therefore, for any parameter valuation v, we have that $DVisit^{priv}(v(\mathcal{A})) = D\overline{Visit^{priv}}(v(\mathcal{A}))$ iff $S_1(v) = S_2(v)$, concluding the reduction. ◀

The Parallel Dynamic Complexity of the Abelian Cayley Group Membership Problem

V. Arvind
The Institute of Mathematical Sciences (HBNI), Chennai, India
Chennai Mathematical Institue, India

Samir Datta
Chennai Mathematical Institute and UMI ReLaX, India

Asif Khan
Chennai Mathematical Institute, India

Shivdutt Sharma
Indian Institute of Information Technology, Una, India

Yadu Vasudev
Indian Institute of Technology Madras, Chennai, India

Shankar Ram Vasudevan
Chennai Mathematical Institute, India

Abstract

Let G be a finite group given as input by its multiplication table. For a subset $S \subseteq G$ and an element $g \in G$ the *Cayley Group Membership Problem* (CGM) is to check if g belongs to the subgroup generated by S. While this problem is easily seen to be in polynomial time, pinpointing its parallel complexity has been of research interest over the years. Barrington et al [6] have shown that for abelian groups the CGM problem can be solved in $O(\log \log |G|)$ parallel time. In this paper we further explore the parallel complexity of the abelian CGM problem, with focus on the dynamic setting: the generating set S changes with insertions and deletions and the goal is to maintain a data structure that supports efficient membership queries to the subgroup $\langle S \rangle$. Though the static version of the CGM problem can be easily reduced to digraph reachability, the reduction does not carry over to the dynamic setting. We obtain the following results:

1. First, we consider the more general problem of Monoid Membership, where G is a monoid input by its multiplication table. When G is a *commutative monoid* we show there is a deterministic dynamic AC^0 algorithm[1] for membership testing that supports $O(1)$ insertions and deletions in each step.
2. Building on the previous result we show that there is a dynamic randomized AC^0 algorithm for abelian CGM that supports $\mathsf{polylog}(|G|)$ insertions/deletions to S in each step.
3. If the number of insertions/deletions is at most $O(\log n / \log \log n)$ then we obtain a deterministic dynamic AC^0 algorithm for abelian CGM.
4. Applying these algorithms we obtain analogous results for the dynamic abelian Group Isomorphism.

We can also handle sub-linearly many changes to the multiplication table for G, utilizing the hamming distance between multiplication tables of any two distinct groups.

2012 ACM Subject Classification Theory of computation → Parallel algorithms

Keywords and phrases Dynamic Complexity, Group Theory, Cayley Group Membership, Abelian Group Isomorphism

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.4

Related Version *Full Version*: https://arxiv.org/abs/2308.10073

Funding *Samir Datta*: Partially supported by a grant from Infosys foundation.
Asif Khan: Partially supported by a grant from Infosys foundation.

[1] Equivalently, a constant time parallel algorithm using polynomially many processors.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 4; pp. 4:1–4:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The main algorithmic problem of interest in this paper, is the *Cayley Group Membership Problem* (CGM): Given as input a finite group G by its multiplication table (also known as its Cayley table), a subset $S \subseteq G$ and an element $g \in G$, test if $g \in \langle S \rangle$, where $\langle S \rangle$ is the subgroup of G generated by the elements in S.

The CGM problem was brought into focus by the work of Barrington et al [6] which raises intriguing questions about its parallel complexity.

Background. Membership testing in finite groups is well-studied [28]. Its computational complexity significantly depends on how G is given as input and on its elements' representation. For example, if the elements of G are represented as permutations on $[n] = \{1, 2, \ldots, n\}$, then G is a subgroup of S_n, the group of all permutations on $[n]$. A natural compact description of G as input is by a generating set, as every finite group G has a generating set of size at most $\log |G|$. In this form, membership testing in *permutation groups* has been studied since the 1970's, pioneered by the work of Sims [29, 28]. There are efficient polynomial (in n) time algorithms for the problem as well as parallel algorithms for it. The problem is in NC: it can be be solved in $\mathsf{polylog}(n)$ time with polynomially many processors [4]. On the other hand, $G = \langle a \rangle$ could be a cyclic subgroup, generated by a, of $\mathbb{F}_p^*$, the multiplicative group of the finite field $\mathbb{F}_p$, where the prime p is given in binary. Testing if $b \in \langle a \rangle$, for $b \in \mathbb{F}_p^*$ is considered computationally hard. The search version of solving for x such that $a^x = b$ is the *discrete log* problem, widely believed intractable for random primes p.

Cayley Table Representation. The Cayley table representation of G, in contrast, makes the CGM problem algorithmically easy: we can define a graph $X = (V, E)$ with $V = G$ as vertex set and $(x, y) \in E$ if $xs = y$ or $ys = x$ for a generator $s \in S$. Then, g is in the subgroup generated by S if and only if the vertex g is reachable from the identity element of G. Indeed, this is an instance of undirected graph reachability which has a polynomial time and even a *deterministic logspace* algorithm due to Reingold [27]. That is, CGM is in the complexity class L (which is contained in P). Since it is in L, it is also in the circuit complexity class $\mathsf{AC}(\log n) = \mathsf{AC}^1$, which means it has log-depth polynomial-size circuits of unbounded fanin. Equivalently, this means CGM has a logarithmic time CRCW PRAM algorithm (we will define the relevant parallel complexity classes in Section 2). Henceforth, we will assume the groups to be given by their multiplication table. In this setting, linear time algorithm for abelian CGM is also known[22].

Parallel Complexity of CGM and Group Isomorphism. Chattopadhyay, Torán and Wagner [7] have shown that the Group Isomorphism problem of checking if two groups G_1 and G_2 given as input by their multiplication tables are isomorphic can be solved by quasipolynomial size constant-depth circuits. While the question whether or not Group Isomorphism is in P is open and is intensely studied in recent times [30, 15, 14], the above parallel complexity upper bound implies that even Parity is not reducible to Group Isomorphism! Similarly, Fleischer has observed, based on [7] that the CGM problem can also be solved by quasipolynomial size constant depth circuits. Since there is no hardness result for CGM, pinpointing its parallel complexity is an interesting question.

As already mentioned, Barrington et al [6] have made nice progress showing that CGM for abelian groups is in $\mathsf{AC}(\log \log n)$. Indeed, since the resulting circuits are dlogtime uniform, the upper bound is FOLL (which means first-order definable with $\log \log n$ quantifier depth, where n is the size of the group). Further, they also show that CGM for nilpotent groups is in the class $\mathsf{AC}((\log\log n)^2)$ and CGM for solvable groups of class d are in $\mathsf{AC}(d\log\log n)$. The interesting questions in the static setting is to improve these upper bounds and/or extend these results to other classes of groups.

In this paper we study the *dynamic parallel complexity* of CGM for abelian groups. Before we describe our results, we give some background.

Dynamic complexity. Dynamic algorithms, broadly, deals with the design of efficient algorithms for problems when the input is modified with small changes. The aim is to solve the problem, for the modified input, significantly more efficiently than running the best known "static" algorithm from scratch. The measure of efficiency is crucial here and defines the model of computation. Dynamic algorithms is a burgeoning field of research (see e.g. [17] and [21, 11, 25]) with many applications that require handling large inputs subject to small changes over time.

From a *parallel complexity perspective*, we have the framework of Patnaik and Immerman [26] that is rooted in descriptive complexity [20]. Closely related is the work of Dong, Su, and Topor [12]. Here the ideal solution is to obtain a dynamic algorithm for the considered problem that runs in *constant parallel time*. Theoretically, constant parallel time is $O(1)$ time on a CRCW PRAM model (denoted by $\mathsf{CRCW}(1)$, where the CRCW model is the most liberal as it allows for concurrent reads and writes). It is well-known that this coincides with the complexity class AC^0 (the class of problems solvable by constant-depth boolean circuits). From a descriptive complexity perspective, when the circuits are dlogtime uniform (more details in Section 2) this corresponds to FO, the class of problems expressible in first-order logic. The dynamic complexity class DynFO [26] is the class of problems for which there exist FO update formula that have access to constantly many auxiliary relations, such that after small changes to the problem input, the formula correctly computes the output of the problem as well as updates to the auxiliary relations. These different ways of describing $O(1)$ parallel time are essentially equivalent: because FO and *uniform* AC^0 are equivalent [5]. There is renewed interest in this model of computation since a long-standing open problem, whether directed graph reachability is in DynFO, under single edge changes [26], was resolved in the affirmative [8].

In the present paper, it is more convenient to describe our results, which are essentially algorithmic and do not have a logical flavour, in terms of the parallel class $\mathsf{CRCW}(1)$ (or equivalently circuit class AC^0).

The results of this paper. In this paper, we obtain results on the dynamic parallel complexity of abelian CGM and abelian Group Isomorphism. Our motivation is to see if we can exploit the underlying group structure to give a dynamic $\mathsf{CRCW}(1)$ algorithm for the CGM membership queries while the generating set S is dynamically changing with insertions and deletions. We are able to obtain for the abelian group case the following results.

- First, we consider the more general problem of Monoid Membership, where G is a *monoid* input by its multiplication table. When G is a *commutative monoid* we give a deterministic dynamic $\mathsf{CRCW}(1)$ algorithm for membership testing that supports $O(1)$ insertions and deletions in each step. The algorithm requires a one-time SAC^1 preprocessing step. The main idea is to maintain the monoid M in a tree-like data structure. The cyclic monoids are at the leaves of the tree and each internal node has the submonoid of M generated by set of all its descendant leaves. Furthermore, each internal node will also hold submonoids corresponding to deletions of its descendant nodes.

- We can use this tree-like data structure more powerfully in the case of abelian groups to obtain a randomized dynamic $\mathsf{CRCW}(1)$ algorithm for abelian CGM that supports $\mathsf{polylog}(|G|)$ insertions/deletions to S in each step. The algorithm needs a one-time $\mathsf{CRCW}(\log n)$ ($O(\log n)$ time algorithm with polynomially many parallel processors) preprocessing step. The main fact that we exploit here is that adding $\mathsf{polylog}(n)$ many unary numbers can be done in $\mathsf{CRCW}(1)$. Thus, from an abelian subgroup H given by $\mathsf{polylog}(n)$ many generators we can randomly sample from H and hence list out all of H with high probability in $\mathsf{CRCW}(1)$.
- If the number of insertions/deletions is at most $O(\log n/\log\log n)$ then we obtain a deterministic dynamic $\mathsf{CRCW}(1)$ algorithm for abelian CGM which needs a one-time $\mathsf{CRCW}(\log n)$ preprocessing step. Here our techniques are linear algebra based: we need to consider some *miniature* linear algebra problems where the number of variables is $O(\log n)$ and we adapt existing linear algebraic techniques to solve this.
- We obtain analogous results for the dynamic abelian Group Isomorphism. We also consider sublinearly many modifications to the Cayley table as well in all the cases.

The techniques used. The main dynamic complexity technique used is the idea of *muddling* [9]. Imagine that for a dynamic problem we have an $\mathsf{CRCW}(1)$ algorithm $\mathcal{A}$ that after each small changes answers queries and updates the auxiliary data structures in $O(1)$ time. And it uses a AC^1 preprocessing step at the beginning to setup the auxiliary data structures. However with successive updates, the auxiliary data structures gradually deteriorate, to the point that after $\log n$ change steps the algorithm can no longer answer the queries and the auxiliary data structures are rendered ineffective. The muddling technique implies that such a problem is in fact in dynamic $\mathsf{CRCW}(1)$, meaning that there exists an $\mathsf{CRCW}(1)$ algorithm that answers the queries after each small change, and updates the auxiliary data structures, and does so for arbitrarily long sequence of changes. This broad technique is applied in this paper to the problems considered and we will have occasion to see the muddling technique in detail.

In addition, we crucially use the structure of finite abelian groups, and some tree-like data structures. We are also able to use reductions of the CGM problem to some linear algebra problems, such that in the dynamic setting we can utilize efficient matrix inverse and determinant updates under small rank changes to the matrices.

Organization. In Section 2 we give some basic definitions and notation, and some background about the dynamic parallel complexity model. In Section 3 we explain the dynamic $\mathsf{CRCW}(1)$ algorithm for commutative monoid membership under single insertions/deletions to the generating set. Sections 4 and 5 contain, respectively, the randomized and deterministic dynamic $\mathsf{CRCW}(1)$ algorithms for abelian CGM. In Section 6 we apply the CGM results to obtain dynamic $\mathsf{CRCW}(1)$ algorithms for Abelian Group Isomorphism. Finally, in Section 7 we discuss dealing with small changes to the group multiplication table itself. In the interest of space, proofs are pushed to the Appendix. Lemma statements in the main are hyperlinked to their proofs in the Appendix.

2 Preliminaries

Group Theory. A *monoid* $(M,\cdot)$ is a set M equipped with a binary operation $\cdot$, that is associative and has an identity element. $(N,\cdot)$ is called a submonoid of $(M,\cdot)$ if $N\subseteq M$ containing the identity element and is closed under the binary operation $\cdot$. A monoid whose binary operation is commutative is called a *commutative monoid.* A monoid is called a *group* if all its elements have inverses, i.e., for each monoid element there exists another element such that their product gives the identity. A commutative group is called an *Abelian Group*.

Complexity Classes. We will mainly consider parallel complexity classes defined by boolean circuits. Let $\mathsf{AC}(t(n))$ denote the class of decision problems that have polynomial-size circuits of *depth* $t(n)$ for inputs of size n, where the AND and OR gates of the circuit are allowed to be unbounded fanin. This circuit model is essentially equivalent to $t(n)$ parallel time on a CRCW PRAM model (with polynomially many processors, denoted by $\mathsf{CRCW}(t(n))$), where CRCW allows for concurrent reads and writes to a memory location. More details of these connections can be found in [20]. In particular, $\mathsf{AC}(1)$ is usually denoted AC^0 and AC^1 denotes $\mathsf{AC}(\log n)$. The class $\mathsf{AC}(\log\log n)$ is of interest in this paper due to the result of Barrington et al [6] showing that abelian CGM is in uniform $\mathsf{AC}(\log\log n)$. This class is also denoted FOLL in [6] (for first-order formulas with $\log\log n$ depth quantifiers for size n inputs). We use both notations interchangeably. If we restrict the circuits to be monotone (negation allowed at the input gates) and fanin of AND gates to be constant in the above circuit families, then the corresponding complexity classes are denoted by $\mathsf{SAC}(t(n))$. For example, SAC^0 and SAC^1 circuit families are same as AC^0 and AC^1 respectively, except that the AND gates can have only a constant size fanin and the negations are allowed only at the input gates [31].

An $\mathsf{AC}((t(n))$ algorithm will actually be given by a family of circuits $\{C_n\}_{n>0}$, where C_n solves the problem for inputs of length n, has depth $t(n)$, and size bounded by some polynomial n^c for a constant $c > 0$. We need a *uniformity condition* that tells us how efficiently we can construct the circuits C_n. A stringent condition is the so-called *dlogtime uniformity*: Each gate in the circuit C_n can be described using $O(\log n)$ bits and the uniformity condition requires that the gate connections can be checked in deterministic time linear in $O(\log n)$ by a random access machine [5]. The class dlogtime uniform AC^0 coincides with FO, where the structures on which the formulas are evaluated are equipped with some suitable predicates [5].

The parallel dynamic algorithms in this paper are describable by circuits that are dlogtime uniform.

The parallel dynamic complexity model. We briefly explain the parallel dynamic complexity model. The underlying model for describing the algorithms can seen as a CRCW PRAM. That means the algorithm can use polynomially many parallel processors accessing a shared memory that allows concurrent reads and concurrent writes with well defined notion of which write succeeds. We can also give a circuit complexity description for the model.

1. For each problem there is a well-defined notion of small changes to the input.
2. In the CRCW PRAM setting, the algorithm uses n^c processors for length n inputs for some constant $c > 0$ that depends on the problem.
3. At each time instant the algorithm receives as input $i(n)$ small changes to the problem input. With respect to the problem input at time instant t, the algorithm is required to output the answer in constant time. I.e., within time instant $t + O(1)$.
4. In the boolean circuit setting, for inputs of length n the model can be seen as a layered boolean circuit that is of width n^c for length n inputs, where the layers denote the time instants. At each layer it receives as input the $i(n)$ changes. For the input at layer t it needs to output the answer before layer $t + O(1)$.

5. We use DynAC^0 to broadly denote the class of problems that have dynamic algorithms that take $O(1)$ time with polynomially many processors[2]. We explicitly state the number of small changes to the input that can be handled at each time step. We also refer to such dynamic algorithms as DynAC^0 algorithms, or equivalently as dynamic $\mathsf{CRCW}(1)$ algorithms.

Depending on the problem at hand, the dynamic algorithm usually works by creating a suitable data structure from the given input which it updates with the small changes to the input.

3 A Dynamic CGM Algorithm for Commutative Monoids

In this section, we consider the more general problem of *Cayley Monoid Membership* for commutative monoids: Given a commutative monoid M by its multiplication table, a subset $S \subseteq M$, and an element $m \in M$, check if m is in the submonoid $\langle S \rangle$ generated by S. By abuse of notation, we term this the CGM problem for commutative monoids.

We present a tree-based data structure to maintain the generating set S, using which we obtain a DynAC^0 algorithm that supports a single insertion/deletion to/from the subset S at each step. We will use this data structure with suitable modifications in Section 4.

The CGM problem for monoids. It is known that CGM for monoids is reducible to directed graph reachability. To see this just construct the Cayley digraph $\mathrm{Cay}(M, S)$ of the monoid M corresponding to generating set S. The graph has vertex set M and for every $m \in M$ and every $m' \in S$, the directed edge $(m, m \cdot m')$ is in the edge set. Clearly, an element $t \in M$ is in the submonoid $\langle S \rangle$ iff there is a directed path from the monoid identity e to t in this digraph. Hence Cayley Membership for monoids is in NL. We note here that, though we know dynamic reachability to be in DynAC^0 under constantly many changes, this reduction of the monoid membership problem to reachability doesn't directly give a DynAC^0 bound for the monoid membership problem. This is because, even after the insertion/deletion of a single element to/from S, the graph $\mathrm{Cay}(M, S)$ changes drastically, i.e., $O(n)$ many edges are affected. We only know how to handle $\mathsf{polylog}(n)$ many changes in DynAC^0 even for the undirected reachability problem.

So, we will use the weaker upper bound of SAC^1 for the CGM problem for commutative monoids. This upper bound actually gives a tree-like data structure, using which we obtain the DynAC^0 algorithm for the problem that supports single insertions/deletions to S.

As M is commutative, any element of the submonoid $\langle S \rangle$ is expressible as a product of powers of elements in S: $\prod_{s \in S} s^{e_s}$, where $0 \leqslant e_s \leqslant n$ and $n = |M|$.

We claim that the entire submonoid $\langle S \rangle$ can be listed as the output of an SAC^1 circuit. The circuit takes S as input, as a n-bit binary number with i^{th} bit indicating whether the element $m_i \in M$ is in S, and outputs an n-bit binary number whose ith bit is 1 iff $m_i \in M$ is in the submonoid $\langle S \rangle$.

Let $S_1, S_2 \subseteq M$ such that the monoid identity 1 is in both S_1 and S_2. Their product is defined as $S_1 \cdot S_2 = \{a \cdot b \mid a \in S_1, b \in S_2\}$.

▶ **Proposition 1.** *Given as input the subsets S_1 and S_2 of a monoid M their product $S_1 \cdot S_2$ can be computed in* SAC^0.

[2] This coincides with DynFO [26] when the dlogtime uniformity conditions are met.

For each $a \in S$ let $P_a = \{a^i : 1 \leqslant i \leqslant n\}$. Notice that P_a can be computed directly from the multiplication table for M in L which is contained in SAC^1.

The tree-like data structure for S. We create a balanced binary tree T with leaves labelled by the distinct elements $a \in S$. Let the root of T be ρ. To the leaf labelled a, we associate the submonoid P_a. With each internal node τ, we associate a commutative submonoid M_τ of M, inductively defined as the product $M_\tau = M_\mu \cdot M_\nu$, where μ and ν are its two children. Since $|M| = n$, the tree T has depth bounded by $\log n$. By Proposition 1 the tree T can be created by an SAC^1 circuit. Moreover, for each internal node of T, the following is immediate.

▶ **Proposition 2.** *For each node τ of the tree T the subset M_τ associated with τ is the commutative submonoid generated by the subset $L_\tau = \{a \in S \mid a$ such that τ has the leaf labelled a as descendant$\}$. I.e., $M_\tau = \langle L_\tau \rangle$.*

It is clear that the subset associated with root ρ of the tree T, is actually $\langle S \rangle$. Thus, we have:

▶ **Lemma 3.** *The CGM problem for commutative monoids is in SAC^1.*

The Dynamic Setting. We will modify the above construction to obtain a dynamic data structure. First we expand the tree T, by including leaves for every element of M. For each element $a \in M$, we pre-compute its power set P_a as already defined. However, we will make P_a to be the submonoid associated with the leaf labelled a, $M_a = P_a$, precisely if $a \in S$, and otherwise we associate the identity element 1, i.e., $M_a = \{1\}$.

We will need to dynamically maintain the following data at each node of the expanded tree. Only the data associated with the tree nodes change with the changing generating set S, while the tree structure itself remains unchanged.

1. For each node ν of the tree we have the submonoid M_ν generated by the subsets associated with the leaves below node ν in the tree.
2. Additionally, at each node ν we will maintain the submonoid $M_{\nu:\mu}$ for each descendant μ of ν in the tree, where $M_{\nu:\mu}$ denotes the submonoid generated by the subsets associated with all leaves that are descendants of ν *but are not* descendants of μ. Equivalently, it is as if the submonoid associated with node μ is reset to $\{1\}$ and then the rest of the submonoid M_ν is computed.

Essentially, as in Propositions 1 and 2, given a subset S we can construct the tree data structure along with the data at each node as described above.

▶ **Proposition 4.** *Given as input a subset $S \subseteq M$ for a commutative monoid M (given by its multiplication table), we can construct the tree data structure T along with the data at each node as described above in SAC^1. Furthermore, given a membership query $m \in M$, testing if m is in the current submonoid $\langle S \rangle$ can be done in AC^0.*

Handling single insertions and deletions. Next we show that the above data structure supports single insertions and deletions to S at each time step. The updates to the data structure can be carried out in AC^0 as described below.

First we consider deletions. Suppose $a \in S$ is deleted. Then we need to update for each tree node ν, the submonoid M_ν associated with it and the submonoids $M_{\nu:\mu}$ for each of its descendant μ. Clearly, M_ν remains unchanged if a is not a descendant of ν. Similarly, $M_{\nu:\mu}$ also remains unchanged if a is a descendant of μ. In general, the required updates are explicated by the Algorithm 1 (which contains the description of both delete and insert updates colour coded as red and blue respectively). Similarly, in the case of insertion of an element $a \in M$ to S, we need to update the submonoids associated with the tree nodes.

Algorithm 1 **Delete**(a). **Insert**(a).

```
1 $M_a \leftarrow \{1\}$                                              $M_a \leftarrow P_a$
2 for $\nu \in V(T)$ such that $a \in descendants(\nu)$ do in parallel
3     $M_\nu \leftarrow M_{\nu:a}$                                    $M_\nu \leftarrow M_\nu \cdot P_a$
4 for $\mu, \nu \in V(T)$ such that $\mu$ and $a$ are descendants of $\nu$ do in parallel
5     $\mu' \leftarrow \mathrm{LCA}(\mu, a)$
6     if $\mu' \neq \mu$ then
7         Let $\mu_1, \mu_2$ be the children of $\mu'$ such that $a$ and $\mu$ are descendants of $\mu_1$ and
          $\mu_2$ respectively
8         $M_{\nu:\mu} \leftarrow (M_{\nu:\mu'}) \cdot (M_{\mu_1:a}) \cdot (M_{\mu_2:\mu})$    $M_{\nu:\mu} \leftarrow (M_{\nu:\mu'}) \cdot (M_{\mu_1} \cdot P_a) \cdot (M_{\mu_2:\mu})$
```

As each node in the tree can be processed in parallel and each requires just a product of a constant number of pre-computed submonoids (lines 3 and 8), we have the following

▶ **Proposition 5.** *The above deletion and insertion operations can be carried out in* AC^0.

The correctness of the update procedures in Algorithm 1 follows by induction on the validity of the tree data structure. The inductive hypothesis being that the data structure contains valid information about the various submonoids at the time step just before the insert/delete updates. To summarize the above results, we have the main theorem of this section.

▶ **Theorem 6.** *Let M be a commutative monoid given by its multiplication table and $S \subseteq M$ generate a submonoid $\langle S \rangle$ of M. Then there is a deterministic* DynAC^0 *algorithm that answers membership queries $m \in \langle S \rangle$ given $m \in M$ and supports single insertions and deletions to the generating set S at each time step, which requires a one-time* SAC^1 *preprocessing step for initialization of auxiliary data structures.*

4 The Dynamic CGM Problem for Abelian Groups

Let G be an n-element abelian group given as input by its multiplication table. Let S be a subset of G. We want to maintain a structure that supports efficient membership testing in the subgroup $H = \langle S \rangle$ generated by S. That means efficiently supporting the following operations.

1. Given a query element $g \in G$ test if $g \in H = \langle S \rangle$ and, if so, express g as a product of the generators in S.
2. The dynamic version of the problem requires that we efficiently support insertions/deletions to set S. We have seen how to handle, even in the setting of commutative monoids, single insertions or deletions to S at each step. More generally, we would like to handle bulk insertions and deletions at each step.

Preprocessing for dynamic abelian CGM. We will first obtain a generating set of size at most $\log n$ for G using which we can represent all elements of G.

An *independent* generating set [16, Section 3.2] for an abelian group G is a generating set $\{g_1, g_2, \ldots, g_\ell\}$ such that $g_1^{e_1} \cdot g_2^{e_2} \cdots g_\ell^{e_\ell} = 1$ for $0 \leqslant e_i \leqslant o(g_i) - 1$ if and only if $e_i = 0$, where $o(g_i)$ is the order of g_i for each i. As a consequence of independence, every element $g \in G$ is *uniquely expressible* as $g = g_1^{e_1} \cdot g_2^{e_2} \cdots g_\ell^{e_\ell}$ for $0 \leqslant e_i \leqslant o(g_i) - 1$. It is easy to see that $\ell \leqslant \log |G|$. The next proposition easily follows from [16, Theorem 3.2.2].

▶ **Proposition 7.** *Given as input a finite abelian group G by its multiplication table, an independent generating set for G (of size at most $\log n$) can be computed in $\mathsf{AC}(\log n)$.*

Thus, in a one-time preprocessing step, we can compute an independent generating set $\{g_1, g_2, \ldots, g_\ell\}$ from the multiplication table of the input group G as well as the unique expression for each $g \in G$ as $g = g_1^{e_1} \cdot g_2^{e_2} \cdots g_\ell^{e_\ell}$ for $0 \leqslant e_i \leqslant o(g_i) - 1$. The two preprocessing steps for G are summarized below.

1. For each pair $(g, i), g \in G, 0 \leqslant i \leqslant n - 1$, we compute and store the power g^i in an $n \times n$ table. This can be done in a straightforward $\mathsf{AC}(\log n)$ computation, since the product of two elements in the table is computable in AC^0. In particular, this computation also yields the order $o(g)$ of each $g \in G$.
2. By Proposition 7, in $\mathsf{AC}(\log n)$ we compute for G an independent generating set $T = \{g_1, g_2, \ldots, g_t\}$, $t \leqslant \log n$ and also a representation for each $g \in G$ as a product $g = \prod_{i=1}^{t} g_i^{e_i}$.

Additionally, we note the easy consequence of Barrington et al's FOLL algorithm [6] for abelian CGM.

▶ **Lemma 8.** *Let $S \subseteq G$, for an abelian group G given by its multiplication table as input. In FOLL a $\log |G|$ size subset T of S can be computed that generates the same subgroup as S.*

Randomized DynAC^0 Algorithm for Abelian CGM

We now present a randomized DynAC^0 algorithm for maintaining the subgroup $H = \langle S \rangle$ of G, given by its generating set. More precisely, the algorithm can process polylog(n) insertions and deletions per step and answer membership queries to the group $\langle S \rangle$ in $O(1)$ parallel time[3]. The following is a crucial lemma that is used by the query algorithm in this section.

▶ **Lemma 9.** *Let $T \subseteq G$ be of size at most $\log^c n$ for some constant c, where G is an abelian group, $|G| = n$, given by its multiplication table with independent generating set $G = \langle g_1, g_2, \ldots, g_\ell \rangle$. Then in randomized AC^0 we can list out the subgroup $\langle T \rangle$ generated by T. In particular, membership testing in the subgroup generated by T can be done in randomized AC^0.*

Given Lemma 9, we can focus on just maintaining a polylog(n) size generating set for the subgroup $\langle S \rangle$ while it undergoes polylog(n) many insertions/deletions in a step.

Auxiliary data structure. First we will create a variant of the tree-based data structure described in Section 3 to represent the generating set S and certain subgroups of $\langle S \rangle$. We will continually update this data structure, as we process the bulk insertions and deletions that occur at each time step.

[3] Updates to the auxiliary data structures are in deterministic AC^0, while queries require randomization.

1. Let S be the current generating set and $2^{k-1} < |S| \leqslant 2^k$, for positive integer $k \leqslant \lceil \log n \rceil$. To begin with, we create an $O(\log n)$ depth *full binary tree* B, with 2^k leaves. We associate, for each generator $x \in S$, the cyclic subgroup $\langle x \rangle$ with a distinct leaf of B. The remaining leaves are associated with the trivial subgroup $\{1\}$. As it is a full binary tree, its nodes can be indexed by $1, 2, \ldots, 2^{k+1} - 1$ with 1 as index for the root. For each $i > 1$, the node indexed i has as parent the node indexed $\lfloor i/2 \rfloor$.
2. Let τ be an internal node of the tree B with children μ and ν. Inductively, to each internal node τ we associate the subgroup, $H_\tau = H_\mu \cdot H_\nu$ which is the product of the subgroups H_μ and H_ν that are associated with the two children. Notice that the product $H_\mu \cdot H_\nu$ is indeed a subgroup of G as G is abelian. Letting S_τ denote the set of leaves in S below node τ. Then, notice that $H_\tau = \langle S_\tau \rangle$ for each node τ of the tree. The root is labeled with $H = \langle S \rangle$.
3. Additionally, for each internal node τ and for each *descendant* μ of τ we keep the subgroup generated by $S_\tau \setminus S_\mu$, which we denote by $H_{\tau:\mu}$. Thus, at each node τ we have the list of subgroups $H_{\tau:\mu}$ with generating set $S_\tau \setminus S_\mu$, one for each descendant μ of τ.
4. Finally, using Lemma 8 we compute, in FOLL, $\log n$ size generating sets $T_{\tau\mu} \subseteq S_\tau \setminus S_\mu$ for each subgroup $H_{\tau:\mu}$ at each node τ in parallel. Similarly, for subgroup H_τ associated with each internal node τ, a $\log n$ size generating set $T_\tau \subseteq S_\tau$ is computed.
5. For access to the data maintained at each node in the tree, we will have an array of pointers indexed by $1, \ldots, 2^{k+1} - 1$. Furthermore, we will keep a boolean array $A[i,j], 1 \leqslant i, j \leqslant 2^{k+1} - 1$ where $A[i,j] = 1$ if and only if i is an ancestor of j.

The following lemma is immediate.

▶ **Lemma 10.** *The tree data structure B, for the generating set S can be built in $O(\log n)$ parallel time (i.e. in $\mathsf{AC}(\log n)$).*

The tree data structure B useful in Section 5 as well.

Handling bulk insertions and deletions. At any point of time during the computation, the current generating set S, is maintained as a data structure described above. Then changes to the generating set arrive in the form of two sets I and D for insertions and deletions respectively, such that $|I|, |D| = O(\log^{c+1} n)$. The actual generating set then becomes $S \cup I \setminus D$. We will first show that a membership query occurring at this point of time can be answered in $O(1)$ parallel time.

▶ **Lemma 11.** *Given the data structure for S along with the update sets of insertions I and deletions D, each of $\log^{c+1} n$ size, we can test if some group element g is in the subgroup generated by $(S \cup I) \setminus D$ in AC^0.*

Continual rebuilding of the tree data structure B. This is the crucial part of the dynamic algorithm[4]. Let's assume that at time instant t, we have the tree data structure for the current generating set, denoted by $S^{(t)}$, available (this assumption is particularly valid at the starting time instant because of the preprocessing). At this instant, the new insertion and deletion bulk updates, denoted by $I^{(t)}$ and $D^{(t)}$ respectively, arrive. Using Lemma 11, we can answer membership queries about the subgroup generated by $S^{(t+1)} = (S^{(t)} \cup I^{(t)}) \setminus D^{(t)}$ in $O(1)$

[4] This continual rebuilding of data structure is a modified adaptation of the muddling technique from [9].

time. In fact, we can do this for any time instant $t+i$ $(1 \leqslant i < \log n)$, by using Lemma 11, keeping the generating set S to be $S^{(t)}$ and I and D to be the insertions and deletions respectively accumulated so far from time instant t to $t+i$, since I and D are still bounded by $\mathsf{polylog}(n)$ $(|\cup_{j=1}^{i} I^{(t+j)}|, |\cup_{j=1}^{i} D^{(t+j)}| \leqslant i \log^c n < \log^{c+1} n$ since $i < \log n)$. But, we can't keep doing this forever, as eventually the accumulated I and D grow beyond $\mathsf{polylog}(n)$. In a sense, the tree data structure is only useful for $\log n$ rounds. To remedy this, we will start AC^1 computation for building the tree data structure for the subgroup generated by $(S^{(t)} \cup I^{(t)}) \setminus D^{(t)}$ at the time instant $t+1$ using Lemma 10 and the result of this computation will be available at time instant $t + \log n$. After this point we can continue to use Lemma 11 for another $\log n$ rounds to answer the CGM queries. In fact, at each time instant $t+i$ we will have an AC^1 computation thread started up for building the tree data structure for the subgroup generated by $S^{(t+i)}$. At any time instant there are at most $\log n$ such thread running and hence the number of gates in any layer remains polynomially bounded.

To summarize, we have shown the following theorem.

▶ **Theorem 12.** *There is a randomized* DynAC^0 *algorithm for the abelian Cayley Group Membership problem,* CGM*, that requires a one-time* AC^1 *preprocessing step, and supports* $\mathsf{polylog}(n)$ *insertions and deletions to the generating set.*

5 A Deterministic Dynamic Algorithm for Abelian CGM

We now present a *deterministic* DynAC^0 algorithm for abelian CGM that can process bulk insertions/deletions of size $t = O(\frac{\log n}{\log\log n})$. The algorithm is linear algebraic. We first observe a property of abelian groups in terms of prime factorization of their order. Let G be an n-element abelian group given by its multiplication table. Let $n = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_\mu^{a_\mu}$ be its prime factorization, where p_i are distinct primes. By the structure of finite abelian groups [16, Theorem 3.3.1] $G = G_1 \times G_2 \times \cdots G_\mu$ is a direct product where G_i is the p_i-Sylow subgroup of G.[5]

For a subset $S \subseteq G$ consider the subgroup $H = \langle S \rangle$ generated by S. Let $b_i = n/p_i^{a_i}, 1 \leqslant i \leqslant \mu$ and $S_i = \{x^{b_i} \mid x \in S\},\ \ 1 \leqslant i \leqslant \mu$. Then $H_i = \langle S_i \rangle$ is a subgroup of G_i for each i and $H = H_1 \times H_2 \times \cdots \times H_\mu$.

An element $g \in G$ is in the subgroup H if and only if $g^{b_i} \in H_i$ for each i. As a consequence we can reduce the dynamic abelian CGM problem to the dynamic abelian CGM problem for abelian p-groups. We state this as a lemma.

▶ **Lemma 13.** *Given an n-element abelian group G by its Cayley table, by a one-time preprocessing computation in* AC^1 *we can compute Cayley tables for each Sylow subgroup G_i. Furthermore, all powers of elements of G can be pre-computed and stored in an array. Hence the parallel dynamic complexity of abelian* CGM *maintaining S, supporting say $t(n)$ insertions/deletions at each step is* AC^0 *reducible to the same problem for abelian p-groups.*

Now, we can focus on the CGM problem restricted to abelian p-groups only.

Reduction to integer linear equations. Using the Proposition 7 we can pre-compute an independent generating set $\{g_1, g_2, \ldots, g_\ell\}$ for the abelian p-group G. Let $|G| = n = p^m$ and $o(g_i) = p^{m_i}$ for each i, where $m_1 + m_2 + \cdots + m_\ell = m$, and $\ell \leqslant \log n$. We also

[5] Let G be a finite group of order $n = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_\mu^{a_\mu}$. Then for each i, G has at least one subgroup of order $p_i^{a_i}$, which is known as a p_i-Sylow subgroup of G. However, if G is abelian then there is a unique p_i-Sylow subgroup of G which we can denote by G_i. In that case, $G = G_1 \times G_2 \times \cdots G_\mu$.

have $m = \log_p n \leqslant \log n$. Each $g \in G$ has a unique representation (pre-computed) as $g = \prod_{i=1}^{\ell} g_i^{b_i}, \quad 0 \leqslant b_i \leqslant p^{m_i} - 1$. Thus g can also be represented as an ℓ-dimensional integer column vector $\bar{b} = [b_1 b_2 \dots b_\ell]^T$.

We will dynamically maintain a logarithmically bounded subset T of the generating set S such that T generate the same group as S, i.e., $|T| = O(\log n)$ and $\langle S \rangle = \langle T \rangle$. As explained, we will represent elements of T by ℓ-dimensional column vectors. Thus, $g \in \langle T \rangle$ iff the system of integer linear equations $Ax = \bar{b}$ is feasible, where the matrix A has columns corresponding to each generator in T, and the i^{th} row of the system of equations is computed modulo p^{m_i} for $1 \leqslant i \leqslant \ell$. We can suitably scale each equation to get a system of integer linear equations modulo p^m. Since p^m is a composite for $m > 1$, the usual recipe for feasibility of linear equations based on matrix rank does not directly apply. However, we can rewrite $Ax = \bar{b} \pmod{p^m}$ equivalently as integer linear equations

$$Ax + p^m y = \bar{b} \tag{1}$$

where y is a column vector of ℓ new variables. Letting $[A|p^m I_\ell] = \tilde{A}$, and $z = [x|y]^T$ we can write this as $\tilde{A}z = \bar{b}$, noting that $\tilde{A}$ is full row rank. Thus, the CGM problem for abelian p-groups is reduced to finding integer solution to Equation (1). To test the feasibilty of such system of equations, we have the following lemma.

▶ **Lemma 14** ([3], Theorem 3.13). *For a prime p, the system $\tilde{A}z = \bar{b}$ (and hence $Ax = \bar{b} \pmod{p^m}$) is feasible (i.e., has an integer solution) iff GCD of the $\ell \times \ell$ subdeterminants of $\tilde{A}$ and the GCD of the $\ell \times \ell$ subdeterminants of the augmented matrix $[\tilde{A}|\bar{b}]$ have the same highest power of p dividing them both.*

Since the matrix A in Equation (1) has $\ell \leqslant \log n$ rows and $t \leqslant 2 \log n$ columns the number of $\ell \times \ell$ subdeterminants is polynomially bounded. So, if we could compute the determinant of each $\ell \times \ell$ submatrix in $\mathsf{AC}(\log\log n)$, we can execute the above feasibility test as well in $\mathsf{AC}(\log\log n)$, and hence solve the CGM problem. Indeed, we have the following lemma to compute determinant of such small matrices.

▶ **Lemma 15.** *Let A be a square matrix of dimension $\mathsf{polylog}(n)$ with entries that are polynomially bounded in n, then each bit of $\det(A)$ can be computed in $\mathsf{AC}(\log\log n)$.*

We will need to dynamically maintain the determinants and inverses of all the $\ell \times \ell$ submatrices of $\tilde{A} = [A|p^m I_\ell]$. The following lemma shows that, at least in the beginning, all these can be precomputed in $\mathsf{AC}(\log\log n)$, giving way for an AC^0 algorithm for CGM (part 3).

▶ **Lemma 16.** *Let $Ax = \bar{b} \pmod{p^m}$ be a system of integer linear equations modulo p^m, where $p^m = n$ is input in unary, and $A \in \mathbb{Z}^{\ell \times t}$ and $\bar{b} \in \mathbb{Z}^l$, $t = O(\log n)$.*

1. *Let $\tilde{A} = [A|p^m I_\ell]$. We can compute the determinants of all square submatrices of $\tilde{A}$ and $[\tilde{A}|\bar{b}]$ in $\mathsf{AC}(\log\log n)$ (i.e. in $\log\log n$ parallel time).*
2. *Furthermore, for the nonsingular submatrices we can also compute their inverses in $\mathsf{AC}(\log\log n)$.*
3. *Given the above we can test the feasibility of $Ax = \bar{b} \pmod{p^m}$ and solve for x in AC^0.*

This preprocessing of the matrix $\tilde{A}$ needs to be combined with a variant of the matrix inverse lemma stated below [18] (this is a variant of the so-called Sherman-Morrison-Woodbury formula) to dynamically compute solutions to $\tilde{A}z = \bar{b}$. This formula essentially allows for a quick updation of the data computed using Lemma 16 for A, if A is replaced with $A + A'$ for a small rank matrix A'.

▶ **Lemma 17** (Binomial Matrix Theorem [18]). *Let M be an invertible $r \times r$ matrix over any field (or ring). Let C, U and V be $t \times t$, $r \times t$ and $t \times r$ matrices respectively, over the same field/ring. The inverse of* $M + UCV$ *is* $M^{-1} - M^{-1}U(I + CVM^{-1}U)^{-1}CVM^{-1}$ *if* $M + UCV$ *is invertible.*

Similarly to update determinants quickly we will need the following.

▶ **Lemma 18** (Matrix Determinant Lemma). *If M is a $r \times r$ matrix over a field and U and V are $r \times t$ and $t \times r$ matrices then* $\det(M + UV^T) = \det(I_t + V^T M^{-1}U)\det(M)$.

Finally, we will also require the following lemma to compute the small matrix inverses and determinants, viz. $(I + CVM^{-1}U)^{-1}$ and $\det(I_t + V^T M^{-1}U)$ respectively, required by Lemmas 17 and 18.

▶ **Lemma 19** ([10], Theorem 8).

1. *Let $t = O(\frac{\log n}{\log\log n})$ and B be a $t \times t$ integer matrix with entries bounded by p^m and q be an $O(\log\log n)$ bit prime number. Then both $\det(B) (\bmod\, q)$ and B^{-1} over $\mathbb{F}_q$ can be computed in AC^0.*
2. *Furthermore, by Chinese remaindering, $\det(B)$ and hence B^{-1}, if it exists, can both be computed in AC^0 by applying the first part for several distinct primes q_i and different submatrices.*

We will now see how to use these in the context of processing $O(\log n/\log\log n)$ bulk insertions and deletions.

Processing Bulk Insertions. Suppose $\hat{T}$ is the set of insertions to S, where $|\hat{T}| = t = O(\log n/\log\log n)$. Thus, the system of linear equations $Ax = \bar{b} \pmod{p^m}$ is now modified to $[A|\hat{A}]x = \bar{b} \pmod{p^m}$, where the columns of A correspond to T, the new columns $\hat{A}$ correspond to the insertions $\hat{T}$, and $\bar{b}$ is the integer vector corresponding to a $g \in G$ whose membership we want to test in $\langle T \cup \hat{T}\rangle$.

We note that in the modified linear equation above, the original coefficient matrix has been modified in at most t columns. Thus, Lemmas 17 and 18 are applicable. With these we can update the data computed by Lemma 16 in AC^0. It can be recomputed in AC^0 by Lemma 19 as at most $O(\log n/\log\log n)$ columns are modified in any submatrix, thinking of the new columns as modifications of zero columns. Furthermore the recomputations involves computing the determinant and inverse of matrices of dimension at most $t = O(\log n/\log\log n)$, where those matrices have integer entries given as input in unary (because each of them is at most p^m in magnitude). A crucial difficulty in the application of Lemmas 17 and 18 is that a submatrix M of A, whose inverse/determinant we need to update, may itself not be invertible. We can deal with this by maintaining the data computed by Lemma 16 for the *invertible* matrices $\xi I - M$, for all submatrices M of A, where ξ is an indeterminate. Lemma 15 and parts 1 and 2 of Lemma 16 can be applied *mutatis mutandis* to matrices $\xi I - M$ (for the submatrices M of $\tilde{A}$). The determinant of $\xi I - M$ will be a degree r polynomial in ξ and $(\xi I - M)^{-1}$ will have entries that are rational functions $f(\xi)/g(\xi)$ where f and g are of degree at most r, where $r = O(\log n)$. Consider Lemma 17 and Lemma 18 applied to $\xi I - M$ instead of M. Notice that $\det(M + UV^T)$ is the constant term of $\det(M + UV^T - \xi I)$ which we can compute in AC^0, essentially by Lemma 19. Similarly, by Lemma 19 the inverse $(M + UCV^T)^{-1}$, if it exists, can be computed in AC^0 from Lemma 17 applied to $\xi I - M$.

Processing Bulk Deletions. Let $\hat{T}$ be the set of elements that are deleted from S. It is clear that modifications to the matrix A are required only if some elements from T are deleted, i.e., $T \cap \hat{T} \neq \emptyset$. However, deletions are bit trickier than insertions to handle. This is because, we can't simply drop the columns from A corresponding to $\hat{T} \cap T$ after bulk deletion $\hat{T}$, as it need not be the case that what remains in T is still a generating set for the $\langle S \setminus \hat{T} \rangle$. We might possibly need to include some elements from $S \setminus \hat{T}$ to $T \setminus \hat{T}$ for getting a correct small generating set for $S \setminus \hat{T}$. The number of columns to be dropped from A is clearly bounded by $O(\log n / \log \log n)$. However, with respect to number of columns that are to be included in A after the change, Lemmas 17 and 18 enable us to update the submatrices' inverse and determinant only if this number is bounded by $O(\log n / \log \log n)$. To our relief, it is indeed the case. We show the following about finite abelian p-groups in general. For a finite abelian p-group G, $|G| = p^m$, let $S \subseteq G$. Let T be a subset of S such that $\langle S \rangle = \langle T \rangle$. Without loss of generality, we assume that $|S| \geqslant 2$. Then we have the following.

▶ **Lemma 20.** *Let G be a finite abelian p-group, $|G| = p^m$, and $T \subseteq S \subseteq G$ such that $\langle T \rangle = \langle S \rangle$. Then for any $g \in T$, there is an $h \in S \setminus T$ such that, $\langle S \setminus \{g\} \rangle = \langle (T \setminus \{g\}) \cup \{h\} \rangle$.*

Proof. Let $H = \langle T \rangle$ and $K = \langle T \setminus \{g\} \rangle$. Since the groups are abelian $H = K\langle g \rangle$. Suppose $|H| = p^{m_1}$ and $|K| = p^{m_2}$. Let $\mu = m_1 - m_2$. Then $|H|/|K| = p^{\mu}$, which is the number of distinct cosets of K in H. Furthermore, as H/K is cyclic, generated by Kg, it follows that $Kg \in H/K$ is an element of order p^{μ}. Hence, we have the disjoint union $H = \sqcup_{j=0}^{p^{\mu}-1} Kg^j$.

▷ Claim 21. For $j \geqslant 0$ and α relatively prime to p, the subgroups $K\langle g^{p^j} \rangle$ and $K\langle g^{p^j \alpha} \rangle$ are identical.

To see this it suffices to note that for any finite cyclic group $\langle a \rangle$ of order d and any α relatively prime to d, a^{α} is also a generator of the cyclic group $\langle a \rangle$. Hence, the cyclic subgroups $\langle g^{p^j} \rangle$ and $\langle g^{p^j \alpha} \rangle$ are identical which proves the claim.

For each $s \in S$, let Kg^{e_s} be the coset to which it belongs. Writing $e_s = p^{\ell_s}\alpha_s$ for α_s relatively prime to p, the above claim implies the subgroups $K\langle s \rangle$ and $K\langle g^{p^{\ell_s}} \rangle$ are identical. That means the two subgroups $\langle T \cup \{s\} \setminus \{g\} \rangle$ and $\langle T \cup \{g^{p^{\ell_s}}\} \setminus \{g\} \rangle$ are identical.

Now, among elements in $S \setminus T$, let h be an element with the least ℓ_h. We claim that $\langle S \setminus \{g\} \rangle = \langle T \cup \{h\} \setminus \{g\} \rangle$.

Suppose $s \in S \setminus \{g\}$ is some other element. Then $\ell_s \geqslant \ell_h$, by the choice of h. That means $K\langle g^{p^{\ell_s}} \rangle$ is a subgroup of $K\langle g^{p^{\ell_h}} \rangle$, which implies $s \in K\langle g^{p^{\ell_h}} \rangle = K\langle h \rangle = \langle (T \setminus \{g\}) \cup \{h\} \rangle$, completing the proof. ◀

From Lemma 20, it is clear that after deletion of $\hat{T}$, there is a $O(\log n / \log \log n)$ size set $R \subseteq (S \setminus \hat{T})$ such that $\langle (T \setminus \hat{T}) \cup R \rangle = \langle S \setminus \hat{T} \rangle$. Hence, after the deletion of $\hat{T}$ from S if we can find such a set R, then we can update the matrix A by dropping the columns corresponding to the elements in $\hat{T}$ and including columns corresponding to R so that the modified matrix A in Equation (1) correctly corresponds to the CGM question with respect to the modified generating set S. For the modified matrix, we can update for each $\ell \times \ell$ submatrix its inverse and determinant in AC^0 using binomial matrix theorem and matrix determinant lemma (Lemmas 17 and 18). This is because the determinants and inverses involved in Lemmas 17 and 18 can be computed in AC^0 due to Lemma 19 as their size bounded by $O(\log n / \log \log n)$.

But, we can't afford to search for R in $S \setminus \hat{T}$, which is of $O(n)$ size. However, we can make use of the tree data structure B, of Section 4 that essentially maintains a $\mathsf{polylog}(n)$ size generating set for S. That is, given B, we always have a $\mathsf{polylog}(n)$ size subset P of S

available, such that it generates the same group as S. We can exhaustively search for a valid R in P, since the search space is polynomially bounded ($\binom{\mathsf{polylog}(n)}{O(\log n/\log\log n)} = \mathsf{poly}(n)$). For each choice of R we can use the AC^0 routine for handling $O(\log n/\log\log n)$ insertions to the generating set, to find the span $\langle (T\setminus\hat{T})\cup R\rangle$. On top of that, we can find an R' such that $\langle (T\setminus\hat{T})\cup R'\rangle$ contains all the elements that could be generated with any other choice of R. With R', we finally update $T\setminus\hat{T}$ to become $(T\setminus\hat{T})\cup R'$ and update all the relevant matrix inverses and determinants.

Continual rebuilding of data structure. We note that the data required by the algorithm to answer membership queries in AC^0 are:

- A generating set T of size $O(\log n)$ such that $\langle T\rangle = \langle S\rangle$.
- A $\mathsf{polylog}(n)$ size generating set P of $\langle S\rangle$ such that $P\subseteq S$.
- An $\ell\times|T|$ dimensional matrix A corresponding to the generating set. Determinant and inverses of all non-singular $\ell\times\ell$ submatrices of $[A|p^m I_\ell]$.

We have already seen in Section 4 that $\mathsf{polylog}(n)$ size generating set P can be maintained in DynAC^0 even under $\mathsf{polylog}(n)$ size bulk changes with help of the tree data structure, though it requires a one time AC^1 preprocessing step.

Having intialized the above auxiliary data structure, after every bulk insertion/deletion we can update them in AC^0 at least for $\log\log n$ time steps. Within this time period, the small generating set T remains logarithmically bounded, because even though after every time step, size of T could grow by $O(\log n/\log\log n)$, it can only grow by $O(\log n)$ over $\log\log n$ steps. Also, within this time period, we can check feasibility of $Ax = \bar{b} \pmod{p^m}$ in AC^0 and hence answer membership queries. We use the muddling technique to extend this $\log\log n$ window such that we can, for arbitrary long sequence change-steps, answer the CGM queries in AC^0 after each bulk insertion/deletion step.

In the beginning we have the correct small generating set $T_0 = T$, because of initialization. After the first change step, say $i = 1$, we start an $\mathsf{AC}(\log\log n)$ computation thread for computing an at most $\log n$ size subset T_1 of the modified T such that it generates the same subgroup. The result of this thread, T_1 is available after a delay of $\log\log n$ time steps, i.e., at time step $i = \log\log n + 1$. With respect to T_1 and the insertion/deletions that accumulate during the recomputation phase, we have all the updated information about the corresponding matrix A, i.e., the determinants and inverses of all $\ell\times\ell$ submatrices of A. So, when the next bulk insertion/deletion arrives, we can update the auxiliary information again in AC^0. In fact, we start an $\mathsf{AC}(\log\log n)$ recomputation thread for computing the small generating set T_i at every time step i within the $\log\log n$ time window so that we have logarithmically bounded generating set at time step $i + \log\log n$. Still the number of recomputation threads running at any given time step is $O(\log\log n)$, thus keeping the overall circuit size polynomially bounded.

To summarize we have shown the following theorem.

▶ **Theorem 22.** *There is a deterministic dynamic* AC^0 *algorithm for the abelian Cayley Group Membership problem,* CGM*, that supports* $O(\log n/\log\log n)$ *insertions and deletions to the generating set, and requires a one-time* AC^1 *preprocessing step.*

6 Dynamic Abelian group isomorphism

Let G_1 and G_2 be abelian groups, each given a multiplication table as input, say T_1 and T_2, respectively. Let $S_1\subseteq G_1$ and $S_2\subseteq G_2$ be subsets. In the static setting, there is a simple polynomial time algorithm for checking if $\langle S_1\rangle$ and $\langle S_2\rangle$ are isomorphic: it suffices to list out the two subgroups $\langle S_1 \rangle$ and $\langle S_2 \rangle$, check they have the same order n, and check for each factor k of n that the number of elements of order k in the two subgroups $\langle S_1 \rangle$ and $\langle S_2 \rangle$ is the same.[6]

In Section D we give a DynAC^0 algorithm for *dynamic* abelian group isomorphism that supports insertions and deletions to both S_1, S_2. Thus, we have shown the following.

▶ **Theorem 23.** *There is a randomized (respectively deterministic)* DynAC^0 *algorithm for abelian group isomorphism that supports* $\mathsf{polylog}(n)$ *(respectively* $O(\log n / \log\log n)$*) insertions and deletions at each step to the generating sets of the two groups.*

7 Making the multiplication table dynamic

We have assumed so far that the overall group G (or monoid) is unchanged and only the generating set for the CGM problem is dynamic. Suppose now that the entries of the multiplication table of G can be modified dynamically. When the table's entries change, it may no longer represent a group (or a monoid). The binary operation $* : G \times G \to G$ is just a *magma*, in general. However, we can show that the dynamic algorithms for abelian CGM still hold, with the proviso that the membership query answers are correct only when the magma is actually an abelian group.

The main property we use here is that at most one group has its multiplication table within linear (i.e. $O(n)$) edit distance from the multiplication table of an n-element magma G.[7] Moreover, from the magma multiplication table we can *decode* this unique group in AC^0. We note that Ergün et al [13] have shown stronger results for this problem in the context of spot checkers; they give randomized self-correction algorithms for a variety of problems. However, for a self-contained presentation, we include a simple proof of a weaker Lemma 28 yielding:

▶ **Theorem 24.** *There is a randomized* DynAC^0 *algorithm that supports* $O(n/\log n)$ *changes to the multiplication table and* $\mathsf{polylog}(n)$ *insertions/deletions to the generating set, with the proviso that when the multiplication table decodes to an abelian group the membership queries are answered with respect to it, and when it does not decode to an abelian group then the query answers could be incorrect. There is also a deterministic* DynAC^0 *algorithm that supports* $O(n/\log n)$ *changes to the multiplication table and* $\log n/\log\log n$ *insertions/deletions to the generating set, with the same proviso as described above.*

8 Conclusion and open ends

We address the dynamic complexity of CGM and isomorphism problems for finite abelian groups input by their multiplication table under $O(\log n/\log\log n)$ changes to the generating set while also allowing sublinear changes to the table itself to be in constant parallel time with an initial logarithmic parallel time precomputation. We can also handle $\mathsf{polylog}(n)$ changes to the generating set by allowing randomness. For the more general algebraic structures, namely commutative monoids, we gave a foundational method to handle single changes, on which the preceding are built.

Natural open questions are to extend the results to more general groups like nilpotent and solvable groups.

[6] Two finite abelian groups are isomorphic iff for each positive integer k the number of elements of order k in the two groups coincide [16].

[7] By the edit distance between multiplication tables $op_1 : G \times G \to G$ and $op_2 : G \times G \to G$ we mean the number of pairs $(a, b) \in G \times G$ such that $op_1(a, b) \neq op_2(a, b)$.

References

1 Miklós Ajtai. $\sum^1{}_1$-formulae on finite structures. *Ann. Pure Appl. Log.*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6.

2 Miklós Ajtai. Approximate counting with uniform constant-depth circuits. In *Advances In Computational Complexity Theory, Proceedings of a DIMACS Workshop, New Jersey, USA, December 3-7, 1990*, pages 1–20, 1990. doi:10.1090/DIMACS/013/01.

3 Vikraman Arvind and T. C. Vijayaraghavan. Classifying problems on linear congruences and abelian permutation groups using logspace counting classes. *Comput. Complex.*, 19(1):57–98, 2010. doi:10.1007/S00037-009-0280-6.

4 László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 409–420. ACM, 1987. doi:10.1145/28395.28439.

5 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. In *Proceedings: Third Annual Structure in Complexity Theory Conference, Georgetown University, Washington, D. C., USA, June 14-17, 1988*, pages 47–59, 1988.

6 David Mix Barrington, Peter Kadau, Klaus-Jörn Lange, and Pierre McKenzie. On the complexity of some problems on groups input as multiplication tables. *Journal of Computer and System Sciences*, 63(2):186–200, 2001. doi:10.1006/JCSS.2001.1764.

7 Arkadev Chattopadhyay, Jacobo Torán, and Fabian Wagner. Graph isomorphism is not AC^0-reducible to group isomorphism. *ACM Trans. Comput. Theory*, 5(4):13:1–13:13, 2013. doi:10.1145/2540088.

8 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.

9 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A strategy for dynamic programs: Start over and muddle through. *Log. Methods Comput. Sci.*, 15(2), 2019. doi:10.23638/LMCS-15(2:12)2019.

10 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 120:1–120:14, 2018. doi:10.4230/LIPICS.ICALP.2018.120.

11 Laxman Dhulipala, David Durfee, Janardhan Kulkarni, Richard Peng, Saurabh Sawlani, and Xiaorui Sun. Parallel batch-dynamic graphs: Algorithms and lower bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1300–1319, 2020. doi:10.1137/1.9781611975994.79.

12 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive incremental evaluation of datalog queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995. doi:10.1007/BF01530820.

13 Funda Ergün, Sampath Kannan, Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60(3):717–751, 2000. doi:10.1006/JCSS.1999.1692.

14 Joshua A. Grochow and Youming Qiao. On p-group isomorphism: Search-to-decision, counting-to-decision, and nilpotency class reductions via tensors. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 16:1–16:38. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CCC.2021.16.

15 Joshua A. Grochow and Youming Qiao. On the complexity of isomorphism problems for tensors, groups, and polynomials I: tensor isomorphism-completeness. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 31:1–31:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ITCS.2021.31.

16 M. Hall. *The Theory of Groups*. AMS Chelsea Publishing Series. AMS Chelsea Pub., 1999.

17 Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms - A quick reference guide. *ACM J. Exp. Algorithmics*, 27:1.11:1–1.11:45, 2022. doi:10.1145/3555806.

18 H.V. Henderson and S.R. Searle. On deriving the inverse of a sum of matrices. *SIAM Review*, 23(1):53–60, 1981.

19 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.

20 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

21 Giuseppe F. Italiano, Silvio Lattanzi, Vahab S. Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 49–58, 2019. doi:10.1145/3323165.3323202.

22 T. Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *Journal of Computer and System Sciences*, 73(6):986–996, 2007. doi:10.1016/J.JCSS.2007.03.013.

23 Meena Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chic. J. Theor. Comput. Sci.*, 1997, 1997. URL: http://cjtcs.cs.uchicago.edu/articles/1997/5/contents.html.

24 Pierre McKenzie and Stephen A. Cook. The parallel complexity of abelian permutation group problems. *SIAM J. Comput.*, 16(5):880–909, 1987. doi:10.1137/0216058.

25 Krzysztof Nowicki and Krzysztof Onak. Dynamic graph algorithms with batch updates in the massively parallel computation model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2939–2958, 2021. doi:10.1137/1.9781611976465.175.

26 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.

27 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. doi:10.1145/1391289.1391291.

28 Á. Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

29 Charles C. Sims. Computation with permutation groups. In Stanley R. Petrick, Jean E. Sammet, Robert G. Tobey, and Joel Moses, editors, *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, SYMSAC 1971, Los Angeles, California, USA, March 23-25, 1971*, pages 23–28. ACM, 1971. doi:10.1145/800204.806264.

30 Xiaorui Sun. Faster isomorphism for p-groups of class 2 and exponent p. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 433–440, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585250.

31 H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM J. Comput.*, 21(4):655–670, 1992. doi:10.1137/0221040.

A Missing proofs from Section 3

▶ **Proposition 2.** *For each node τ of the tree T the subset M_τ associated with τ is the commutative submonoid generated by the subset $L_\tau = \{a \in S \mid a$ such that τ has the leaf labelled a as descendant$\}$. I.e., $M_\tau = \langle L_\tau \rangle$.*

Proof. This is easily proved by induction on the tree T. The point to note is that commutativity of the monoid M is crucial: if $M_\mu = \langle L_\mu \rangle$ and $M_\nu = \langle L_\nu \rangle$ by induction hypothesis, then we note that $M_\mu \cdot M_\nu = \langle L_\mu \cup L_\nu \rangle$ because all the elements commute with each other. Hence $M_\tau = M_\mu \cdot M_\nu = \langle L_\tau \rangle$. ◀

▶ **Lemma 3.** *The* CGM *problem for commutative monoids is in* SAC^1.

Proof. Given $m \in M$, to check if $m \in \langle S \rangle$ we just need to check if m is in the submonoid M_ρ associated with the root ρ of T. ◀

▶ **Proposition 4.** *Given as input a subset $S \subseteq M$ for a commutative monoid M (given by its multiplication table), we can construct the tree data structure T along with the data at each node as described above in* SAC^1*. Furthermore, given a membership query $m \in M$, testing if m is in the current submonoid $\langle S \rangle$ can be done in* AC^0*.*

Proof. For the tree construction, it suffices to observe that we can do the computation of each $M_{\nu:\mu}$ in parallel in SAC^1. For membership testing, if the ρ is the root of the tree then the submonoid $M_\rho = \langle S \rangle$ is available as a list at the node ρ. Hence membership testing is in AC^0. ◀

B Missing proofs from Section 4

▶ **Lemma 8.** *Let $S \subseteq G$, for an abelian group G given by its multiplication table as input. In* FOLL *a $\log |G|$ size subset T of S can be computed that generates the same subgroup as S.*

Proof. Let $S = \{x_1, x_2, \ldots, x_s\}$. For each $i > 1$ in parallel, we can check if x_{i+1} is in $\langle x_1, x_2, \ldots, x_i \rangle$ using the FOLL algorithm of [6]. If $x_{i+1} \notin \langle x_1, x_2, \ldots, x_i \rangle$ then we include x_{i+1} into the set T. Clearly, $|T| \leqslant \log |G|$ and generates the same subgroup and S. ◀

▶ **Lemma 9.** *Let $T \subseteq G$ be of size at most $\log^c n$ for some constant c, where G is an abelian group, $|G| = n$, given by its multiplication table with independent generating set $G = \langle g_1, g_2, \ldots, g_\ell \rangle$. Then in randomized* AC^0 *we can list out the subgroup $\langle T \rangle$ generated by T. In particular, membership testing in the subgroup generated by T can be done in randomized* AC^0*.*

Proof. Let $T = \{x_1, x_2, \ldots, x_r\}$. For each $g \in G$ we have the pre-computed unique product

$$g = \prod_{i=1}^{\ell} g_i^{\alpha_i},$$

using the independent generating set $\{g_1, g_2, \ldots, g_\ell\}$. In particular, for each $x_j \in T$ we have

$$x_j = \prod_{i=1}^{\ell} g_i^{\alpha_{ij}},$$

where $0 \leqslant \alpha_{ij} \leqslant o(g_i) - 1$ for each $i \in [\ell]$. As explained below, we can randomly sample from the group generated by T by picking numbers $\beta_j \in_R [n], 1 \leqslant j \leqslant r$ uniformly at random and computing the product

$$x = \prod_{j=1}^{r} x_j^{\beta_j}.$$

The number of such products is n^r. Furthermore, each element of the subgroup $\langle T \rangle$ occurs in this product with multiplicity exactly $|\{(\beta_1, \beta_2, \ldots, \beta_r) \mid \prod_j x_j^{\beta_j} = 1\}$, as this set is the kernel of the group homomorphism mapping $(\beta_1, \beta_2, \ldots, \beta_r) \mapsto \prod_{j=1}^r x_j^{\beta_j}$. Thus, x is uniformly distributed in $\langle T \rangle$. If we draw, say n^2 such samples x in parallel, the probability that all elements of $\langle T \rangle$ appear is at least $1 - e^{-n}$. Finally, we analyze the complexity of computing the product $x = \prod_{j=1}^r x_j^{\beta_j}$. Notice that it amounts to computing the product $\prod_{i=1}^{\ell} g_i^{\sum_{j=1}^r \beta_j \alpha_{ij}}$.

Now, each of these ℓ exponents $\sum_{j=1}^{r} \beta_j \alpha_{ij}$ is a $\log^c n$ sum of *unary* numbers and can be computed modulo the unary number $o(g_i)$ in AC^0. The final product can be looked up in the pre-computed table to find x. This proves that the group $\langle T \rangle$ can be listed in randomized AC^0 and hence membership testing in $\langle T \rangle$ is also in randomized AC^0. ◀

▶ **Lemma 10.** *The tree data structure B, for the generating set S can be built in $O(\log n)$ parallel time (i.e. in $\mathsf{AC}(\log n)$).*

Proof. The tree has $\log n$ levels and the straightforward computation required at each of the (at most $|S|$) nodes at each level is AC^0. ◀

▶ **Lemma 11.** *Given the data structure for S along with the update sets of insertions I and deletions D, each of $\log^{c+1} n$ size, we can test if some group element g is in the subgroup generated by $(S \cup I) \setminus D$ in AC^0.*

Proof. Let $D = \{x_{i_1}, x_{i_2}, \ldots, x_{i_d}\}$ be the deletions from S, $d = \log^{c+1} n$. From the data structure for S we can find the d subtrees rooted at nodes $\nu_{i_1}, \nu_{i_2}, \ldots, \nu_{i_d}$ where each node ν_{i_j} is the root of the maximal subtree that has exactly one deletion x_{i_j} occurring among its leaves. This can be done in $O(1)$ parallel time using the ancestor boolean array $A[u, v]$. In the data structure for S we already have a $\log n$ size generating set, say $T_{\nu_{i_j} x_{i_j}}$, for each subgroup $H_{\nu_{i_j}:x_{i_j}}$.

Additionally, we find the *maximal subtrees*, rooted at nodes $\mu_{\ell_1}, \mu_{\ell_2}, \ldots, \mu_{\ell_s}$ of the tree, such that these subtrees contain no deletion x_{i_j} as descendant. To get a bound on s, note that each such maximal subtree has as sibling a subtree that contains one or more deletions among its leaves. The roots of all subtrees that have deletions among its leaves are just the ancestors of the ν_{i_j} nodes, and hence are at most $d\lceil \log n \rceil$ in number (each leaf has at most $\lceil \log n \rceil$ ancestors). Thus, $s = O(d \log n) = O(\log^{c+2} n)$. At each node μ_{ℓ_j} $(j \in [s])$ we already have a $\log n$ size generating set, $T_{\mu_{\ell_j}}$ for each subgroup $H_{\mu_{\ell_j}}$. Let $T = \cup_j T_{\nu_{i_j} x_{i_j}}$ and $\hat{T} = \cup_j T_{\mu_{\ell_j}}$. It follows from the above that we can compute T and $\hat{T}$ in $O(1)$ parallel time.

Putting it all together, the group generated by $(S \cup I) \setminus D$ is actually generated by $\hat{T} \cup T \cup I$ which is of $\mathsf{polylog}(n)$ size. Therefore, applying Lemma 9 we can do membership testing in this subgroup in randomized AC^0. ◀

C Missing proofs from Section 5

▶ **Lemma 15.** *Let A be a square matrix of dimension $\mathsf{polylog}(n)$ with entries that are polynomially bounded in n, then each bit of $\det(A)$ can be computed in $\mathsf{AC}(\log \log n)$.*

Proof. It is well known that the determinant of a matrix of n variables can be computed by boolean threshold circuits[8] of polynomial size and logarithmic depth, i.e. it is in TC^1. (Proof sketch: the determinant of a matrix of polynomial dimension with polynomial in n bit entries can be computed in arithmetic SAC^1 [23, Table 2]. In other words, it can be computed by a layered logarithmic depth circuit with gates from $\{+, -, *\}$ where the $*$-gates have fan-in 2. Now by applying [19, Corollary 6.7] each layer of this arithmetic circuit can be simulated in TC^0, i.e. constant-depth threshold circuits). Hence, replacing n with $\log n$, it follows that the determinant of matrices of $\mathsf{polylog}(n)$ dimension with $\mathsf{polylog}(n)$ bit entries can be computed by a threshold circuit of depth $O(\log \log n)$ and size $\mathsf{polylog}(n)$. Furthermore, threshold gates of $\mathsf{polylog}\ n$ fanin can be computed by $\mathsf{poly}(n)$ size uniform AC^0 [1, 2]. Replacing the threshold gates by the corresponding AC^0-circuit of size $\mathsf{poly}(n)$ completes the proof. ◀

[8] Threshold circuits allow for unbounded fanin threshold gates apart from NOT, AND, and OR gates.

▶ **Lemma 16.** *Let $Ax = \bar{b} \pmod{p^m}$ be a system of integer linear equations modulo p^m, where $p^m = n$ is input in unary, and $A \in \mathbb{Z}^{\ell \times t}$ and $\bar{b} \in \mathbb{Z}^l$, $t = O(\log n)$.*

1. *Let $\tilde{A} = [A|p^m I_\ell]$. We can compute the determinants of all square submatrices of $\tilde{A}$ and $[\tilde{A}|\bar{b}]$ in $\mathsf{AC}(\log\log n)$ (i.e. in $\log\log n$ parallel time).*
2. *Furthermore, for the nonsingular submatrices we can also compute their inverses in $\mathsf{AC}(\log\log n)$.*
3. *Given the above we can test the feasibility of $Ax = \bar{b} \pmod{p^m}$ and solve for x in AC^0.*

Proof. For the first part, the number of square submatrices is polynomially bounded as $\tilde{A}$ has dimension $O(\log n) \times O(\log n)$. Reducing modulo p^m, the entries of the matrix are bounded by n. Thus, by Lemma 15 it follows that the determinant as an integer can be computed in $\mathsf{AC}(\log\log n)$. Reducing modulo p^m yields the answer and we know from [19] that the division by a unary number is possible in AC^0.

For the second part, consider every nonsingular submatrix N, i.e. $\det N$ is non-zero. We can compute the entries of N^{-1} by Cramer's rule, as each cofactor of N is also a submatrix of $\tilde{A}$. Since we can compute division of $O(\log n)$-bit integers in AC^0 (see [19, Theorem 5.1]) it follows that these computations can also be done in $\mathsf{AC}(\log\log n)$.

For the last part, notice that feasibility can be tested in AC^0, given the data of first two parts, by Lemma 14. From the set of all $l \times l$ non-singular submatrices of $\tilde{A}$, we choose N, that has the least power of p dividing its determinant, $\nu_p(\det(N))$ is the smallest.

W.l.o.g. N is the first l columns of $\tilde{A}$. Let $Nz' = \bar{b}$ be the system of equations obtained by keeping only the first l columns of $\tilde{A}$ and truncating z beyond the lth coordinate. The solution to this is $z' = N^{-1}\bar{b}$, where the right hand side is computable in AC^0 given N^{-1}. Now we can extend this solution to a solution of $\tilde{A}z = \bar{b}$ by putting $z_i = z'_i$ for each column i in $[l]$ and $z_i = 0$ for other columns. It is easy to see that this must be a solution of $\tilde{A}z = \bar{b}$, as we know the latter is feasible. However, this solution may not necessarily be integer, as we would like. But this rational solution will have the property that denominators of the solution are relatively prime to p as we show below.

Because of Cramer's rule, for any $i \in [l]$, $z_i = \det([N_i|\bar{b}])/\det(N)$, where N_i is the matrix obtained from N by dropping its ith column. Let s be the gcd of the determinants of all $l \times l$ submatrices of $\tilde{A}$. Let t be the gcd of the determinants of all $l \times l$ submatrices of $[\tilde{A}|\bar{b}]$. From the feasibility criterion of Lemma 14, we have that $\nu_p(s) = \nu_p(t)$. But, from the choice of N it is clear that, $\nu_p(s) = \nu_p(\det(N))$ and hence $\nu_p(\det(N)) = \nu_p(t)$. In particular, this implies that, $\nu_p([N_i|\bar{b}]) \geqslant \det(N)$. From this, we have that the denominator of z_i is relatively prime to p for any $i \in [l]$.

Now, we can uniquely modify such a solution to be integral easily in AC^0. ◀

D Missing parts from Section 6

Let G_1 and G_2 be abelian groups, each given a multiplication table as input, say T_1 and T_2, respectively. Let $S_1 \subseteq G_1$ and $S_2 \subseteq G_2$ be subsets. In the static setting, there is a simple polynomial time algorithm for checking if $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are isomorphic: it suffices to list out the two subgroups $\langle S_1 \rangle$ and $\langle S_2 \rangle$, check they have the same order n, and check for each factor k of n that the number of elements of order k in the two subgroups $\langle S_1 \rangle$ and $\langle S_2 \rangle$ is the same.[9]

[9] Two finite abelian groups are isomorphic iff for each positive integer k the number of elements of order k in the two groups coincide [16].

We give a DynAC^0 algorithm for the *dynamic version* of abelian group isomorphism that supports insertions and deletions to both S_1 and S_2. Let $n_1 = \prod_{x \in S_1} o(x)$ and $n_2 = \prod_{y \in S_2} o(y)$, where the orders $o(x), o(y), x \in S_1, y \in S_2$ can be pre-computed for the elements of the two groups G_1 and G_2. Let $n_1 = \prod_{i=1}^{r} p_i^{a_i}$ and $n_2 = \prod_{i=1}^{r} p_i^{b_i}$ be their prime factorizations. We can assume both n_1 and n_2 have the same prime factors. Otherwise, $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are not isomorphic. Let $n_{1i} = n_1/p_i^{a_i}$ and $n_{2i} = n_2/p_i^{b_i}$ and $S_{1i} = \{x^{n_{1i}} \mid x \in S_1\}$ and $S_{1i} = \{y^{n_{1i}} \mid y \in S_2\}$ for $1 \leqslant i \leqslant r$. Since a finite abelian group is a direct product of its (unique) Sylow subgroups we have

▶ **Proposition 25.** *The groups $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are isomorphic iff their p_i-Sylow subgroups $\langle S_{1i} \rangle$ and $\langle S_{2i} \rangle$ are isomorphic.*

Thus, as argued in Sections 4 and 5, it suffices to solve the problem for abelian p-groups. Henceforth, we assume both $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are p-groups. The following lemma from Mckenzie and Cook's work [24], paraphrased in our context, is useful for our algorithm.

▶ **Lemma 26** ([24], Proposition 6.4)**.** *Let $\langle S_1 \rangle \leqslant G_1$ and $\langle S_2 \rangle \leqslant G_2$ be abelian p-groups, and k be largest positive integer such that $p^k \leqslant \max\{|G_1|, |G_2|\}$. For $1 \leqslant j \leqslant k$ let $S_{1j} = \{x^{p^j} \mid x \in S_1\}$ and $S_{2j} = \{y^{p^j} \mid y \in S_2\}$. Then $\langle S_1 \rangle$ and $\langle S_2 \rangle$ are isomorphic if and only if $|\langle S_{1j} \rangle| = |\langle S_{2j} \rangle|$ for $1 \leqslant j \leqslant k$.*

Effectively, the above lemma is a reduction from abelian group isomorphism to abelian CGM. Thus, as observed in [7], in the static setting we can note that the above lemma immediately shows that abelian group isomorphism problem we consider can be solved by $\mathsf{AC}(\log\log n)$ circuits with majority gates. This is by applying the Barrington et al algorithm [6] to enumerate the subgroups $\langle S_{1j} \rangle$ and $\langle S_{2j} \rangle$ in $\mathsf{AC}(\log\log n)$ for each $1 \leqslant j \leqslant k$ and then comparing their orders (for which majority gates are required).

Our strategy for the dynamic version is also based on Lemma 26 because we can apply the results for abelian CGM shown in Sections 4 and 5.

In the dynamic setting, where we have insertions and deletions to the generating sets S_1, S_2, we will use the same data structures developed in Section 4 (for supporting $\mathsf{polylog}(n)$ insertions and deletions) and Section 5 (for supporting $\log n/\log\log n$ insertions and deletions) for the abelian CGM problem but now in parallel for all the generating sets $\langle S_{1j} \rangle$ and $\langle S_{2j} \rangle$ for $1 \leqslant j \leqslant k$.

In order to compute $|\langle S_{1j} \rangle|$ and $|\langle S_{2j} \rangle|$ from membership queries the following lemma, from [24], is useful.

▶ **Lemma 27** ([24], Proposition 6.6)**.** *Let $H = \langle g_1, \ldots, g_r \rangle$ be a finite abelian p-group. Then, $|H| = t_1 t_2 \ldots t_r$ where t_j is the least positive integer such that $g_j^{t_j} \in \langle g_{j+1}, \ldots, g_r \rangle$ for $1 \leqslant j \leqslant r$.*

In the above lemma, as H is a p-group notice that each t_j is a power of p. We will be applying this lemma to groups $\langle S_{1j} \rangle$ and $\langle S_{2j} \rangle$. As $|\langle S_{1j} \rangle| \leqslant n_1 \leqslant |G_1|$ and $|\langle S_{2j} \rangle| \leqslant n_2 \leqslant |G_2|$, and both n_1 and n_2 are logarithmic size in binary, for these groups H at most logarithmically many of the integers t_i are more than 1. Letting $t_j = p^{r_j}$, computing the product $\prod_j t_j = p^{\sum_j r_j}$ amounts to adding at most logarithmically many r_j, each logarithmically bounded. As already observed, such tiny additions can be computed in AC^0.

E Missing parts from Section 7

▶ **Lemma 28.** *Let M_G denote the multiplication table of the group G. Suppose M is a multiplication table obtained from M_G by changing at most δn many entries of M_G, for $\delta < 1/13$. Then there is an AC^0 circuit that takes M as input and outputs M_G.*

Proof. For each $x_i \in G$ the row of x_i in M has at most δn errors in it. Thus, the row for the identity element, say $x_1 = e$ is uniquely determined, because $x_1 \in G$ is the unique element with $x_1 x_j = x_j$ for majority of $j \in [n]$.

For each $z \in G$ there is a unique inverse $z^{-1} \in G$ and $zz^{-1} = e = z^{-1}z$. That means in M_G there are exactly n occurrences of e in the multiplication table. Therefore, in M there are at most $(1+\delta)n$ occurrences of e and at least $(1-\delta)n$ occurrences of e.

Let $S = \{(z,w) \mid z,w \in G, z * w = e\}$, where $*$ is the product operation in the table M. Then

$$(1-\delta)n \leqslant |S| \leqslant (1+\delta)n.$$

Thus, for at least $(1-2\delta)n$ pairs $(z,w) \in S$ we have $z * w = zw = e$ in G.

Now, in order to recover the correct value of the product $x_i x_j$, we look up the products $(x_i * z) * (w * x_j)$ in the table M. Then we have

- $|\{z \in G \mid x_i * z \neq x_i z\}| \leqslant \delta n$.
- $|\{w \in G \mid w * x_j \neq w x_j\}| \leqslant \delta n$.
- $|\{(z,w) \in S \mid z * w \neq zw\}| \leqslant \delta n$.
- $|\{(z,w) \in S \mid (x_i z) * (w x_j) \neq (x_i z)(w x_j)\}| \leqslant \delta n$.

Thus, for at least $(1-6\delta)n$ pairs $(z,w) \in S$ we have $(x_i * z) * (w * x_j) = (x_i z)(w x_j) = x_i(zw)x_j = x_i x_j$.

If we choose $\delta < 1/13$ then the number of such pairs $(z,w) \in S$ is more than $7n/13$. By approximate majority which can be computed in AC^0 [1, 2], we can find this correct value of $x_i x_j$. We can thus recover the entire table M_G in AC^0. ◀

The above lemma suggests the following simple dynamic algorithm for the abelian CGM problem that supports polylog(n) changes to the group multiplication table, as well as bulk insertions/deletions to the generating set (as discussed in Sections 4 and 5).

1. Let M be the current multiplication table. We apply the AC^0 algorithm of Lemma 28 to decode M. Let G be the resulting table.
2. If the decoded table does not give an abelian group, the query answers can be arbitrary (but consistent which can be ensured by remembering the answers to queries already made).
3. Suppose the decoded table gives an abelian group G'. If $G' \neq G$ then for the next $\log n$ steps we rebuild the static data structure for G' in $\mathsf{AC}(\log n)$. We can answer any membership queries, occurring in this window of $\log n$ time steps, arbitrarily. After $\log n$ steps we can replace G with G' and its data structure and continue.

In summary, we have the following.

▶ **Theorem 24.** *There is a randomized* DynAC^0 *algorithm that supports* $O(n/\log n)$ *changes to the multiplication table and* polylog(n) *insertions/deletions to the generating set, with the proviso that when the multiplication table decodes to an abelian group the membership queries are answered with respect to it, and when it does not decode to an abelian group then the query answers could be incorrect. There is also a deterministic* DynAC^0 *algorithm that supports* $O(n/\log n)$ *changes to the multiplication table and* $\log n/\log\log n$ *insertions/deletions to the generating set, with the same proviso as described above.*

Concurrent Stochastic Games with Stateful-Discounted and Parity Objectives: Complexity and Algorithms

Ali Asadi
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Krishnendu Chatterjee
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Raimundo Saona
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Jakub Svoboda
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Abstract

We study two-player zero-sum concurrent stochastic games with finite state and action space played for an infinite number of steps. In every step, the two players simultaneously and independently choose an action. Given the current state and the chosen actions, the next state is obtained according to a stochastic transition function. An objective is a measurable function on plays (or infinite trajectories) of the game, and the value for an objective is the maximal expectation that the player can guarantee against the adversarial player. We consider: (a) stateful-discounted objectives, which are similar to the classic discounted-sum objectives, but states are associated with different discount factors rather than a single discount factor; and (b) parity objectives, which are a canonical representation for ω-regular objectives. For stateful-discounted objectives, given an ordering of the discount factors, the limit value is the limit of the value of the stateful-discounted objectives, as the discount factors approach zero according to the given order.

The computational problem we consider is the approximation of the value within an arbitrary additive error. The above problem is known to be in EXPSPACE for the limit value of stateful-discounted objectives and in PSPACE for parity objectives. The best-known algorithms for both the above problems are at least exponential time, with an exponential dependence on the number of states and actions. Our main results for the value approximation problem for the limit value of stateful-discounted objectives and parity objectives are as follows: (a) we establish TFNP[NP] complexity; and (b) we present algorithms that improve the dependency on the number of actions in the exponent from linear to logarithmic. In particular, if the number of states is constant, our algorithms run in polynomial time.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Concurrent Stochastic Games, Parity Objectives, Discounted-sum Objectives

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.5

Related Version *Full Version*: https://arxiv.org/abs/2405.02486

Funding This research was partially supported by ERC CoG 863818 (ForM-SMArt), Austrian Science Fund (FWF) 10.55776/COE12, and French Agence Nationale de la Recherche (ANR) ANR-21-CE40-0020 (CONVERGENCE project).

1 Introduction

In this work, we present improved complexity results and algorithms for the value approximation of concurrent stochastic games with two classic objectives. Below we present the model of concurrent stochastic games, the relevant objectives, the computational problems, previous results, and finally our contributions.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 5; pp. 5:1–5:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Concurrent stochastic games. Concurrent stochastic games are two-player zero-sum games played on finite-state graphs for an infinite number of steps. These games were introduced in the seminal work of Shapley [26] and are a fundamental model in game theory. In each step, both players simultaneously and independently of the other player choose an action. Given the current state and the chosen actions, the next state is obtained according to a stochastic transition function. An infinite number of such steps results in a play which is an infinite sequence of states and actions. Concurrent stochastic games have been widely studied in the literature from the mathematical perspective [26, 15, 16, 24], and from the algorithmic and computational complexity perspective, including: complexity for reachability objectives [9, 14, 17, 22], algorithms for limit-average objectives [21, 25], complexity for qualitative solutions for omega-regular objectives [8], complexity for quantitative solutions for omega-regular objectives [6, 13], and in the context of temporal logic [1]. In particular, in the analysis of reactive systems, concurrent games provide the appropriate model for reactive systems with components that interact synchronously [1, 11, 12]. Hence, concurrent parity games are relevant for the verification of synchronous reactive systems.

Objectives. An objective is a measurable function that assigns to every play a real-valued reward. The classic discounted-sum objective is as follows: every transition is assigned a reward and the objective assigns to a play the discounted-sum of the rewards. While the classic objective has a single discount factor, the stateful-discounted objective has multiple discount factors. In the stateful-discounted objective, each state is associated with a discount factor, and, in the objective, the discount at a step depends on the current state. We also consider the boolean parity objectives, which are a canonical form to express all ω-regular objectives [27], which provide a robust specification for all properties that arise in verification. For example, all LTL formulas can be converted to deterministic parity automata. In parity objectives, every state is associated with an integer priority, and a play is winning for (or satisfies) the objective if the minimum priority visited infinitely often is even.

Strategies and values. Strategies are recipes that define the choice of actions of the players. They are functions that, given a game history, return a distribution over actions. Given a concurrent stochastic game and an objective, the value of Player 1 at a state is the maximal expectation that the player can guarantee for the objective against all strategies of Player 2. For stateful-discounted objectives, given an ordering of the discount factors, the limit value at a state is the limit of the value function for the discounted objective as the discount factors approach zero in the given order.

Computational problems. Given a concurrent stochastic game, the main computational problems are: (a) the *value-decision* problem, given a state and a threshold α, asks whether the value at the state is at least α; and (b) the *value-approximation* problem, given a state and an error $\varepsilon > 0$, asks to compute an approximation of the value for the state within an additive error of ε. We consider the above problems for the limit value of stateful-discounted objectives and the value for parity objectives.

Motivation. The motivation to study the limit of the stateful-discounted objective is as follows. First, this limit generalizes the classic limit-average objectives. Second, it characterizes the value for the parity objectives in concurrent stochastic games [18, 10], where the order of limit corresponds to the order of importance of priorities in parity objectives. Third, the limit value has been shown to correspond to the value for other objectives such as priority mean-payoff for various subclasses of concurrent stochastic games [19]. The motivation to study the value-approximation problem as opposed to the value-decision is that for concurrent games, even for special classes of objectives such as reachability and safety, values can be irrational, and the decision problem related to exact value is SQRT-SUM hard [14] as explained below. Hence, approximation of values is a natural problem to study from an algorithmic and computational complexity perspective.

Previous results. For a single discount factor, the limit value corresponds to the value of the well-studied mean-payoff or long-run average objectives [24], and, for parity objectives, the computational problems admit a linear reduction to the limit value of stateful-discounted objectives [18, 10]. The value-decision problem for concurrent stochastic games is SQRT-SUM hard [14]: this result holds for reachability objectives, and hence also for parity objectives and the limit value for even a single discount factor. The SQRT-SUM problem is a classic problem in computational geometry, and whether SQRT-SUM belongs to NP has been a long-standing open problem. The complexity upper bounds for the value-approximation problem of concurrent stochastic games are as follows: (a) EXPSPACE for the limit value of stateful-discounted objectives; and (b) PSPACE for parity objectives [5, 6]. The above result for the limit value follows from a reduction to the theory of reals, where the number of discount factors corresponds to the number of quantifier alternation. For the special class of reachability objectives, the complexity upper bound of TFNP[NP] for the value-approximation problem has been established in [17], where TFNP[NP] is the total functional form of the second level of the polynomial hierarchy. The result of [17] has been recently extended to limit-average objectives (which correspond to the limit-value of single discount factor) [4]. To the best of our knowledge, the above complexity upper bounds are the best bounds for limit value of general stateful-discounted objectives and parity objectives. The best-known algorithms for the value-approximation problem are as follows: (a) double exponential time for the limit value of stateful-discounted objectives; (b) exponential time for parity objectives, where the exponent is a product that depends at least linearly on the number of states and actions [6, 5] (see Section 3 for further details). While iterative approaches are desirable, they neither exist for parity objectives nor guarantee efficiency even in special cases. For example, for reachability and safety objectives, iterative approaches like value-iteration or strategy-iteration have a double-exponential lower bound [20]; and, for parity objectives, iterative approaches like strategy iteration are not known as strategies require infinite-memory [8].

Our contributions. In this work, our main contributions are as follows: (a) we establish TFNP[NP] upper bounds for the value-approximation problem for concurrent stochastic games, both for the limit value of stateful-discounted and parity objectives; and (b) we present algorithms which are exponential time and improve the dependency on the number of actions in the exponent from linear to logarithmic. In particular, if the number of states is constant, our algorithms run in polynomial time. The comparison of previous results and our results is summarized in Tables 1 and 2.

Technical contributions. We first present a bound on the roots of multi-variate polynomials with integer coefficients (Section 4.2). Given the bounds on roots of polynomials, we establish new characterizations for the limit and stateful-discounted values (Section 4.3 and Section 4.4), which lead to an approximation of the limit value by the stateful-discounted value when the discount factors are double-exponentially small (Section 4.5). Given this connection, we establish the improved complexities and algorithms for the value-approximation for the limit value of stateful-discounted objectives and parity objectives in Section 5 and Section 6. Proofs omitted due to space restrictions are provided in the Full version.

Table 1 Complexity upper bounds of the value-approximation in concurrent stochastic games for the limit value of stateful-discounted objectives and parity objectives.

	Complexity	
	Previous	**Ours**
Limit	EXPSPACE (Theory of reals)	TFNP[NP] (Theorem 5-Item 1, Theorem 6-Item 1)
Parity	PSPACE [6, 5]	

Table 2 Algorithmic upper bounds of the value-approximation in concurrent stochastic games for the limit value of stateful-discounted objectives and parity objectives, where n is the number of states, m is the number of actions, d is the number of discount factors/parity index, B is the bit-size of numbers in the input, ε is the additive error, and exp is the exponential function.

	Algorithms	
	Previous	**Ours**
Limit	$\exp\left(\mathcal{O}(2^d m^2 n + \log(1/\varepsilon) + \log(\mathrm{B}))\right)$ (Theory of reals)	$\exp\left(\mathcal{O}\begin{pmatrix} nd\log(m) + \log(\mathrm{B}) \\ + \log(\log(1/\varepsilon)) \end{pmatrix}\right)$ (Theorem 5-Item 2, Theorem 6-Item 2)
Parity	$\exp\left(\mathcal{O}\begin{pmatrix} mn + d\log(n) + \log(\mathrm{B}) \\ + \log(\log(1/\varepsilon)) \end{pmatrix}\right)$ [6, 5]	

2 Preliminaries

We present standard definitions related to concurrent stochastic games.

Basic Notations. Given a finite set $\mathcal{X}$, a probability distribution over $\mathcal{X}$ is a function $\mu\colon \mathcal{X} \to [0,1]$ such that $\sum_{x\in\mathcal{X}} \mu(x) = 1$. The set of all probability distributions over $\mathcal{X}$ is denoted by $\Delta(\mathcal{X})$. For $\mu \in \Delta(\mathcal{X})$, the support of μ is defined as $\operatorname{supp}(\mu) \coloneqq \{x \mid \mu(x) > 0\}$. For a positive integer k, the set of positive integers smaller than or equal to k is defined as $[k] \coloneqq \{1, \dots, k\}$. Given a real x, we denote 2^x by $\exp(x)$.

Concurrent stochastic games. A concurrent stochastic game (CSG) is a two-player finite game $G = (\mathcal{S}, \mathcal{A}, \mathcal{B}, \delta)$ consisting of

- the set of states $\mathcal{S}$, of size n;
- the sets of actions for each player $\mathcal{A}$ and $\mathcal{B}$, with at most m actions; and
- the stochastic transition function $\delta\colon \mathcal{S} \times \mathcal{A} \times \mathcal{B} \to \Delta(\mathcal{S})$.

Steps. Given an initial state $s \in \mathcal{S}$, the game proceeds as follows. In each step, both players choose an action simultaneously, $a \in \mathcal{A}$ and $b \in \mathcal{B}$. Based on both actions (a, b) and current state s, the next state is drawn according to the probability distribution $\delta(s, a, b)$.

Histories and plays. At step k of CSGs, each player possesses information in the form of the finite sequence of the states visited and the actions chosen by both players. A k-history $\omega^{(k)} = \langle s_0, a_0, b_0, s_1, a_1, b_1, \dots, s_k\rangle$ is a finite sequence of states and actions such that, for all steps $0 \le t < k$, we have $s_{t+1} \in \operatorname{supp}(\delta(s_t, a_t, b_t))$. The set of all k-histories is denoted by $\Omega^{(k)}$. Similarly, a play $\omega = \langle s_0, a_0, b_0, s_1, a_1, b_1, \dots\rangle$ is an infinite sequence of states and actions such that, for all steps $t \ge 0$, we have $s_{t+1} \in \operatorname{supp}(\delta(s_t, a_t, b_t))$. The set of all plays is denoted by Ω. For any state s, the set of all plays starting at s, i.e., $\omega = \langle s_0, a_0, b_0, \dots\rangle$ where $s_0 = s$, is denoted by Ω_s.

Objectives. An objective is a measurable function that assigns a real number to all plays. Player 1 aims to maximize the expectation of the objective, while Player 2 minimizes it.

- *Parity objective.* Given a priority function $p\colon \mathcal{S} \to \{0, \dots, d\}$ with d as its index, the *parity* objective is an indicator of the even parity condition on minimal priority visited infinitely often in plays. More formally, we define $\text{Parity}_p \colon \Omega \to \{0, 1\}$ as

$$\text{Parity}_p(\omega) \coloneqq \begin{cases} 1 & \min\{p(s) \mid \forall i \geq 0\ \exists j \geq i\ s_j = s\} \text{ is even} \\ 0 & \text{otherwise} \end{cases}$$

- *Stateful-discounted objective.* Consider d discount factors $\lambda_1, \cdots, \lambda_d \in (0, 1]$. Given an assignment function $\chi\colon \mathcal{S} \to [d]$, we define the discount function $\Lambda\colon \mathcal{S} \to \{\lambda_1, \cdots, \lambda_d\}$ as $\Lambda(s) \coloneqq \lambda_{\chi(s)}$ for all states $s \in \mathcal{S}$. Given a reward function $r\colon \mathcal{S} \times \mathcal{A} \times \mathcal{B} \to [0, 1]$ that assigns a reward value $r(s, a, b)$ for all (s, a, b), the stateful-discounted objective $\text{Disc}_\Lambda \colon \Omega \to [0, 1]$ is defined as, for all $\omega = \langle s_0, a_0, b_0, \cdots \rangle$,

$$\text{Disc}_\Lambda(\omega) \coloneqq \sum_{i \geq 0} \left(r(s_i, a_i, b_i) \Lambda(s_i) \prod_{j<i} 1 - \Lambda(s_j) \right) .$$

Strategies. A strategy is a function that assigns a probability distribution over actions to every finite history and is denoted by $\sigma\colon \bigcup_k \Omega^{(k)} \to \Delta(\mathcal{A})$ for Player 1 (resp. $\tau\colon \bigcup_k \Omega^{(k)} \to \Delta(\mathcal{B})$ for Player 2). Given strategies σ and τ, the game proceeds as follows. At step k, the current history is some $\omega^{(k)} \in \Omega^{(k)}$. Player 1 (resp. Player 2) chooses an action according to the distribution $\sigma\left(\omega^{(k)}\right)$ (resp. $\tau\left(\omega^{(k)}\right)$). The set of all strategies for Player 1 and Player 2 is denoted by Σ and Γ respectively. A *stationary* strategy depends on the past observations only through the current state. A stationary strategy for Player 1 (resp. Player 2) is denoted by $\sigma\colon \mathcal{S} \to \Delta(\mathcal{A})$ (resp. $\tau\colon \mathcal{S} \to \Delta(\mathcal{B})$). The set of all stationary strategies for Player 1 and Player 2 is denoted by Σ^S and Γ^S respectively. A *pure stationary* strategy $\sigma\colon \mathcal{S} \to \mathcal{A}$ (resp. $\tau\colon \mathcal{S} \to \mathcal{B}$) for Player 1 (resp. Player 2) is a stationary strategy that maps to Dirac distributions only. The set of all pure stationary strategies for Player 1 and Player 2 is denoted by Σ^{PS} and Γ^{PS} respectively.

Probability space. An initial state s and a pair of strategies (σ, τ) induce a unique probability over Ω_s, endowed with the sigma-algebra generated by the cylinders corresponding to finite histories. We denote by $\mathbb{P}_s^{\sigma,\tau}$ and $\mathbb{E}_s^{\sigma,\tau}$ the probability and the expectation respectively.

We state the determinacy for CSGs with stateful-discounted and parity objectives.

▶ **Theorem 1** (Parity determinacy [23]). *For all CSGs, states s, and priority functions p,*

$$\sup_{\sigma \in \Sigma} \inf_{\tau \in \Gamma} \mathbb{E}_s^{\sigma,\tau}[\text{Parity}_p] = \inf_{\tau \in \Gamma} \sup_{\sigma \in \Sigma} \mathbb{E}_s^{\sigma,\tau}[\text{Parity}_p] .$$

▶ **Theorem 2** (Stateful-discounted determinacy [26]). *For all CSGs, states s, reward functions, and discount functions Λ, we have*

$$\sup_{\sigma \in \Sigma^S} \inf_{\tau \in \Gamma^S} \mathbb{E}_s^{\sigma,\tau}[\text{Disc}_\Lambda] = \inf_{\tau \in \Gamma^S} \sup_{\sigma \in \Sigma^S} \mathbb{E}_s^{\sigma,\tau}[\text{Disc}_\Lambda] .$$

Values. The above determinacy results imply that switching the quantification order of strategies do not make a difference and leads to the unique notion of value. The stateful-discounted value for a state s is defined as $\text{val}_\Lambda(s) \coloneqq \sup_{\sigma \in \Sigma^S} \inf_{\tau \in \Gamma^S} \mathbb{E}_s^{\sigma,\tau}[\text{Disc}_\Lambda]$. We define the parity value $\text{val}_p(s)$ for a state s analogously. The limit value for a state s is defined as $\text{val}_\chi(s) \coloneqq \lim_{\lambda_1 \to 0^+} \cdots \lim_{\lambda_d \to 0^+} \text{val}_\Lambda(s)$.

ε-optimal strategies. Given $\varepsilon \geq 0$, a strategy σ for Player 1 is *ε-optimal* for the stateful-discounted objective if, for all states $s \in \mathcal{S}$, we have $\inf_{\tau \in \Gamma^{\mathcal{S}}} \mathbb{E}_s^{\sigma,\tau}[\mathrm{Disc}_\Lambda] \geq \mathrm{val}_\Lambda(s) - \varepsilon$. We say the strategy is *optimal* if $\varepsilon = 0$. The notion of ε-optimal strategies for Player 2 is defined analogously. Similarly, we define ε-optimal strategies for the parity objectives.

Approximate value problems. We consider two value problems stated as follows.

LimitValue. Consider a CSG G, a state s, a reward function r, an assignment function $\chi \colon \mathcal{S} \to [d]$, and an additive error ε. The transition function δ and the reward function r are represented by rational numbers using at most B bits. Compute an approximation v of the limit value at state s such that $|v - \mathrm{val}_\chi(s)| \leq \varepsilon$.

ParityValue. Consider a CSG G, a state s, a priority function p with index d, and an additive error ε. The transition function δ is represented by rational numbers using at most B bits. Compute an approximation v of the parity value at state s such that $|v - \mathrm{val}_p(s)| \leq \varepsilon$.

3 Overview of Results

We first discuss the previous results in the literature, and then, we show our contributions.

Previous results. We discuss the previous works on computing the approximation of limit and parity values in CSGs. A natural approach for these computational problems is via the theory of reals. We first recall the main computational result of the theory of reals, which is a specialization of [3, Theorem 1].

▶ **Theorem 3** ([3, Theorem 1]). *Consider ℓ variables $x_1, \cdots, x_\ell$ and the set of polynomials $\mathcal{P} = \{P_1, \cdots, P_k\}$, where, for all $i \in [k]$, we have P_i is a polynomial in $x_1, \cdots, x_\ell$ of degree at most D with integer coefficients of bit-size at most* B*. Let $X_1, \cdots, X_d$ be a partition of $x_1, \cdots, x_\ell$ into d subsets such that X_i has size ℓ_i. Let $\Phi = (Q_d X_d) \cdots (Q_1 X_1)\ \phi(P_1, \cdots, P_k)$ be a sentence with d alternating quantifiers $Q_i \in \{\exists, \forall\}$ such that $Q_{i+1} \neq Q_i$, and $\phi(P_1, \cdots, P_k)$ is a quantifier-free formula with atomic formulas of the form $P_i \bowtie 0$ where $\bowtie \in \{<, >, =\}$. Then, there exists an algorithm to decide the truth of Φ in time*

$$k^{\prod_i \mathcal{O}(\ell_i+1)} \cdot D^{\prod_i \mathcal{O}(\ell_i)} \cdot \mathcal{O}(\mathrm{len}(\phi)\mathrm{B}^2)\,,$$

where $\mathrm{len}(\phi)$ *is the length of the quantifier-free formula ϕ.*

Along with the above algorithmic result, the following complexity result also follows from [3]: if there is constant number of quantifier alternations, then complexity is PSPACE, and in general the complexity is EXPSPACE. We now discuss the algorithms and complexity results from the literature for the limit value of stateful discounted-sum objectives. The basic computational approach is via the theory of reals. For a single discount factor, the reduction to the theory of reals and dealing with its limit (which corresponds to limit-average objectives) was presented in [7]. In the general case (d discount factors), each limit can be considered as the quantification $\exists \varepsilon_{i'}\ \forall \varepsilon_i \leq \varepsilon_{i'}$ in the theory of reals. Thus, concurrent stochastic games with the limit value of stateful-discounted objectives can be reduced to the theory of reals with quantifier alternation. This reduction gives a theory of reals sentence with the following parameters:

$$\ell = \mathcal{O}(m^2 n), \quad k = \mathcal{O}(m^2 n), \quad D = 4, \quad \prod_i (\ell_i + 1) = \mathcal{O}(2^d m^2 n),$$

Applying Theorem 3 to the reduction we obtain the following result.

▶ **Theorem 4** (LimitValue: Previous Result). *For the* LimitValue *problem, the following assertions hold.*

1. *The problem is in* EXPSPACE*; and*
2. *the problem can be solved in time* $\exp\left(\mathcal{O}\left(2^d m^2 n + \log(1/\varepsilon) + \log(\mathrm{B})\right)\right)$.

For parity objectives, the result of [18, 10] reduces CSGs with parity objectives to CSGs with the limit value of stateful-discounted objectives. The reduction is achieved as follows. Consider the formula $R(a_0, a_1, \ldots, a_{2n-1})$ from [10], which is a formula with multiple discount factors. Since for stateful-discounted objectives the mapping is contractive, the fixpoints are unique (least and greatest fixpoints coincide). The last sentence of [10, Theorem 4] states that the limit of $R(a_0, a_1, \ldots, a_{2n-1})$ corresponds to the value for parity objectives. The Pre operator of the formula corresponds to the Bellman-operator for stateful-discounted objectives, which establish the connection to stateful-discounted games. This connection is made more explicit in the construction provided in [18, Section 2.2]. This linear reduction and the above theorem lead to similar results for parity objectives.

Besides this reduction to the theory of reals, there are two other approaches for the ParityValue problem. First, we can consider the nested fixpoint characterization as provided in [13] and a reduction to the theory of reals with quantifier alternation. However, this does not lead to a better complexity. Second, a different approach is presented in [6, 5, Chapter 8]. This approach has the following components: (a) it enumerates over all possible subsets of actions for every state; (b) for each of the enumeration, it requires a solution of a qualitative value problem (or limit-sure winning) in concurrent stochastic games with parity objectives, and the value-approximation for concurrent stochastic games with reachability objectives. This approach gives PSPACE complexity and the algorithmic complexity is $\exp\left(\mathcal{O}\left(mn + d\log(n) + \log(\log(1/\varepsilon)) + \log(\mathrm{B})\right)\right)$.

Our contributions. Our main results are the following.

▶ **Theorem 5** (LimitValue: Complexity and Algorithm). *For the* LimitValue *problem, the following assertions hold.*

1. *The problem is in* TFNP[NP]*; and*
2. *the problem can be solved in time* $\exp\left(\mathcal{O}\left(nd\log(m) + \log(\mathrm{B}) + \log(\log(1/\varepsilon))\right)\right)$.

▶ **Theorem 6** (ParityValue: Complexity and Algorithm). *For the* ParityValue *problem, the following assertions hold.*

1. *The problem is in* TFNP[NP]*; and*
2. *the problem can be solved in time* $\exp\left(\mathcal{O}\left(nd\log(m) + \log(\mathrm{B}) + \log(\log(1/\varepsilon))\right)\right)$.

4 Mathematical Properties

In this section, we present an approach of the limit value approximation via the stateful-discounted value (Theorem 12). We follow the approach of [2] extending it step by step using similar arguments. We use this technical result to improve complexities and algorithmic bounds of computing ε-approximation of the limit and parity values. This section is organized as follows. In Section 4.1, we present some useful definitions. In Section 4.2, we present a bound on the roots of multi-variate polynomials which is used to establish a connection between the stateful-discounted and limit values. In Sections 4.3 and 4.4, we introduce new characterizations of the stateful-discounted and limit values. In Section 4.5, we show Theorem 12.

4.1 Definitions

We present some basic notations and definitions related to polynomials and matrices.

Basic notations. Given a positive integer k, we define $\mathrm{bit}(k) := \lceil \log_2(k+2) \rceil$. For a rational k_1/k_2, we define $\mathrm{bit}(k_1/k_2) := \mathrm{bit}(k_1) + \mathrm{bit}(k_2)$. Given a real x, the standard sign function is $\mathrm{sign}(x) = -1$ if $x < 0,\ 0$ if $x = 0,\ 1$ if $x > 0$. Moreover, we use the classic arithmetic with infinity, i.e., $x + \infty = \infty$ and $x - \infty = -\infty$.

Polynomials. A *uni-variate* polynomial P of degree D with integer coefficients of bit-size B is defined as $P(x) := \sum_{i=0}^{D} c_i x^i$ where $|c_i| < 2^{\mathrm{B}}$. A k-variate polynomial P in $x_1, \cdots, x_k$ of degree $D_1, \cdots, D_k$ with integer coefficients of bit-size B is defined as

$$P(x_1, \cdots, x_k) := \sum_{0 \le i_1 \le D_1} \cdots \sum_{0 \le i_k \le D_k} c_{i_1, \cdots, i_k} \prod_{j=1}^{k} x_j^{i_j} ,$$

where $|c_{i_1,\cdots,i_k}| < 2^{\mathrm{B}}$. Polynomial P is nonzero if $c_{i_1,\cdots,i_k} \neq 0$ for some $i_1, \cdots, i_k$. We say α is a root of P if $P(\alpha) = 0$. In this work, we only consider real roots.

Matrix notations. Given a square matrix M, we denote the determinant of M by $\det(M)$ and denote the signed sum of all minors of M by $S(M)$. Given two $k \times \ell$ matrices M_1 and M_2, we denote the Hadamard product of M_1 and M_2 by $M_1 \odot M_2$. Given a positive integer k, often implicitly clear from context, we denote by $\mathbf{1}$ (resp. $\mathbf{0}$) the k-dimensional vector with all elements equal to 1 (resp. 0) and denote by Id the $k \times k$ identity matrix.

Matrix games. A matrix M defines a game played between two opponents, where rows (resp. columns) correspond to possible actions for the row- (resp. column-) player, and the entry $(M)_{i,j}$ is the reward the column-player pays the row-player when the pair of actions (i, j) is chosen. The value of a matrix game, denoted $\mathrm{val}\, M$, is the maximum amount the row-player can guarantee, i.e., the amount they can obtain regardless of the column-player's strategy.

4.2 Bounds on Roots of Polynomials with Integer Coefficients

We present a bound on the roots of multi-variate polynomials P with integer coefficients (Theorem 7). This result shows that there exists a region close to $\mathbf{0}$ within which P does not have a root. We use this technical result to establish a connection between the stateful-discounted value and limit value.

▶ **Lemma 7.** *Consider a nonzero polynomial P in $x_1, \cdots, x_\ell$ of degrees $D_1, \cdots, D_\ell$ with integer coefficients of bit-size* B*. Let $D := \max(D_1, \cdots, D_\ell)$ and $\mathrm{B}_1 := 4\ell \operatorname{bit}(D) + \mathrm{B} + 1$. Then,*

$$\begin{aligned}&\forall x_1 \in (0, \exp(-\mathrm{B}_1)] \quad \forall x_2 \in \left(0, (x_1)^{D+1}\right] \quad \cdots \quad \forall x_\ell \in \left(0, (x_{\ell-1})^{D+1}\right] \\ &|P(x_1, \cdots, x_\ell)| \geq \exp(\mathrm{B}_1 - \ell) \cdot (x_\ell)^{D+1} .\end{aligned}$$

Proof sketch. We partition P into $P(x) = P_0 + P_1(x_1) + \cdots + P_\ell(x_1, \cdots, x_\ell)$. Consider the smallest i where $P_i \neq 0$. The proof has following key components: (a) By fixing $x_1, \cdots, x_{i-1}$, we obtain a uni-variate polynomial from P_i. By the fact that $x_i \ll x_j$ for all $j < i$, we obtain $|P_i(x_1, \cdots, x_i)| \neq 0$. (b) Due to the constraints over the variables, $x_j \ll x_i$ for all $j > i$, we have $|P_j(x_1, \cdots, x_j)| \ll |P_i(x_1, \cdots, x_i)|$. Thus, we can ignore the effect of $P_j(x_1, \cdots, x_j)$ on $|P(x_1, \cdots, x_\ell)|$ for all $j > i$. Thus, $P(x_1, \cdots, x_\ell) \approx P_i(x_1, \cdots, x_i) \neq 0$. ◀

4.3 Characterization of Stateful-discounted Value

We introduce a new characterization of the stateful-discounted value in CSGs (Theorems 8 and 9), which generalizes results of [2] from a single discount factor to multiple discount factors. In particular, Theorem 9 generalizes Theorem 1 of [2].

Stateful-discounted payoff. Consider a CSG G, a state s, a reward function r, and a discount function Λ. Given a pair of stationary strategies (σ, τ), we define the stateful-discounted payoff as $\nu^{\sigma,\tau}(s) := \mathbb{E}_s^{\sigma,\tau}[\mathrm{Disc}_\Lambda]$. By fixing σ and τ, we obtain a transition function $\delta^{\sigma,\tau}(s, s') := \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \sigma(s)(a) \cdot \tau(s)(b) \cdot \delta(s, a, b)(s')$, which is described as a matrix, i.e., $\delta^{\sigma,\tau} \in \mathbb{R}^{n \times n}$. The stage reward function is defined as $r^{\sigma,\tau}(s) := \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \sigma(s)(a) \cdot \tau(s)(b) \cdot r(s, a, b)$, which is described as a vector, i.e., $r^{\sigma,\tau} \in \mathbb{R}^n$. Therefore, the Bellman operator defined in [26] can be written as a recursive expression:

$$\nu^{\sigma,\tau} = \Lambda \odot r^{\sigma,\tau} + (\mathbf{1} - \Lambda) \odot (\delta^{\sigma,\tau} \nu^{\sigma,\tau}) .$$

The matrix $\mathrm{Id} - \left((\mathbf{1} - \Lambda)\, \mathbf{1}^\top\right) \odot \delta^{\sigma,\tau}$ is strictly diagonally dominant, and therefore, is invertible. By Cramer's rule, we have

$$\nu^{\sigma,\tau}(s) = \frac{\nabla_\Lambda^s(\sigma, \tau)}{\nabla_\Lambda(\sigma, \tau)} , \tag{1}$$

where $\nabla_\Lambda(\sigma, \tau) := \det\left(\mathrm{Id} - \left((\mathbf{1} - \Lambda)\, \mathbf{1}^\top\right) \odot \delta^{\sigma,\tau}\right)$ and $\nabla_\Lambda^s(\sigma, \tau)$ is the determinant of a matrix derived by substituting the s-th column of the matrix $\mathrm{Id} - \left((\mathbf{1} - \Lambda)\mathbf{1}^\top\right) \odot \delta^{\sigma,\tau}$ with $\Lambda \odot r^{\sigma,\tau}$.

Auxiliary matrix game $W_\Lambda^s(z)$. We define a matrix game where the actions of each player are the pure stationary strategies in the stochastic game. The payoff of the game is obtained by the linearization of the quotient in Eq. (1). More formally, for all parameters $z \in \mathbb{R}$, $\widehat{\sigma} \in \Sigma^{PS}$, and $\widehat{\tau} \in \Gamma^{PS}$, we define the payoff of the matrix game as

$$W_\Lambda^s(z)[\widehat{\sigma}, \widehat{\tau}] := \nabla_\Lambda^s(\widehat{\sigma}, \widehat{\tau}) - z \cdot \nabla_\Lambda(\widehat{\sigma}, \widehat{\tau}) .$$

The value of $W_\Lambda^s(z)$ is denoted by $\mathrm{val}\,(W_\Lambda^s(z))$.

The following results (Theorem 8 and Theorem 9) connect the stateful-discounted value with the value of the matrix game.

▶ **Lemma 8.** *Consider a CSG G, a state s, a reward function, and an assignment function $\chi \colon \mathcal{S} \to [d]$. Then, the following assertions hold.*

1. *The map $(z, \lambda_1, \cdots, \lambda_d) \mapsto \mathrm{val}(W^s_\Lambda(z))$ is continuous;*
2. *for all discount factors $\lambda_1, \cdots, \lambda_d$ and $z_1, z_2 \in \mathbb{R}$ such that $z_1 \leq z_2$, we have that $\mathrm{val}\left(W^s_\Lambda(z_1)\right) \geq \mathrm{val}\left(W^s_\Lambda(z_2)\right) + (z_2 - z_1)\left(\min_i \lambda_i\right)^n$, in particular, $z \mapsto \mathrm{val}(W^s_\Lambda(z))$ is strictly decreasing; and*
3. *for all discount factors $\lambda_1, \cdots, \lambda_d$, we have $\mathrm{val}\left(W^s_\Lambda(\mathrm{val}_\Lambda(s))\right) = 0$.*

Proof sketch. We present the proof sketch for each item as follows.

1. The entries of the matrix game depend continuously on the parameters $z, \lambda_1, \cdots, \lambda_d$. Thus, its value is also continuous in the parameters.
2. For all $z_1 < z_2$, we show that all of the entries of $W^s_\Lambda(z_1)$ are strictly larger than $W^s_\Lambda(z_2)$. By quantifying this difference in each entry of the matrix, we obtain the desired inequality.
3. We show that there exist optimal strategies in the matrix game $W^s_\Lambda(\mathrm{val}_\Lambda(s))$ which are derived from optimal strategies in G. These strategies guarantee that the value of the matrix game is 0. ◀

▶ **Corollary 9.** *Consider a CSG G, a state s, a reward function, and a discount function Λ. Then, $\mathrm{val}_\Lambda(s)$ is the unique $z^* \in \mathbb{R}$ such that $\mathrm{val}\left(W^s_\Lambda(z^*)\right) = 0$.*

Proof. By Theorem 8-Item 2, the mapping $z \mapsto \mathrm{val}\left(W^s_\Lambda(z)\right)$ is strictly decreasing. By Theorem 8-Item 3, we know that $\mathrm{val}\left(W^s_\Lambda(\mathrm{val}_\Lambda(s))\right) = 0$. Hence, there exists the unique $z^* = \mathrm{val}_\Lambda(s) \in \mathbb{R}$ such that $\mathrm{val}\left(W^s_\Lambda(z^*)\right) = 0$, which yields the result. ◀

4.4 Characterization of Limit Value

We introduce a new characterization of the limit value in CSGs (Theorems 10 and 11), which generalizes results presented in [2] from a single discount factor to multiple discount factors. In particular, Theorem 11 generalizes Theorem 2 of [2].

Limit function. Given a CSG G, a reward function, and an assignment function $\chi \colon \mathcal{S} \to [d]$, we define the limit function as

$$F^s_\chi(z) := \lim_{\lambda_1 \to 0^+} \cdots \lim_{\lambda_d \to 0^+} \frac{\mathrm{val}\left(W^s_\Lambda(z)\right)}{(\lambda_d)^n}.$$

The following statements (Theorem 10 and Theorem 11) connect the limit value with the limit function.

▶ **Lemma 10.** *Consider a CSG G, a state s, a reward function, and an assignment function $\chi \colon \mathcal{S} \to [d]$. Then, the following assertions hold.*

1. *For all $z \in \mathbb{R}$, the limit $F^s_\chi(z)$ exists in $\mathbb{R} \cup \{-\infty, +\infty\}$; and*
2. *There exists $z_1, z_2 \in \mathbb{R}$ such that $F^s_\chi(z_2) \leq 0 \leq F^s_\chi(z_1)$.*

Proof sketch. We present the proof sketch for each item as follows.

1. By Theorem 7, we show that given a fixed parameter z, there exist two multi-variate polynomials P and Q in $\lambda_1, \cdots, \lambda_d$ such that for all small enough $\lambda_1, \cdots, \lambda_d$, we have $\mathrm{val}(W^s_\Lambda(z)) = \frac{P(\lambda_1, \cdots, \lambda_d)}{Q(\lambda_1, \cdots, \lambda_d)}$, which implies the existence of the limit.
2. We provide two numbers z_1 and z_2 such that $\mathrm{sign}(F^s_\chi(z_1)) \neq \mathrm{sign}(F^s_\chi(z_2))$. ◀

▶ **Corollary 11.** *Consider a CSG G, a state s, a reward function, and an assignment function $\chi \colon \mathcal{S} \to [d]$. Then, $\mathrm{val}_\chi(s)$ is the unique $z^* \in \mathbb{R}$ such that*

$$\forall z > z^* \quad F^s_\chi(z) < 0 \qquad \textit{and} \qquad \forall z < z^* \quad F^s_\chi(z) > 0\,.$$

Proof sketch. By Theorem 8-Item 2, F^s_χ is decreasing. By Theorem 10-Item 2, there exists a unique sign-changing point. By the definition of the limit value and F^s_χ, we show that this sign-changing point is $\mathrm{val}_\chi(s)$. ◀

4.5 Approximation of Limit Value

We introduce an approach for the approximation of the limit value via the stateful-discounted value (Theorem 12).

▶ **Theorem 12.** *Consider a CSG G, a state s, a reward function r, an assignment function $\chi\colon \mathcal{S} \to [d]$, and an additive error $\varepsilon > 0$. The transition function δ and the reward function r are represented by rational numbers of bit-size* B. *Fix*

$$D \coloneqq \max(|\Sigma^{PS}|, |\Gamma^{PS}|), \qquad \mathrm{B}_1 \coloneqq 11Dn(\mathrm{B} + \mathrm{bit}(n) + \mathrm{bit}(D) + \mathrm{bit}(\varepsilon))\,,$$

and, for all $1 \le i \le d$, we set $\lambda_i^0 \coloneqq \exp\left(-\mathrm{B}_1(nD+1)^{i-1}\right)$. Then, we have

$$|\,\mathrm{val}_{\Lambda^0}(s) - \mathrm{val}_\chi(s)| \le \varepsilon\,.$$

Proof sketch. The proof has following key components:

- We show that there exists a finite set of rational functions with bounded degrees and coefficients such that for all $z, \lambda_1, \cdots, \lambda_d$, the value of $W^s_\Lambda(z)$ corresponds to one of these functions evaluated in $z, \lambda_1, \cdots, \lambda_d$.
- We derive some insights on the asymptotic behavior of the sign of the map $(\lambda_1, \cdots, \lambda_d) \mapsto \mathrm{val}(W^s_\Lambda(z))$ as $\lambda_d, \cdots, \lambda_1$ go to 0 respectively.
- We establish the connection between the stateful-discounted and limit value by the characterizations introduced in Theorem 9 and Theorem 11, and above insights. ◀

Novelty. As mentioned previously, our result is a generalization of [2]. The key non-trivial aspect of the generalization relies on the fact that [2] considers uni-variate polynomials, whereas our result requires analysis of multi-variate polynomials. Theorem 7 is the key mathematical foundation, and the proofs require significant technical generalization.

5 Algorithms for LimitValue and ParityValue

In this section, we present algorithms for computing ε-approximation of stateful-discounted, limit, and parity values. The section is organized as follows. In Section 5.1, we present an algorithm for computing ε-approximate stateful-discounted value. In Section 5.2, we present an algorithm for computing ε-approximate limit value, and as a consequence, we also obtain an algorithm for computing ε-approximate parity value.

5.1 Algorithm for Approximate Stateful-discounted Value

In this subsection, we present an algorithm for computing ε-approximation of the stateful-discounted value in CSGs. Given a CSG G, a reward function, and a discount function Λ, Algorithm 1 runs a binary search over the stateful-discounted value of state s. At the beginning, $\underline{z}$ and $\overline{z}$ are the under and over approximation of $\mathrm{val}_\Lambda(s)$. In each step, the algorithm halves the interval $[\underline{z}, \overline{z}]$ by increasing $\underline{z}$ or decreasing $\overline{z}$ based on the sign of $\mathrm{val}\left(W^s_\Lambda\left(\frac{\underline{z}+\overline{z}}{2}\right)\right)$. After $\mathrm{bit}(\varepsilon)$ steps, the algorithm outputs the ε-approximate value $(\overline{z}+\underline{z})/2$. The correctness and the time complexity of the algorithm are shown in Theorem 13.

Algorithm 1 ApproxDiscounted.

Input: Game G, state s, reward function r, a discount function Λ, additive error ε
Output: Approximate stateful-discounted value v such that $|v - \mathrm{val}_\Lambda(s)| \leq \varepsilon$

```
procedure ApproxDiscounted($G, s, r, \Lambda, \varepsilon$)
    $\underline{z} \leftarrow 0$ and $\overline{z} \leftarrow 1$
    while $\overline{z} - \underline{z} > \varepsilon$ do
        $z \leftarrow \frac{\underline{z}+\overline{z}}{2}$
        $\nu \leftarrow \mathrm{val}(W^s_\Lambda(z))$
        if $\nu \geq 0$ then $\underline{z} \leftarrow z$
        else $\overline{z} \leftarrow z$
    return $\frac{\underline{z}+\overline{z}}{2}$
```

▶ **Lemma 13.** *Consider a CSG G, a state s, a reward function r, a discount function Λ, and an additive error $\varepsilon > 0$. The transition function δ, the reward function r, and the discount function Λ are represented by rational numbers of bit-size* B. *Then, Algorithm 1 computes the ε-approximation of the stateful-discounted value of state s. Moreover, the algorithm runs in time* $\exp\left(\mathcal{O}(n\log(m) + \log(B) + \log(\log(1/\varepsilon))\right)$.

Proof sketch. The correctness of Algorithm 1 is due to the properties established in Theorem 8. The time complexity of Algorithm 1 is due to the construction of the matrix game and computing its value. ◀

5.2 Algorithms for Approximate Limit and Parity Values

In this subsection, we present an algorithm for computing ε-approximation of the limit and parity values in CSGs. Given a CSG G, a reward function, and an assignment function χ, Algorithm 2 outputs the $\varepsilon/2$-approximate of the stateful-discounted value of state s for some Λ_0 by calling ApproxDiscounted. By Theorem 12, the stateful-discounted value is an $\varepsilon/2$-approximation of the limit value. Thus, the returned value of the algorithm is indeed an ε-approximate of the limit value. The correctness and the time complexity of the algorithm is shown in Theorem 14. Since CSGs with parity objectives have a linear-size reduction to CSGs with the limit value of stateful-discounted objectives, as a consequence of the above algorithm, we obtain an algorithm for parity value approximation.

Algorithm 2 ApproxLimit.

Input: Game G, state s, reward function r, assignment function χ, additive error ε
Output: Approximate limit value v such that $|v - \mathrm{val}(s)| \leq \varepsilon$

```
procedure ApproxLimit($G, s, r, \chi, \varepsilon$)
    $D \leftarrow m^n$
    $\mathrm{B}_1 \leftarrow 11Dn\left(\mathrm{B} + \mathrm{bit}(n) + \mathrm{bit}(D) + \mathrm{bit}(\varepsilon)\right)$
    for $i \leftarrow 1$ to $d$ do
        $\lambda_i^0 \leftarrow \exp\left(-\mathrm{B}_1(nD+1)^{i-1}\right)$
      $v \leftarrow$ ApproxDiscounted($G, s, r, \Lambda^0, \varepsilon/2$)
    return $v$
```

▶ **Lemma 14.** *Consider a CSG G, a state s, a reward function r, an assignment function $\chi\colon \mathcal{S} \to [d]$, and an additive error $\varepsilon > 0$. The transition function δ and the reward function r are represented by rational numbers of bit-size* B. *Then, Algorithm 2 computes the ε-approximation of the limit value of state s. Moreover, the algorithm runs in time* $\exp\left(\mathcal{O}(nd\log(m) + \log(\mathrm{B}) + \log(\log(1/\varepsilon)))\right)$.

Proof sketch. The correctness of the algorithm is due to Theorem 13 and Theorem 12. The time complexity is due to the size of the representation of Λ^0 and Theorem 13. ◀

Proof of Theorem 5-Item 2. It is a direct implication of Theorem 14. ◀

▶ **Corollary 15.** *Consider a CSG G, a priority function p, a state s, and an additive error $\varepsilon > 0$. The transition function δ is represented by rational numbers of bit-size* B. *Then, there exists an algorithm that computes the ε-approximation of the parity value of state s. Moreover, the algorithm runs in time* $\exp\left(\mathcal{O}(nd\log(m) + \log(\mathrm{B}) + \log(\log(1/\varepsilon)))\right)$.

Proof. By [18, 10], there exists a linear-size reduction from the CSGs with parity objectives to the CSGs with the limit-value of stateful-discounted objectives. Therefore, the result follows from Theorem 14. ◀

Proof of Theorem 6-Item 2. It is a direct implication of Theorem 15. ◀

6 Computational Complexities of LimitValue and ParityValue

In this section, we show that the LimitValue and ParityValue problems are in TFNP[NP]. This section is organized as follows. In Section 6.1, we present some useful definitions related to Markov Chains (MCs) and Markov Decision Processes (MDPs), and floating-point representation. In Section 6.2, we present the complexity results for LimitValue and ParityValue problems.

6.1 Definitions

We present some basic notations and definitions related to Markov Chains, Markov Decision Processes, and the classic symbolic representation for numbers and probability distributions, called floating-point.

Markov decision processes and Markov chains. For $i \in \{1, 2\}$, a Player-i Markov decision process (Player-i MDP) is a special class of CSGs where the other player has only one action and is denoted by $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \delta\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S}))$. A Markov chain (MC) is a special class of MDPs where both players have only one action and is denoted by $\mathcal{C} = (\mathcal{S}, \delta\colon \mathcal{S} \to \Delta(\mathcal{S}))$. In Markov chains we write $\delta(s, s')$ to denote $\delta(s)(s')$.

Absorbing MCs. We say an MC $\mathcal{C}$ is *absorbing* if there exists a subset of absorbing states $\mathcal{S}_0 \subseteq \mathcal{S}$ such that

- For all $s \in \mathcal{S}_0$, we have $\delta(s, s) = 1$; and
- For all $s_0 \in \mathcal{S} \setminus \mathcal{S}_0$, there exist states $s_1, \ldots, s_k$ such that $\delta(s_i, s_{i+1}) > 0$ and $s_k \in \mathcal{S}_0$.

States in $\mathcal{S}_0$ are called absorbing.

MDPs and MCs given stationary strategies in CSGs. Given a stationary strategy σ for Player 1 in a game G, by fixing the strategy σ, we obtain a Player-2 MDP $G_\sigma = (S, \mathcal{B}, \delta_\sigma)$ where the transition function $\delta_\sigma \colon \mathcal{S} \times \mathcal{B} \to \Delta(\mathcal{S})$ is given by $\delta_\sigma(s,b)(s') \coloneqq \sum_{a \in \mathcal{A}} \delta(s,a,b)(s') \cdot \sigma(s)(a)$, for all $s, s' \in \mathcal{S}$ and $b \in \mathcal{B}$. Analogously, we obtain Player-1 MDP G_τ by fixing a stationary strategy τ for Player 2. Moreover, by fixing stationary strategies σ and τ for both players, we obtain an MC $G_{\sigma,\tau} = (\mathcal{S}, \delta_{\sigma,\tau})$, where the transition function $\delta_{\sigma,\tau} \colon \mathcal{S} \to \Delta(\mathcal{S})$ is given by $\delta_{\sigma,\tau}(s)(s') = \sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} \delta(s,a,b)(s') \cdot \sigma(s)(a) \cdot \tau(s)(b)$, for all $s, s' \in \mathcal{S}$.

Reachability objectives in MCs. Given an MC $\mathcal{C}$ and a target set $T \subseteq \mathcal{S}$, the reachability objective is the indicator function of plays eventually reaching T. More formally, for a play $\omega = \langle s_0, s_1, \cdots \rangle$, we define $\mathrm{Reach}_T \colon \Omega \to \{0,1\}$ as

$$\mathrm{Reach}_T(\omega) \coloneqq \begin{cases} 1 & \exists i \in \mathbb{N}\; s_i \in T \\ 0\,. & \text{otherwise} \end{cases}$$

We define the probability of reaching the target set T from state s as $\mathrm{val}_T(s) \coloneqq \mathbb{E}_s[\mathrm{Reach}_T]$.

Floating-point number representation. We define the set of floating-point numbers with precision ℓ as $\mathcal{F}(\ell) \coloneqq \left\{ m \cdot 2^e \;\middle|\; m \in \{0, \cdots, 2^\ell - 1\},\; e \in \mathbb{Z} \right\}$. The floating-point representation of an element $x = m \cdot 2^e \in \mathcal{F}(\ell)$ uses $\mathrm{bit}(m) + \mathrm{bit}(|e|)$ bits. We define the relative distance of two positive real numbers $x, \widetilde{x}$ as $\mathrm{rel}(x, \widetilde{x}) \coloneqq \max\left\{ \frac{x}{\widetilde{x}}, \frac{\widetilde{x}}{x} \right\} - 1$. We say x is (ℓ, i)-close to $\widetilde{x}$ if $\mathrm{rel}(x, \widetilde{x}) \leq (1 - 2^{1-\ell})^{-i} - 1$, where ℓ is a positive integer and i is a non-negative integer.

Floating-point probability distribution representation. We denote by $\mathcal{D}(\ell)$ the set of all floating-point probability distributions with precision ℓ. A probability distribution $\mu \in \Delta([t])$ belongs to $\mathcal{D}(\ell)$ if there exists $w_1, w_2, \cdots, w_t \in \mathcal{F}(\ell)$ such that

- For all $i \in [t]$, we have $\mu(i) = \frac{w_i}{\sum_{j \in [t]} w_j}$; and
- $\sum_{j \in [t]} w_j$ and 1 are (ℓ, t)-close.

We define the relative distance rel for probability distributions as $\mathrm{rel}(\mu, \widetilde{\mu}) \coloneqq \max\{\mathrm{rel}(\mu(i), \widetilde{\mu}(i)) : i \in [t]\}$. We say μ is (ℓ, i)-close to $\widetilde{\mu}$ if $\mathrm{rel}(\mu, \widetilde{\mu}) \leq (1 - 2^{1-\ell})^{-i} - 1$, where ℓ is a positive integer and i is a non-negative integer.

6.2 Complexity Results

We present algorithms for computing ε-approximate stateful-discounted value in MCs, MDPs, and CSGs, which generalize the result of [4] from a single discount factor to multiple discount factors. The algorithm for MCs (Theorem 16) is achieved by a reduction from stateful-discounted objective to reachability objectives in MCs and using the algorithm for computing reachability values in MCs presented in [17]. By our technical result on the limit value approximation via the stateful-discounted value (Theorem 12), we consequently obtain a TFNP[NP] procedure for LimitValue. Since there exists a linear-size reduction from CSGs with parity objectives to CSGs with the limit-value of stateful-discounted objectives [18, 10], ParityValue is also in TFNP[NP].

▶ **Lemma 16.** *Consider an MC $\mathcal{C}$, a reward function r, and a discount function Λ. For all $s \in \mathcal{S}$, we set*

$$\delta(s) \in D(\ell), \quad r(s) \in \mathcal{F}(\ell), \quad \Lambda(s) \in \mathcal{F}(\ell)\,,$$

where $\ell \geq 1000n^2$. Then, there exists a polynomial-time algorithm that for all states $s \in \mathcal{S}$, computes an approximation v for the stateful-discounted value such that $|v - \mathrm{val}_\Lambda(s)| \leq 104n^4 2^{-\ell}$.

Proof sketch. We construct a new MC $\widetilde{\mathcal{C}}$ with a reachability objective derived from the MC $\mathcal{C}$ such that the reachability value of $\widetilde{\mathcal{C}}$ is an approximation of the stateful-discounted value of $\mathcal{C}$. The result follows from the algorithm for computing ε-approximation of reachability value in MCs presented in [17]. ◀

▶ **Lemma 17.** *The problem of deciding if the stateful-discounted value for Player-1 MDPs is below a threshold up to an additive error is in NP where the input is a Player-1 MDP $\mathcal{P}$, a reward function r, a discount function Λ, a state s, a threshold $0 \leq \alpha \leq 1$, an additive error $\varepsilon = 2^{-\kappa}$ and a positive integer ℓ such that, for all $s' \in \mathcal{S}$ and $a \in \mathcal{A}$, we have*

$$\delta(s', a) \in D(\ell), \quad r(s', a) \in \mathcal{F}(\ell), \quad \Lambda(s') \in \mathcal{F}(\ell), \quad \ell \geq 1000n^2 + \kappa\,.$$

Note that, the numbers α and ε are represented in fixed-point binary and the NP procedure is such that

- *If $\alpha \leq \mathrm{val}_\Lambda(s) - \varepsilon$, then it outputs YES; and*
- *If $\alpha \geq \mathrm{val}_\Lambda(s) + \varepsilon$, then it outputs NO.*

Proof sketch. The procedure guesses a pure stationary strategy σ. By fixing σ, we obtain an MC $\mathcal{P}_\sigma$. The algorithm verifies whether the threshold can be approximately achieved if the player follows the strategy σ. The verification procedure is implemented by Theorem 16. ◀

Theorem 17 also holds for Player-2 MDPs by symmetric arguments. More formally, we have the following result.

▶ **Corollary 18.** *The problem of deciding if the stateful-discounted value for Player-2 MDPs is above a threshold up to an additive error is in NP where the input is a Player-2 MDP $\mathcal{P}$, a reward function r, a discount function Λ, a state s, a threshold $0 \leq \alpha \leq 1$, an additive error $\varepsilon = 2^{-\kappa}$ and a positive integer ℓ such that, for all $s' \in \mathcal{S}$ and $a \in \mathcal{A}$, we have*

$$\delta(s', a) \in D(\ell), \quad r(s', a) \in \mathcal{F}(\ell), \quad \Lambda(s') \in \mathcal{F}(\ell), \quad \ell \geq 1000n^2 + \kappa\,.$$

▶ **Lemma 19.** *The problem of computing an ε-approximation of the stateful-discounted value for CSGs is in TFNP[NP] for inputs CSGs G, reward functions, discount functions Λ, states s, additive errors $\varepsilon = 2^{-\kappa}$, and positive integers ℓ such that, for all states $s' \in \mathcal{S}$, we have $\Lambda(s') \in \mathcal{F}(\ell)$.*

Proof sketch. The procedure guesses two stationary strategies σ and τ and an approximate value α. By fixing the strategy σ (resp. τ), we obtain a Player-1 MDP G_σ (resp. Player-2 MDP G_τ). By the NP oracles calling the procedures defined in Theorems 17 and 18, the algorithm verifies whether the guessed threshold can be approximately achieved by both of the strategies. ◀

Proof of Theorem 5-Item 1. It is a direct implication of Theorem 19 and Theorem 12. ◀

Proof of Theorem 6-Item 1. By [18, 10], there exists a linear-size reduction from Parity-Value to LimitValue. Therefore, the result follows from Theorem 5-Item 1. ◀

Concluding remarks. In this work, we present improved complexity upper bounds and algorithms for the value approximation problem for concurrent stochastic games with two classic objectives. There are several interesting directions for future work. First, whether the complexity can be further improved from TFNP[NP] to TFNP is a major open question, even for reachability objectives. Second, whether for parity objectives, the dependency on d can be improved from linear to logarithmic, retaining the logarithmic dependence on m, is another interesting open question. Finally, the study of priority mean-payoff objectives for concurrent stochastic games and their connection to stateful-discounted objectives is another interesting direction for future work.

References

1 Rajeev Alur, Thomas A Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

2 Luc Attia and Miquel Oliu-Barton. A formula for the value of a stochastic game. *Proceedings of the National Academy of Sciences*, 116(52):26435–26443, 2019.

3 Saugata Basu. New results on quantifier elimination over real closed fields and applications to constraint databases. *Journal of the ACM (JACM)*, 46(4):537–555, 1999. doi:10.1145/320211.320240.

4 Sougata Bose, Rasmus Ibsen-Jensen, and Patrick Totzke. Bounded-memory strategies in partial-information games. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–14, 2024. doi:10.1145/3661814.3662096.

5 Krishnendu Chatterjee. *Stochastic ω-regular games.* University of California, Berkeley, 2007.

6 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A Henzinger. The complexity of quantitative concurrent parity games. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 678–687, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109631.

7 Krishnendu Chatterjee, Rupak Majumdar, and Thomas A Henzinger. Stochastic limit-average games are in EXPTIME. *International Journal of Game Theory*, 37(2):219–234, 2008. doi:10.1007/S00182-007-0110-5.

8 Luca de Alfaro and Thomas A Henzinger. Concurrent omega-regular games. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science*, pages 141–154. IEEE, 2000. doi:10.1109/LICS.2000.855763.

9 Luca de Alfaro, Thomas A Henzinger, and Orna Kupferman. Concurrent reachability games. In *Proceedings 39th Annual Symposium on Foundations of Computer Science*, pages 564–564. IEEE Computer Society, 1998.

10 Luca de Alfaro, Thomas A Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In *International Colloquium on Automata, Languages, and Programming*, pages 1022–1037. Springer, 2003. doi:10.1007/3-540-45061-0_79.

11 Luca de Alfaro, Thomas A Henzinger, and Freddy YC Mang. The control of synchronous systems. In *International Conference on Concurrency Theory*, pages 458–473. Springer, 2000.

12 Luca de Alfaro, Thomas A Henzinger, and Freddy YC Mang. The control of synchronous systems, part II. In *International Conference on Concurrency Theory*, pages 566–581. Springer, 2001.

13 Luca de Alfaro and Rupak Majumdar. Quantitative solution of omega-regular games. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 675–683, 2001. doi:10.1145/380752.380871.

14 Kousha Etessami and Mihalis Yannakakis. Recursive concurrent stochastic games. *Logical Methods in Computer Science*, 4, 2008. doi:10.2168/LMCS-4(4:7)2008.

15 Hugh Everett. Recursive games. *Contributions to the Theory of Games*, 3(39):47–78, 1957.

16 Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes.* Springer-Verlag, 1997.

17 Søren Kristoffer Stiil Frederiksen and Peter Bro Miltersen. Approximating the value of a concurrent reachability game in the polynomial time hierarchy. In *International Symposium on Algorithms and Computation*, pages 457–467. Springer, 2013. doi:10.1007/978-3-642-45030-3_43.

18 Hugo Gimbert and Wiesław Zielonka. Discounting infinite games but how and why? *Electronic Notes in Theoretical Computer Science*, 119(1):3–9, 2005. doi:10.1016/J.ENTCS.2004.07.005.

19 Hugo Gimbert and Wiesław Zielonka. Blackwell optimal strategies in priority mean-payoff games. *International Journal of Foundations of Computer Science*, 23(03):687–711, 2012. doi:10.1142/S0129054112400345.

20 Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. The complexity of solving reachability games using value and strategy iteration. In *International Computer Science Symposium in Russia*, pages 77–90. Springer, 2011. doi:10.1007/978-3-642-20712-9_7.

21 Kristoffer Arnsfelt Hansen, Michal Koucky, Niels Lauritzen, Peter Bro Miltersen, and Elias P Tsigaridas. Exact algorithms for solving stochastic games. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 205–214, 2011.

22 Kristoffer Arnsfelt Hansen, Michal Koucky, and Peter Bro Miltersen. Winning concurrent reachability games requires doubly-exponential patience. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 332–341. IEEE, 2009. doi:10.1109/LICS.2009.44.

23 Donald A Martin. The determinacy of Blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998. doi:10.2307/2586667.

24 Jean-François Mertens and Abraham Neyman. Stochastic games. *International Journal of Game Theory*, 10:53–66, 1981.

25 Miquel Oliu-Barton. New algorithms for solving zero-sum stochastic games. *Mathematics of Operations Research*, 46(1):255–267, 2021. doi:10.1287/MOOR.2020.1055.

26 Lloyd Stowell Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

27 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages: Volume 3 Beyond Words*, pages 389–455. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.

A Decomposition Approach to the Weighted k-Server Problem

Nikhil Ayyadevara
University of Michigan, Ann Arbor, MI, USA

Ashish Chiplunkar
Indian Institute of Technology, New Delhi, India

Amatya Sharma
University of Michigan, Ann Arbor, MI, USA

Abstract

A natural variant of the classical online k-server problem is the *weighted k-server problem*, where the cost of moving a server is its weight times the distance through which it moves. Despite its apparent simplicity, the weighted k-server problem is extremely poorly understood. Specifically, even on uniform metric spaces, finding the optimum competitive ratio of randomized algorithms remains an open problem – the best upper bound known is $2^{2^{k+O(1)}}$ due to a deterministic algorithm (Bansal et al., 2018), and the best lower bound known is $\Omega(2^k)$ (Ayyadevara and Chiplunkar, 2021).

With the aim of closing this exponential gap between the upper and lower bounds, we propose a decomposition approach for designing a randomized algorithm for weighted k-server on uniform metrics. Our first contribution includes two relaxed versions of the problem and a technique to obtain an algorithm for weighted k-server from algorithms for the two relaxed versions. Specifically, we prove that if there exists an α_1-competitive algorithm for one version (which we call *Weighted k-Server – Service Pattern Construction*) and there exists an α_2-competitive algorithm for the other version (which we call *Weighted k-server – Revealed Service Pattern*), then there exists an $(\alpha_1\alpha_2)$-competitive algorithm for weighted k-server on uniform metric spaces. Our second contribution is a $2^{O(k^2)}$-competitive randomized algorithm for Weighted k-server – Revealed Service Pattern. As a consequence, the task of designing a $2^{\mathrm{poly}(k)}$-competitive randomized algorithm for weighted k-server on uniform metrics reduces to designing a $2^{\mathrm{poly}(k)}$-competitive randomized algorithm for Weighted k-Server – Service Pattern Construction. Finally, we also prove that the $\Omega(2^k)$ lower bound for weighted k-server, in fact, holds for Weighted k-server – Revealed Service Pattern.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Caching and paging algorithms

Keywords and phrases Online Algorithms, k-server, paging

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.6

1 Introduction

The k-server problem proposed by Manasse et al. [12] is a fundamental problem in online computation, and it has been actively studied for over three decades. In this problem, we are given a metric space M and k identical servers $s_1, \ldots, s_k$ located at points of M. In every round, a point of M is requested, and an online algorithm serves the request by moving (at least) one server to the requested point. The objective is to minimize the total distance traversed by all k servers.

Like several other online problems, the performance of algorithms for the k-server problem is measured using the framework of competitive analysis introduced by Sleator and Tarjan [15]. An online algorithm for a minimization problem is said to be α-competitive if, on every input, the ratio of the algorithm's (expected) cost to the cost of the optimal solution is at most α, possibly modulo an additive constant independent of the online input. In the deterministic setup, Manasse et al. [12] showed that no k-server algorithm can be better than

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 6; pp. 6:1–6:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

k-competitive on any metric space with more than k points. In their breakthrough result, Koutsoupias and Papadimitriou [11] gave the best known deterministic algorithm that is $(2k-1)$-competitive on every metric space, famously known as the Work Function Algorithm (WFA). In the setup of randomized algorithms, it is conjectured that the competitive ratio of k-server is $O(\text{poly}(\log k))$, and this remains unsolved. Very recently, refuting the so-called *randomized k-server conjecture*, Bubeck, Coester, and Rabani [6] exhibited a family of metric spaces on which the randomized competitive ratio of the k-server problem is $\Omega(\log^2 k)$.

The k-server problem is a generalization of the online paging problem. The paging problem concerns maintaining in a "fast" memory a subset of k pages out of the n pages in a "slow" memory. In each round, one of n ($\gg k$) pages is requested, and it must replace some page in the fast memory, unless it is already in the fast memory. The objective is to minimize the number of page replacements. The paging problem is exactly the k-server under the uniform metric on the set of pages. The paging problem has been well studied, and several deterministic algorithms like Least Recently Used (LRU), First In First Out (FIFO), etc. are known to be k-competitive [15]. The randomized algorithm by Achlioptas et al. [1] is known to be $H(k)$-competitive, matching the lower bound by Fiat et al. [8]. Here $H(k) = 1 + 1/2 + \cdots + 1/k = \Theta(\log k)$.

1.1 Weighted k-server

The weighted k-server problem, first defined by Newberg [13], is a natural generalization of the k-server problem. In the weighted k-server problem, the servers are distinguishable: the i'th server has weight w_i, where $w_1 \leq \cdots \leq w_k$. The cost incurred in moving a server is its weight times the distance it travels. The objective is to minimize the total weighted distance moved by all k servers. It is easy to see that an α-competitive algorithm for the (unweighted) k-server problem has a competitive ratio of at most $\alpha \cdot w_k/w_1$ for the weighted k-server problem. However, this bound can be arbitrarily bad as w_k/w_1 is unbounded. So, the challenge is to establish weight-independent bounds on the competitive ratio of the weighted k-server problem. Surprisingly, this simple introduction of weights makes this problem incredibly difficult, and a weight-independent upper bound on the competitive ratio for an arbitrary metric is only known for the case when $k \leq 2$ [14].

Owing to its difficulty on general metric spaces, it is natural to completely understand the weighted k-server problem on the simplest class of metric spaces, the uniform metric spaces first. Uniform metric spaces are the ones in which every pair of points is separated by a unit distance. The objective of this problem translates to minimizing the weighted sum of the number of movements of each server, and thus, this problem is equivalent to paging with the cost of a page replacement dependent on the cache slot it is stored in[1]. In their seminal work, Fiat and Ricklin [9] gave a deterministic algorithm for the weighted k-server problem on uniform metrics with a competitive ratio doubly exponential in k, which was later improved by Bansal et al. [4] to $2^{2^{k+2}}$. This doubly exponential behavior of the competitive ratio was proven tight by Bansal et al. [3] when they showed that the deterministic competitive ratio is no less than $2^{2^{k-4}}$.

In the randomized setup, the only known algorithm which uses randomization in a non-trivial manner is the memoryless algorithm of Chiplunkar and Vishwanathan [7], which has a competitive ratio of 1.6^{2^k}. Chiplunkar and Vishwanathan also showed that this ratio is tight

[1] Note that this problem is different from weighted paging [16], where the weights are on the pages (points in the metric space) instead on the cache slots (servers). In fact, weighted paging is equivalent to unweighted k-server on star metrics.

for the class of randomized memoryless algorithms. Recently, Ayyadevara and Chiplunkar [2] showed that no randomized algorithm (memoryless or otherwise) can achieve a competitive ratio better than $\Omega(2^k)$. Closing the exponential gap between the 1.6^{2^k} upper bound and the $\Omega(2^k)$ lower bound on the randomized competitive ratio is still an open problem.

Very recently, Gupta et al. [10] studied the weighted k-server problem in the offline and resource augmentation settings, showing the first hardness of approximation result for polynomial-time algorithms.

1.2 Our Contributions

Throughout this paper, we focus on the weighted k-server problem on uniform metrics, and we avoid mentioning the metric space henceforward. Considering the fact that the competitive ratio of a server problem is typically exponentially better in the randomized setting than the deterministic setting, it is reasonable to conjecture that there exists a randomized $2^{\text{poly}(k)}$-competitive randomized algorithm for weighted k-server. In this paper, we propose a way of designing such an algorithm using our key idea of decomposing the weighted k-server problem into the following two relaxed versions.

Weighted k-Server – Service Pattern Construction (WkS-SPC)

The input is the same as the weighted k-server. The difference is that, in response to each request, the algorithm must only commit to the movement of some subset of servers, without specifying where those servers move to. However, it is required that there exists some solution to the given instance that agrees with the algorithm's server movements. Note that the algorithm could potentially benefit from not being lazy, that is, by moving more than one server at the same time. The (expected) cost of the algorithm is, as defined earlier, the weighted sum of the number of movements of each server (recall that we are working on a uniform metric space so the distance between every pair of points is one unit). An algorithm is said to be α-competitive if the (expected) cost of its solution is at most α times the optimum cost.

Weighted k-Server – Revealed Service Pattern (WkS-RSP)

In this version, the adversary, in addition to giving requests, is obliged to help the algorithm by providing additional information as follows. The adversary must serve each request and reveal to the algorithm the subset of servers it moved. Note that the adversary does not reveal the destination to which it moved its servers – revealing destinations makes the problem trivial because the algorithm can simply copy the adversary's movements. Given a request and the additional information about the adversary's server movements, the algorithm is required to move its own servers to cover the request. In an ideal scenario where the adversary serves the requests optimally, we require the algorithm to produce a solution whose cost competes with the cost of the optimal solution. However, consider a malicious adversary which, in an attempt to be as unhelpful to the algorithm as possible, produces a far-from-optimum solution and shares its information with the algorithm. In this case, we do not require that the algorithm competes with the optimum solution – such an algorithm would already solve the weighted k-server problem without the adversary's help. Instead, we require the algorithm to compete with the adversary's revealed solution. Formally, an algorithm is said to be α-competitive if the (expected) cost of its output is at most α times the cost of the adversary's (possibly sub-optimal) solution.

Obviously, an algorithm for the weighted k-server problem gives an algorithm for each of the above problems. Interestingly, we prove that the converse is also true. Formally,

▶ **Theorem 1** (Composition Theorem). *If there exists an α_1-competitive algorithm for WkS-SPC and there exists an α_2-competitive algorithm for WkS-RSP, then there is an $(\alpha_1\alpha_2)$-competitive algorithm for the weighted k-server on uniform metrics.*

We prove this theorem in Section 3. As a consequence of this theorem, it is enough to design $2^{\mathrm{poly}(k)}$-competitive algorithms for WkS-SPC and WkS-RSP to close the exponential gap between the upper and lower bounds on the randomized competitive ratio of weighted k-server. We already present such an algorithm for WkS-RSP in Section 4. We prove,

▶ **Theorem 2.** *There is a randomized algorithm for WkS-RSP with a competitive ratio of $2^{O(k^2)}$.*

This reduces the task of designing a $2^{\mathrm{poly}(k)}$-competitive algorithm for weighted k-server to designing such an algorithm for WkS-SPC, a potentially easier problem.

Can we improve the $2^{O(k^2)}$ upper bound for WkS-RSP to, for example, poly(k)? We answer this question in the negative. We show that, in fact, the lower bound construction by Ayyadevara and Chiplunkar [2] for weighted k-server applies to WkS-RSP too[2], giving the following result.

▶ **Theorem 3.** *The randomized competitive ratio of WkS-RSP is $\Omega(2^k)$.*

The proof of this result is deferred to Section A.

2 Preliminaries

In this section we define the problems WkS-SPC and WkS-RSP formally, but before that, we restate the definitions of some terms introduced by Bansal et al. [3] which will be needed in our problem definitions.

2.1 Service Patterns, Feasible Labelings, Extensions

Throughout this paper, we assume without loss of generality that all the servers of the algorithm and the adversary move in response to the first request. Given a solution to an instance of the weighted k-server problem with T requests, focus on the movements of the ℓ'th server for an arbitrary ℓ. The time instants at which these movements take place partition the interval $[1, T+1)$ into left-closed-right-open intervals so that the server stays put at some point during each of these intervals. Thus, ignoring the locations of the servers and focusing only on the time instants at which each server moves, we get a tuple of k partitions of $[1, T+1)$, also known as a service pattern. Formally,

▶ **Definition 4** (Service Pattern and Levels [3]). *A k-tuple $\mathcal{I} = (\mathcal{I}^1, \ldots, \mathcal{I}^k)$ is called a service pattern over an interval $[t_{begin}, t_{end})$ if each $\mathcal{I}^\ell$ is a partition of $[t_{begin}, t_{end})$ comprising of left-closed-right-open intervals with integer boundaries. We call $\mathcal{I}^\ell$ the ℓ'th level of $\mathcal{I}$.*

[2] The construction by Bansal et al. [3] applies too, and for the same reason, implying a doubly exponential lower bound on the deterministic competitive ratio of WkS-RSP.

Observe that the cost of a solution is completely determined by its service pattern $\mathcal{I} = (\mathcal{I}^1, \dots, \mathcal{I}^k)$: the cost equals the sum over $\ell \in \{1, \dots, k\}$ of the number of intervals in $\mathcal{I}^\ell$ times the weight of the ℓ'th server.

In order to completely specify a solution, in addition to a service pattern $\mathcal{I} = (\mathcal{I}^1, \dots, \mathcal{I}^k)$, we need to specify for each ℓ and each interval $I \in \mathcal{I}^\ell$ the location of the ℓ'th server during the time interval I. We refer to this assignment as a labeling of the service pattern. Moreover, to serve the t'th request σ_t, we need at least one server to occupy σ_t at time t, that is, we need that there exists a level of $\mathcal{I}$ in which the (unique) interval containing t is labeled σ_t. Such a labeling is called a feasible labeling. Formally,

▶ **Definition 5** (Labeling and Feasibility [3]). *A labeling of a service pattern $\mathcal{I} = (\mathcal{I}^1, \dots, \mathcal{I}^k)$ is a function from the multi-set $\mathcal{I}^1 \uplus \dots \uplus \mathcal{I}^k$ to the set U of points in the metric space. We say that a labeling γ of $\mathcal{I}$ is feasible with respect to a request sequence $\rho = (\sigma_1, \dots, \sigma_T)$, if for each time t, there exists an interval $I \in \mathcal{I}^1 \uplus \dots \uplus \mathcal{I}^k$ containing t such that $\gamma(I) = \sigma_t$. We say that a service pattern $\mathcal{I}$ is feasible with respect to ρ if there exists a feasible labeling of $\mathcal{I}$ with respect to ρ.*

Recall that in the definition of the weighted k-server problem in Section 1, we assumed that the servers are numbered in a non-decreasing order of their weights. Consider the more interesting case where the weights increase at least geometrically. If we enforce that every time a server moves, all servers lighter than it move too, we lose at most a constant factor in the competitive ratio. The advantage of this enforcement is that we have a more structured class of service patterns, called hierarchical service patterns.

▶ **Definition 6** (Hierarchical Service Pattern [3]). *A service pattern $\mathcal{I} = (\mathcal{I}^1, \dots, \mathcal{I}^k)$ is hierarchical if for every $\ell \in \{1, \dots, k-1\}$, the partition $\mathcal{I}^\ell$ refines the partition $\mathcal{I}^{\ell+1}$.*

Clearly, any service pattern can be made hierarchical in an online manner with at most a k factor loss in the cost. Since we aim to obtain $2^{\text{poly}(k)}$-competitive algorithms, the k factor loss is affordable, and therefore, we only consider hierarchical service patterns throughout this paper. Henceforth, by service pattern we actually mean a hierarchical service pattern.

Next, consider some online algorithm for the weighted k-server problem and the solution it outputs on some request sequence. For each t, let $\mathcal{I}_t$ denote the service pattern corresponding to the algorithm's solution until the t'th request. Observe that $\mathcal{I}_{t-1}$ and $\mathcal{I}_t$ are closely related: if the algorithm moves the lightest ℓ servers to serve the t'th request (possibly $\ell = 0$), then the interval $[t, t+1)$ gets added to the first ℓ levels of $\mathcal{I}_{t-1}$, whereas in each of the remaining levels, the last interval in the level merges with $[t, t+1)$. We call $\mathcal{I}_t$ the ℓ-extension of $\mathcal{I}_{t-1}$. More formally,

▶ **Definition 7** (ℓ-extension). *Let $\mathcal{I}_{t-1} = (\mathcal{I}^1_{t-1}, \dots, \mathcal{I}^k_{t-1})$ be a hierarchical service pattern over the interval $[1, t)$, and let L^i_{t-1} be the last interval in $\mathcal{I}^i_{t-1}$. For $\ell \in \{0, \dots, k\}$, we define the ℓ-extension of $\mathcal{I}_{t-1}$ to be the service pattern $\mathcal{I}_t = (\mathcal{I}^1_t, \dots, \mathcal{I}^k_t)$ over the interval $[1, t+1)$ where:*

- $\forall i \leq \ell,\ \mathcal{I}^i_t = \mathcal{I}^i_{t-1} \cup \{[t, t+1)\}$.
- $\forall i > \ell,\ \mathcal{I}^i_t = (\mathcal{I}^i_{t-1} \setminus L^i_{t-1}) \cup \{L^i_{t-1} \cup [t, t+1)\}$.

Observe that the ℓ-extension of a hierarchical service pattern is a hierarchical service pattern.

2.2 Problem Definitions

Recall that our core idea to solve weighted k-server problem is to construct an algorithm using algorithms for its two relaxed versions. We defined them informally in Section 1. Their formal definitions are as follows.

▶ **Definition 8** (Weighted k-Server – Service Pattern Construction (WkS-SPC)). *For every online request $\sigma_t \in U$, an algorithm for WkS-SPC is required to output a service pattern $\mathcal{I}_t$, which is the ℓ_t-extension of $\mathcal{I}_{t-1}$ for some $\ell_t \in \{0, \ldots, k\}$, such that $\mathcal{I}_t$ is feasible with respect to the request sequence $\sigma_1, \sigma_2, \ldots, \sigma_t$. Equivalently, the algorithm outputs ℓ_t for each t. An algorithm for WkS-SPC is said to be α-competitive if the (expected) cost of the algorithm's service pattern is at most α times the optimal cost.*

▶ **Definition 9** (Weighted k-server – Revealed Service Pattern (WkS-RSP)). *For every online request $\sigma_t \in U$, the adversary reveals a service pattern $\mathcal{I}_t$, which is the ℓ_t-extension of $\mathcal{I}_{t-1}$ for some $\ell_t \in \{0, \ldots, k\}$, such that $\mathcal{I}_t$ is feasible with respect to the request sequence $\sigma_1, \sigma_2, \ldots, \sigma_t$. Equivalently, the algorithm's input is the pair (σ_t, ℓ_t). An algorithm for WkS-RSP is required to serve the request σ_t, i.e., move servers to ensure that σ_t is covered by some server. An algorithm for WkS-RSP is said to be β-competitive if the (expected) cost of the algorithm's solution is at most β times the cost of the final service pattern revealed by the adversary.*

3 The Composition Theorem

In this section, we explain how we can construct a weighted k-server algorithm using algorithms for its two relaxations – WkS-SPC and WkS-RSP[3].

▶ **Theorem 1** (Composition Theorem). *If there exists an α_1-competitive algorithm for WkS-SPC and there exists an α_2-competitive algorithm for WkS-RSP, then there is an $(\alpha_1\alpha_2)$-competitive algorithm for the weighted k-server on uniform metrics.*

Proof. Let $\mathcal{A}_1$ be an α_1-competitive algorithm for WkS-SPC, and $\mathcal{A}_2$ be an α_2-competitive algorithm for WkS-RSP. Our algorithm $\mathcal{A}$ for weighted k-server internally runs the two algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. At all times, $\mathcal{A}$ keeps each of its servers at the same point where the corresponding server of $\mathcal{A}_2$ is located. For every input request σ_t, $\mathcal{A}$ performs the following sequence of steps.

1. $\mathcal{A}$ passes σ_t to $\mathcal{A}_1$.
2. In response, $\mathcal{A}_1$ outputs an ℓ_t such that the service pattern $\mathcal{I}_t$, which is the ℓ_t-extension to $\mathcal{I}_{t-1}$, is feasible for the request sequence $\sigma_1, \sigma_2, \ldots, \sigma_t$.
3. $\mathcal{A}$ passes (σ_t, ℓ_t) to $\mathcal{A}_2$.
4. In response, $\mathcal{A}_2$ moves its servers to serve the request σ_t.
5. $\mathcal{A}$ copies the movements of $\mathcal{A}_2$'s servers.

To analyze the competitiveness of $\mathcal{A}$, consider an arbitrary sequence ρ of requests, and let T denote its length. Let OPT denote the cost of an optimal solution for ρ. Denote the cost of a service pattern $\mathcal{I}$ by $\text{cost}(\mathcal{I})$. Recall that $\mathcal{I}_T$ is the final service pattern output by $\mathcal{A}_1$. Let $\mathcal{I}'_T$ denote the service pattern corresponding to $\mathcal{A}_2$'s output. Note that $\mathcal{I}_T$ and $\mathcal{I}'_T$ are random variables, and since $\mathcal{A}$'s output is same as $\mathcal{A}_2$'s output, the cost of $\mathcal{A}$'s output is $\text{cost}(\mathcal{I}'_T)$.

[3] On a high level, our construction resembles the result by Ben-David et al. [5], which states that if there is an α_1-competitive randomized algorithm against online adversary and an α_2-competitive algorithm against any oblivious adversary, then there is an $(\alpha_1\alpha_2)$ competitive randomized algorithm for any adaptive offline adversary.

Since the output of $\mathcal{A}_1$ is a sequence of extensions such that the service pattern remains feasible with respect to the request sequence at all times, the sequence $(\sigma_t, \ell_t)_{t=1,\dots,T}$ is a valid instance of WkS-RSP (with probability one over the randomness of $\mathcal{A}_1$). Since $\mathcal{A}_2$ is α_2-competitive, we have $\mathbb{E}[\text{cost}(\mathcal{I}'_T) \mid \mathcal{I}_T = \mathcal{I}] \le \alpha_2 \cdot \text{cost}(\mathcal{I})$ for every service pattern $\mathcal{I}$ feasible with respect to ρ. This implies $\mathbb{E}[\text{cost}(\mathcal{I}'_T)] \le \alpha_2 \cdot \mathbb{E}[\text{cost}(\mathcal{I}_T)]$. Since $\mathcal{A}_1$ is α_1-competitive, $\mathbb{E}[\text{cost}(\mathcal{I}_T)] \le \alpha_1 \cdot \text{OPT}$. Thus, $\mathbb{E}[\text{cost}(\mathcal{I}'_T)] \le \alpha_1\alpha_2 \cdot \text{OPT}$. This implies that $\mathcal{A}$ is $(\alpha_1\alpha_2)$-competitive. ◀

4 Competing with a Revealed Service Pattern

We organize this section as follows. In Section 4.1, we prove some structural results that are used in the definition and analysis of our algorithm. We define our algorithm formally in Section 4.2 and analyze its competitive ratio in Section 4.3. We use the following notation.

- U denotes the set of points in a uniform metric space.
- For t from 1 to T, The t'th request is $\sigma_t \in U$, and $\rho_t = (\sigma_1, \dots, \sigma_t)$ denotes the sequence of requests received until time t.
- The service pattern revealed by the adversary with the t'th request is denoted by $\mathcal{I}_t = (\mathcal{I}_t^1, \dots, \mathcal{I}_t^k)$. Recall that $\mathcal{I}_t$ is the ℓ_t-extension of $\mathcal{I}_{t-1}$. Without loss of generality, we assume that the adversary moves all its servers with the first request, and therefore $\ell_1 = k$.
- L_t^ℓ denotes the last interval in $\mathcal{I}_t^\ell$, that is, the unique interval in $\mathcal{I}_t^\ell$ that covers $[t, t+1)$.
- s_t^ℓ denotes the location of our algorithm's ℓ'th server after processing the t'th request. Since our algorithm is randomized, s_t^ℓ is a random variable. Note that for the t'th request to be served, we must have $\sigma_t \in \{s_t^1, \dots, s_t^k\}$ with probability one.

Consider the adversary's service pattern $\mathcal{I}_t = (\mathcal{I}_t^1, \dots, \mathcal{I}_t^k)$. For an arbitrary ℓ, fix the labels of the last intervals $L_t^{\ell+1}, \dots, L_t^k$ in the top $k - \ell$ levels $\mathcal{I}_t^{\ell+1}, \dots, \mathcal{I}_t^k$ of $\mathcal{I}_t$, and consider all feasible labelings of $\mathcal{I}_t$ with respect to ρ that agree with the fixed labels. The set of labels that these labelings assign to the last interval L_t^ℓ of $\mathcal{I}_t^\ell$ will be crucial for our algorithm. We now define this set formally.

▶ **Definition 10.** *For any $t \in \{1, \dots, T\}$, $\ell \in \{1, \dots, k\}$, and $p^{\ell+1}, \dots, p^k \in U$, the set $Q_t^\ell(p^{\ell+1}, \dots, p^k)$ is defined to be the set of points p^ℓ for which there exists a feasible labeling γ of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma(L_t^i) = p^i$ for all $i \in \{\ell, \dots, k\}$.*

4.1 Structural Results

Bansal et al. [3] considered the following combinatorial question: given a service pattern $\mathcal{I}$ and a request sequence ρ, how many labels can an interval in the k'th level of $\mathcal{I}$ get, over all possible feasible labelings of $\mathcal{I}$ with respect to ρ? They derived the following interesting property.

▶ **Fact 11** (Dichotomy Property [3]). *There exists a sequence $n_1, n_2, \dots$ of integers with $n_k \le 2^{2^{k+3\log k}}$ such that the following holds: for every k, every sequence of requests $\rho = (\sigma_1, \dots, \sigma_T)$, every service pattern $\mathcal{I} = (\mathcal{I}^1, \dots, \mathcal{I}^k)$ over $[1, T+1)$, and every $I \in \mathcal{I}^k$, the set Q of labels of I over all feasible labelings of $\mathcal{I}$ with respect to ρ is the entire U, or it has size at most n_k.*

The next lemma generalizes the above result to intervals in every level.

▶ **Lemma 12** (Generalized Dichotomy Property). *For every $t \in \{1, \dots, T\}$, $\ell \in \{1, \dots, k\}$, and $p^{\ell+1}, \dots, p^k \in U$, the set $Q^\ell_t(p^{\ell+1}, \dots, p^k)$ is the entire U, or it has size at most n_ℓ, where $n_\ell \le 2^{2^{\ell+3\log \ell}}$ is the constant from Fact 11.*

Proof. The lemma holds trivially when $Q^\ell_t(p^{\ell+1}, \dots, p^k) = \emptyset$, so assume $Q^\ell_t(p^{\ell+1}, \dots, p^k) \neq \emptyset$. Let ρ' denote the request sequence during the interval L^ℓ_t and $\mathcal{J}$ denote the restriction of the service pattern $\mathcal{I}_t$ to the interval L^ℓ_t and levels $1, \dots, \ell$. Let ρ'' denote the subsequence of ρ' formed by removing all the requests to $p^{\ell+1}, \dots, p^k$. Let Q denote the set of labels of the interval L^ℓ_t over all the feasible labelings of the ℓ-level service pattern $\mathcal{J}$ with respect to ρ''. From Fact 11 we get that the set Q is either U or has size at most n_ℓ. We argue that $Q^\ell_t(p^{\ell+1}, \dots, p^k) = Q$, and this implies the claim. Refer to Figure 1 for a working illustration on an instance with $k = 5$ and $\ell = 3$.

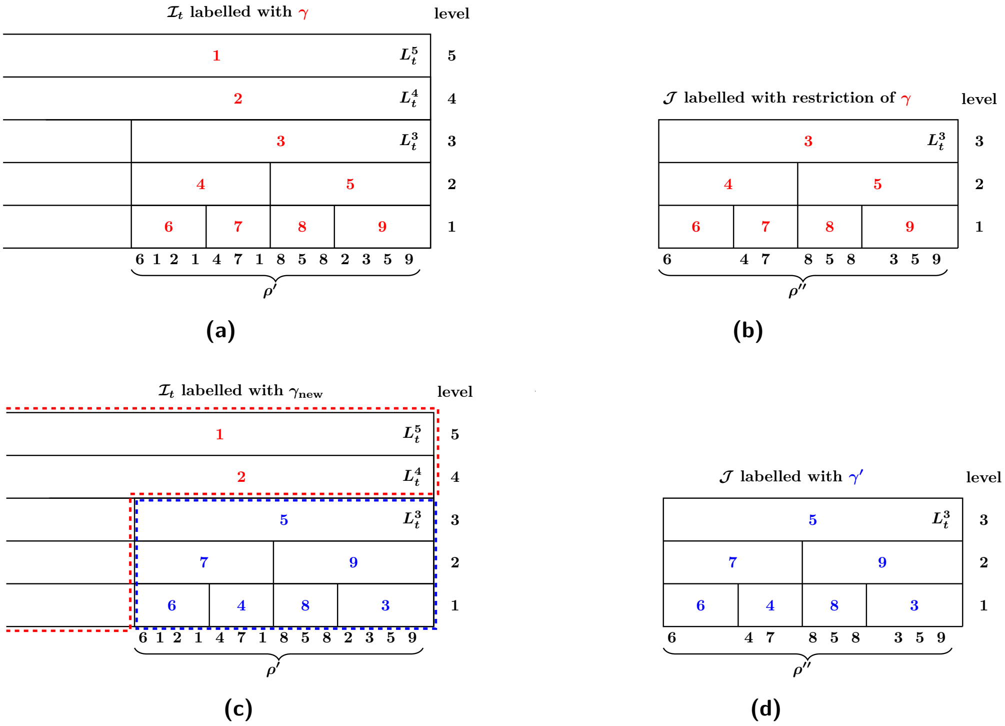


Figure 1 An illustration of Lemma 12 for $k = 5$ and $\ell = 3$. (a) Depicts $\mathcal{I}_t$ with a labeling γ (colored in red) feasible with respect to ρ_t such that $\gamma(L^5_t) = 1$ and $\gamma(L^4_t) = 2$. (b) Depicts the 3-level service pattern $\mathcal{J}$ labeled with the restriction of γ, along with ρ'', the subsequence of ρ' formed by removing all the requests to points 1 and 2. (d) Depicts the labeling γ' of $\mathcal{J}$ feasible with respect to ρ''. (c) Shows the new labeling γ_{new} constructed by overwriting γ' onto γ for every interval in $\mathcal{J}$.

For any $p^\ell \in Q^\ell_t(p^{\ell+1}, \dots, p^k)$, from Definition 10 we get that there exists a feasible labeling γ of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma(L^i_t) = p^i$ for all $i \in \{\ell, \dots, k\}$. In the labeling γ, the servers $\ell + 1, \dots, k$ can only serve the requests to the points $p^{\ell+1}, \dots, p^k$ during the interval L^ℓ_t. This implies that all the requests in ρ'' must be served by servers $1, \dots, \ell$. Thus, the restriction γ' of γ to $\mathcal{J}$ is feasible with respect to ρ''. But $\gamma'(L^\ell_t) = \gamma(L^\ell_t) = p^\ell$. Therefore, $p^\ell \in Q$. Thus, $Q^\ell_t(p^{\ell+1}, \dots, p^k) \subseteq Q$.

Now consider any point $p^\ell \in Q$, and let γ' be a feasible labeling of $\mathcal{J}$ with respect to ρ'' such that $\gamma'(L_t^\ell) = p^\ell$. Suppose γ is a feasible labeling of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma(L_t^i) = p^i$ for all $i \in \{\ell+1, \ldots, k\}$ (such a labeling exists because we assumed $Q_t^\ell(p^{\ell+1}, \ldots, p^k) \neq \emptyset$). Overwrite the labeling γ' onto γ to get a new labeling γ_{new}. Formally, $\gamma_{\text{new}}(I) = \gamma'(I)$ if interval I is in $\mathcal{J}$, else $\gamma_{\text{new}}(I) = \gamma(I)$. We claim that γ_{new} is also a feasible labeling of $\mathcal{I}_t$ with respect to ρ_t. This can be argued as follows.

The labeling γ_{new} serves all the requests in ρ'' because it agrees with γ' in the service pattern $\mathcal{J}$. γ_{new} serves all the requests during the interval L_t^ℓ other than those in ρ'' because all these requests are made at points in $\{p^{\ell+1}, \ldots, p_k\}$, and $\gamma_{\text{new}}(L_t^i) = \gamma(L_t^i) = p^i$ for all $i \in \{\ell+1, \ldots, k\}$. Finally, γ_{new} serves all the requests before the interval L_t^ℓ because it agrees with γ before the interval L_t^ℓ.

Thus, γ_{new} is a feasible labeling of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma_{\text{new}}(L_t^i) = p^i$ for all $i \in \{\ell+1, \ldots, k\}$. From Definition 10, we get that $\gamma_{\text{new}}(L_t^\ell) = p^\ell \in Q_t^\ell(p^{\ell+1}, \ldots, p^k)$. This implies $Q \subseteq Q_t^\ell(p^{\ell+1}, \ldots, p^k)$. ◀

We now state a useful consequence of the simple fact that the restriction of a solution for the first t requests to the first $t-1$ requests is a solution for the first $t-1$ requests.

▶ **Lemma 13.** *For every $t \in \{2, \ldots, T\}$, $\ell \in \{\ell_t + 1, \ldots, k\}$, and $p^{\ell+1}, \ldots, p^k \in U$, $Q_t^\ell(p^{\ell+1}, \ldots, p^k) \subseteq Q_{t-1}^\ell(p^{\ell+1}, \ldots, p^k)$.*

Proof. Recall that $\mathcal{I}_t = (\mathcal{I}_t^1, \ldots, \mathcal{I}_t^k)$ is the ℓ_t-extension of $\mathcal{I}_{t-1} = (\mathcal{I}_{t-1}^1, \ldots, \mathcal{I}_{t-1}^k)$, which means $L_{t-1}^\ell = L_t^\ell \setminus [t, t+1) \neq \emptyset$. By Definition 10, if some point p^ℓ is in $Q_t^\ell(p^{\ell+1}, \ldots, p^k)$, then there exists a feasible labeling γ of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma(L_t^i) = p^i$ for all $i \in \{\ell, \ldots, t\}$. Restrict γ to obtain a labeling γ' of $\mathcal{I}_{t-1}$ in the obvious manner: $\gamma'(L_{t-1}^i) = \gamma(L_t^i) = p^i$ for all $i \in \{\ell, \ldots, t\}$ and $\gamma'(I) = \gamma(I)$ for all other intervals I of $\mathcal{I}_{t-1}$ (which are also intervals of $\mathcal{I}_t$). It is easy to check that γ' is a feasible labeling of $\mathcal{I}_{t-1}$ with respect to ρ_{t-1}. Thus, from Definition 10 we get, $p^\ell \in Q_{t-1}^\ell(p^{\ell+1}, \ldots, p^k)$. ◀

4.2 Algorithm

Before stating our WkS-RSP algorithm formally, we give some intuitive explanation. Recall that s_t^ℓ denotes the location of the algorithm's ℓ'th server after serving the t'th request. The critical invariant maintained by our algorithm is the following.

▶ **Invariant 14.** *For every t and ℓ, in response to the t'th request, the algorithm keeps its ℓ'th server at a uniformly random point in $Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k)$.*

Lemma 16 states this claim formally. For now, let us just understand the scenarios in which the algorithm needs to move the ℓ'th server at time t so that it occupies *some* point in $Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k)$. These scenarios are as follows.

1. The algorithm moves the ℓ''th server for some $\ell' > \ell$. This means that, potentially, $s_t^{\ell'} \neq s_{t-1}^{\ell'}$, so $Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k)$ could be different from $Q_{t-1}^\ell(s_{t-1}^{\ell+1}, \ldots, s_{t-1}^k)$.
2. $\ell_t \geq \ell$. This means that $L_t^\ell = [t, t+1)$ is disjoint from L_{t-1}^ℓ, and again, potentially, $Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k)$ could be different from $Q_{t-1}^\ell(s_{t-1}^{\ell+1}, \ldots, s_{t-1}^k)$.
3. None of the above happens, so by Lemma 13, $Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k) \subseteq Q_{t-1}^\ell(s_{t-1}^{\ell+1}, \ldots, s_{t-1}^k)$. However, $s_{t-1}^\ell \notin Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k)$ (that is, $s_{t-1}^\ell \in Q_{t-1}^\ell(s_{t-1}^{\ell+1}, \ldots, s_{t-1}^k) \setminus Q_t^\ell(s_t^{\ell+1}, \ldots, s_t^k)$), so the ℓ'th server can no longer remain at the same place s_{t-1}^ℓ as before.

We call the movement in the first two scenarios a *forced movement* (because the movement of some other server forced this movement), and the movement in the third scenario an *unforced movement.* If none of the above scenarios arises, then the ℓ'th server stays put. Algorithm 1 is the formal description of our algorithm for WkS-RSP.

Algorithm 1 WkS-RSP.

```
 1: for $t = 1$ to $T$ do
 2:   Input: request $\sigma_t \in U$, and $\ell_t \in \{0, \dots, k\}$.
 3:   {Recall: $\mathcal{I}_t$ is the $\ell_t$-extension of $\mathcal{I}_{t-1}$.}
 4:   flag ← FALSE
 5:   for $\ell = k$ to 1 do
 6:     {Decide movements of servers in decreasing order of weight.}
 7:     {flag = TRUE indicates that an unforced movement of some server heavier than the
        $\ell$'th has happened.}
 8:     Compute $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$ (by brute force).
 9:     if flag OR $\ell \le \ell_t$ then
10:       $s_t^\ell \leftarrow$ a uniformly random point in $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$. {forced movement}
11:     else if $s_{t-1}^\ell \notin Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$ then
12:       $s_t^\ell \leftarrow$ a uniformly random point in $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$. {unforced movement}
13:       flag ← TRUE.
14:     else
15:       $s_t^\ell \leftarrow s_{t-1}^\ell$. {no movement}
16:     end if
17:   end for
18: end for
```

Note that it is unclear so far why Algorithm 1 is well-defined – why the set $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$ is nonempty when we attempt to send the ℓ'th server to a uniformly random point in it in steps 10 and 12 – and why every request gets served. We provide an answer now.

▶ **Lemma 15.** *For every $t \in \{1, \dots, T\}$ the following statements hold with probability one.*

1. *For every $\ell \in \{0, \dots, k\}$, there exists a feasible labeling γ of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma(L_t^i) = s_t^i$ for all $i \in \{\ell + 1, \dots, k\}$.*
2. *For every $\ell \in \{1, \dots, k\}$, the set $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$ is non-empty.*
3. *Algorithm 1 serves the t'th request.*

Proof. For an arbitrary $t \in \{1, \dots, T\}$, we prove the lemma by reverse induction on $\ell \in \{0, \dots, k\}$ in an interleaved manner. More precisely, as the base case, we prove the first claim for $\ell = k$. Assuming that the first claim holds for an arbitrary $\ell > 0$, we prove that the second claim holds for the same ℓ. Assuming that the second claim holds for an arbitrary $\ell > 0$, we prove that the first claim holds for $\ell - 1$. Finally, assuming that the first claim holds for $\ell = 0$, we prove that the third claim holds.

As the base case, we need to prove the first claim for $\ell = k$. We know that the service pattern $\mathcal{I}_t$ that the adversary provides is feasible. This implies that there exists a feasible labeling γ of $\mathcal{I}_t$ with respect to ρ_t. The condition $\gamma(L_t^i) = s_t^i$ for all $i \in \{\ell + 1, \dots, k\}$ is vacuously true.

For the inductive step, assume that the first claim holds for some $0 < \ell \le k$. Hence, there exists a feasible labeling γ of $\mathcal{I}_t$ such that $\gamma(L_t^i) = s_t^i$, for all $i \in \{\ell + 1, \dots, k\}$. By Definition 10, the point $\gamma(L_t^\ell)$ lies in $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k)$. Thus, $Q_t^\ell(s_t^{\ell+1}, \dots, s_t^k) \neq \emptyset$.

We designed the algorithm so that s_t^ℓ is guaranteed to be in $Q_t^\ell(s_t^{\ell+1},\dots,s_t^k)$. By Definition 10, there exists a feasible labeling γ' of $\mathcal{I}_t$ such that $\gamma'(L_t^i) = s_t^i$ for all $i \in \{\ell,\dots,k\}$. Hence, the first claim holds for $\ell - 1$ as well. This proves the first two claims.

Finally, since the first claim holds for $\ell = 0$, there exists a feasible labeling γ of $\mathcal{I}_t$ with respect to ρ_t such that $\gamma(L_t^i) = s_t^i$ for all $i \in \{1,\dots,k\}$. By definition of feasibility of a labeling (Definition 5), γ must assign the label σ_t to some interval in $\mathcal{I}_t$ that contains t. Since the only intervals in $\mathcal{I}_t$ that contains t are the L_t^i's, we must have $\sigma_t = s_t^i$ for some i. Since s_t^i's are the positions of the algorithm's servers after processing the t'th request, the request gets served. ◀

4.3 Competitive Analysis

We begin by proving that the algorithm indeed maintains Invariant 14 after serving every request.

▶ **Lemma 16.** *For every $t \in \{1,\dots,T\}$, every $\ell \in \{1,\dots,k\}$, and every $p^{\ell+1},\dots,p^k \in U$, conditioned on $s_t^i = p^i$ for all $i \in \{\ell+1,\dots,k\}$ and $Q_t^\ell(p^{\ell+1},\dots,p^k) \neq \emptyset$, s_t^ℓ is a uniformly random point in $Q_t^\ell(p^{\ell+1},\dots,p^k)$.*

Proof. We prove this lemma by induction of time t. The base case of $t = 1$ is true because we are assuming $\ell_1 = k$, which makes the condition in step 9 of the algorithm true.

Consider the inductive case, where $t > 1$. Note that the algorithm executes exactly one step out of 10, 12, and 15. Conditioned on the algorithm executing step 10 or 12, s_t^ℓ is located at a uniformly random point in $Q_t^\ell(p^{\ell+1},\dots,p^k)$ by design. On the other hand, suppose the algorithm executes step 15, that is, the checks in steps 9 and 11 fail. Then the fact that the check of step 9 failed implies that the algorithm did not move any server heavier than the ℓ'th. Thus $(s_{t-1}^{\ell+1},\dots,s_{t-1}^k) = (s_t^{\ell+1},\dots,s_t^k) = (p^{\ell+1},\dots,p^k)$. Therefore, by induction hypothesis, s_{t-1}^ℓ is a uniformly random point in $Q_{t-1}^\ell(p^{\ell+1},\dots,p^k)$. Additionally, conditioned on the failure of the check in step 11, $s_{t-1}^\ell \in Q_t^\ell(s_t^{\ell+1},\dots,s_t^k)$, so $s_t^\ell = s_{t-1}^\ell$ is a uniformly random point in $Q_{t-1}^\ell(p^{\ell+1},\dots,p^k) \cap Q_t^\ell(p^{\ell+1},\dots,p^k)$. But by Lemma 13, $Q_{t-1}^\ell(p^{\ell+1},\dots,p^k) \cap Q_t^\ell(p^{\ell+1},\dots,p^k) = Q_t^\ell(p^{\ell+1},\dots,p^k)$, so s_t^ℓ is a uniformly random point in $Q_t^\ell(p^{\ell+1},\dots,p^k)$. Thus, irrespective of which one of steps 10, 12, and 15 is executed, s_t^ℓ is a uniformly random point in $Q_t^\ell(p^{\ell+1},\dots,p^k)$, and this implies the claim. ◀

Having established that our algorithm is well-defined and that it indeed serves every request, we now focus on bounding the cost of algorithm's solution. For each $\ell \in \{1,\dots,k\}$, define the random variables X^ℓ and Y^ℓ to be the number of forced movements and unforced movements respectively, of the algorithm's ℓ'th server. First, we bound X^ℓ for all ℓ as follows.

▶ **Lemma 17.** *The following inequalities hold with probability one.*

1. $X^\ell \le X^{\ell+1} + Y^{\ell+1} + |\mathcal{I}_T^\ell|$ *for all* $\ell \in \{1,\dots,k-1\}$.
2. $X^k \le |\mathcal{I}_T^k|$.

Proof. For $\ell < k$, every forced movement of the ℓ'th server happens at time t only if `flag` is true or $\ell \le \ell_t$. Observe that in the former case, the $(\ell+1)$'th server must have moved at time t too, so we charge the movement of the ℓ'th server to the movement of the $(\ell+1)$'th server, which could be either forced or unforced. In the latter case, since $\mathcal{I}_T$ is a hierarchical service pattern, a new interval starts at time t in the ℓ'th level $\mathcal{I}_T^\ell$ of $\mathcal{I}_T$, so we charge the movement of the ℓ'th server to that interval. The argument for the second claim is the same as above except that `flag` is never true; a forced movement of the k'th server happens at time t only if $\ell_t = k$. ◀

Next, we bound Y^ℓ by first bounding the number of unforced movements of the ℓ'th server in an arbitrary interval in which no forced movement of the ℓ'th server happens.

▶ **Lemma 18.** *For every $\ell \in \{1, \ldots, k\}$, every $p^{\ell+1}, \ldots, p^k \in U$, and every t_{begin}, t_{end} such that $1 \leq t_{begin} < t_{end} \leq T+1$, the following holds. Conditioned on the event that $s_t^j = p^j$ for all $j \in \{\ell+1, \ldots, k\}$, and all $t \in (t_{begin}, t_{end})$, the expected number of unforced movements of the algorithm's ℓ'th server is at most $H(n_\ell)$, where H denotes the harmonic function defined as $H(n) = 1 + 1/2 + \cdots + 1/n$.*

Proof. Conditioned on the event that $s_t^j = p^j$ for all $j \in \{\ell+1, \ldots, k\}$ and all $t \in (t_{\text{begin}}, t_{\text{end}})$, we use Lemma 13 to claim that $Q_{t-1}^\ell(p^{\ell+1}, \ldots, p^k) \supseteq Q_t^\ell(p^{\ell+1}, \ldots, p^k)$ for every $t \in (t_{\text{begin}}, t_{\text{end}})$. For brevity we write Q_t for $Q_t^\ell(p^{\ell+1}, \ldots, p^k)$. Let Z_t be the indicator random variable of the event that an unforced movement happens at time t. From the algorithm, we know that this event happens if and only if $s_{t-1}^\ell \in Q_{t-1} \setminus Q_t$. From Lemma 16, we know that s_{t-1}^ℓ is a uniformly random point in Q_{t-1}. Thus,

$$\mathbb{E}[Z_t] = \frac{|Q_{t-1}| - |Q_t|}{|Q_{t-1}|} \leq \frac{1}{|Q_{t-1}|} + \frac{1}{|Q_{t-1}|-1} + \cdots + \frac{1}{|Q_t|+1} = H(|Q_{t-1}|) - H(|Q_t|)$$

Let t_1 denote the earliest time t for which $Q_t \subsetneq Q_{t-1}$. Then the expected number of unforced movements is bounded as

$$\sum_{t=t_1}^{t_{\text{end}}-1} \mathbb{E}[Z_t] \leq 1 + \sum_{t=t_1+1}^{t_{\text{end}}-1} H(|Q_{t-1}|) - H(|Q_t|) = 1 + H(|Q_{t_1}|) - H(|Q_{t_{\text{end}}-1}|).$$

Since $Q_{t_1} \subsetneq Q_{t_1-1} \subseteq U$, by Lemma 12, we have $|Q_{t_1}| \leq n_\ell$. By the second claim of Lemma 15, $|Q_{t_{\text{end}}-1}| \geq 1$. Thus, the expected number of unforced movements is at most $H(n_\ell)$. ◀

▶ **Lemma 19.** *For every $\ell \in \{1, \ldots, k\}$, we have $\mathbb{E}[Y^\ell] \leq H(n_\ell) \cdot \mathbb{E}[X^\ell]$.*

Proof. Let us condition on the sequence of timestamps at which a forced movement of the ℓ'th server takes place. If t_1, t_2 are two consecutive timestamps in this sequence, it is easy to notice that the servers $\ell+1, \ldots, k$ remain at the same position throughout this interval. Then Lemma 18 applied to the interval (t_1, t_2) implies that the expected number of unforced movements of the ℓ'th server in this interval is at most $H(n_\ell)$. Summing up over all pairs t_1, t_2 of consecutive timestamps, we get $\mathbb{E}[Y^\ell | X^\ell = x] \leq H(n_\ell) \cdot x$, and therefore, $\mathbb{E}[Y^\ell] \leq H(n_\ell) \cdot \mathbb{E}[X^\ell]$. ◀

▶ **Theorem 2.** *There is a randomized algorithm for WkS-RSP with a competitive ratio of $2^{O(k^2)}$.*

Proof. From Lemma 15, we already know that Algorithm 1 serves every request in ρ_T. Towards proving competitiveness of the algorithm, we first define the constants $c_k, \ldots, c_1$ inductively as follows: $c_k = H(n_k) + 1$ and $c_\ell = (H(n_\ell) + 1) \cdot (c_{\ell+1} + 1)$, for every $\ell \in \{1, \ldots, k-1\}$. We claim that for every $\ell \in \{1, \ldots, k\}$, the expected number of movements of the algorithm's ℓ'th server, which equals $\mathbb{E}[X^\ell] + \mathbb{E}[Y^\ell]$, is at most c_ℓ times the number of movements of the adversary's ℓ'th server, which equals $|\mathcal{I}_T^\ell|$. We prove this claim using reverse induction on ℓ from k to 1.

For the base case, i.e. $\ell = k$, from Lemma 19 we have that $\mathbb{E}[Y^k] \leq H(n_k) \cdot \mathbb{E}[X^k]$. From Lemma 17, we know that $\mathbb{E}[X^k] \leq |\mathcal{I}_T^k|$. Thus, the expected number of algorithm's k'th server movements is,

$$\mathbb{E}[X^k] + \mathbb{E}[Y^k] \leq (H(n_k) + 1) \cdot \mathbb{E}[X^k] = c_k \cdot |\mathcal{I}_T^k|.$$

For the inductive case, assume that $\mathbb{E}[X^{\ell+1}] + \mathbb{E}[Y^{\ell+1}] \leq c_{\ell+1} \cdot |\mathcal{I}_T^{\ell+1}|$, for an arbitrary $\ell \in \{1, \dots, k-1\}$. From Lemma 19, we have that $\mathbb{E}[Y^\ell] \leq H(n_\ell) \cdot \mathbb{E}[X^\ell]$, and from Lemma 17, we have that $\mathbb{E}[X^\ell] \leq \mathbb{E}[X^{\ell+1}] + \mathbb{E}[Y^{\ell+1}] + |\mathcal{I}_T^\ell|$. Thus, we have,

$$\mathbb{E}[X^\ell] + \mathbb{E}[Y^\ell] \leq (H(n_\ell) + 1) \cdot \mathbb{E}[X^\ell] \leq (H(n_\ell) + 1) \cdot \left(\mathbb{E}[X^{\ell+1}] + \mathbb{E}[Y^{\ell+1}] + |\mathcal{I}_T^\ell|\right).$$

Recall that $\mathcal{I}_T = (\mathcal{I}_T^1, \dots, \mathcal{I}_T^k)$ is a hierarchical service pattern, and therefore, $|\mathcal{I}_T^{\ell+1}| \leq |\mathcal{I}_T^\ell|$. Using this fact, the induction hypothesis, and the definition of c_ℓ, we get,

$$\mathbb{E}[X^\ell] + \mathbb{E}[Y^\ell] \leq (H(n_\ell)+1) \cdot \left(c_{\ell+1} \cdot |\mathcal{I}_T^{\ell+1}| + |\mathcal{I}_T^\ell|\right) \leq (H(n_\ell)+1) \cdot (c_{\ell+1}+1) \cdot |\mathcal{I}_T^\ell| = c_\ell \cdot |\mathcal{I}_T^\ell|,$$

as required.

As a consequence of the above inductive claim, the total cost of the algorithm is at most $\max\{c_1, \dots, c_k\} = c_1$ times the cost of the adversary's service pattern. Moreover, from the recurrence relation defining $c_k, \dots, c_1$ and the upper bound on n_ℓ from Fact 11, it is clear that c_1 is $2^{O(k^2)}$. Thus, the competitive ratio of our algorithm is $2^{O(k^2)}$. ◀

5 Concluding Remarks and Open Problems

The main open question of finding the randomized competitive ratio of weighted k-server on uniform metrics still remains unresolved. Our decomposition approach and the randomized algorithm for WkS-RSP imply that the task of designing a $2^{\text{poly}(k)}$-competitive randomized algorithm for weighted k-server on uniform metrics is equivalent to designing a $2^{\text{poly}(k)}$-competitive algorithm for WkS-SPC. We do not know any non-trivial bounds on the competitive ratio of WkS-SPC and it is not even clear whether it is easier or harder than WkS-RSP in terms of competitiveness. While the known lower bound constructions for weighted k-server also apply to WkS-RSP, these constructions fail to get a lower bound on the competitive ratio of WkS-SPC. We therefore propose the open problem of finding bounds on the competitive ratio of WkS-SPC in the deterministic as well as randomized setting.

In the deterministic setting, the competitive ratio of WkS-SPC is bounded from below by the competitive ratio of weighted k-server divided by the competitive ratio of WkS-RSP, again due to Theorem 1. However, for this to give a non-trivial lower bound on the competitive ratio of WkS-SPC, we require an upper bound on the competitive ratio of WkS-RSP that is less than the known lower bound on the competitive ratio of weighted k-server. Unfortunately, no such upper bound is known. Thus, showing a separation between weighted k-server and WkS-RSP is an interesting open problem.

Additionally, we also believe that closing the quadratic gap between the exponents in the upper and lower bounds on the randomized competitive ratio of WkS-RSP is an interesting open problem, because it will result in a better understanding of the weighted k-server problem.

Finally, for the weighted k-server problem with $k > 2$, no weight-independent and metric-independent upper bounds on the competitive ratio are known on any well-structured class of metrics larger than the class of uniform metrics. Proving such bounds seems rather ambitious, given our limited understanding of weighted k-server on uniform metrics.

References

1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.

2 Nikhil Ayyadevara and Ashish Chiplunkar. The randomized competitive ratio of weighted k-server is at least exponential. In *ESA*, volume 204 of *LIPIcs*, pages 9:1–9:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.9.

3 Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *FOCS*, pages 493–504, 2017. doi:10.1109/FOCS.2017.52.

4 Nikhil Bansal, Marek Eliás, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *SODA*, pages 992–1001, 2018. doi:10.1137/1.9781611975031.64.

5 Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. doi:10.1007/BF01294260.

6 Sébastien Bubeck, Christian Coester, and Yuval Rabani. The randomized k-server conjecture is false! In *STOC*, pages 581–594. ACM, 2023. doi:10.1145/3564246.3585132.

7 Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized k-server problems. *ACM Trans. Algorithms*, 16(1):14:1–14:28, 2020. doi:10.1145/3365002.

8 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.

9 Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theoretical Computer Science*, 130(1):85–99, 1994. doi:10.1016/0304-3975(94)90154-6.

10 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Efficient algorithms and hardness results for the weighted k-server problem. In *APPROX/RANDOM*, volume 275 of *LIPIcs*, pages 12:1–12:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.12.

11 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

12 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *STOC*, pages 322–333, 1988. doi:10.1145/62212.62243.

13 Lee A. Newberg. The k-server problem with distinguishable servers. Master's thesis, University of California, Berkeley, 1991.

14 René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1):96–125, 2014. doi:10.1137/120885309.

15 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

16 Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002. doi:10.1007/s00453-001-0124-5.

A Lower Bound for WkS-RSP

In this section, we show that the lower-bound construction for weighted k-server by Ayyadevara and Chiplunkar [2] applies to WkS-RSP and gives the same lower bound. In [2] the generation of an adversarial request sequence involves repeatedly calling a randomized recursive procedure named strategy. Their analysis bounds the adversary's cost and the expected cost of an arbitrary deterministic algorithm, both amortized per strategy call. Then the exponential lower bound is established by an application of Yao's principle. We show that essentially the same adversarial construction of input distribution works for WkS-RSP. The only change needed in the construction is that now the adversary must provide its service pattern along with the requested point in an online manner.

The weights of the servers are $1, \beta, \ldots, \beta^{k-1}$ where β is a large integer. The sequence $n_0, n_1, \ldots$ is defined as $n_0 = 1$ and for $\ell > 0$,

$$n_\ell = \left(\left\lceil \frac{n_{\ell-1}}{2} \right\rceil + 1\right) \cdot \left(\left\lfloor \frac{n_{\ell-1}}{2} \right\rfloor + 1\right).$$

The adversarial strategy in [2] uses the following combinatorial result from [3].

▶ **Fact 20** ([3]). *Let $\ell \in \mathbb{N}$ and let P be a set of n_ℓ points. There exists a set-system $\mathcal{Q}_\ell \subseteq 2^P$ satisfying the following properties.*

1. *$\mathcal{Q}_\ell$ contains $\lceil n_{\ell-1}/2 \rceil + 1$ sets, each of size $n_{\ell-1}$.*
2. *For every $p \in P$, there exists a set in $\mathcal{Q}_\ell$ not containing p.*
3. *For every $p \in P$, there exists a $q \in P$ such that every set in $\mathcal{Q}_\ell$ contains at least one of p and q.*

We modify the adversarial strategy from [2] for weighted k-server to get the following strategy for WkS-RSP.

Procedure 2 adversary.

```
Mark all points in $S$;
repeat infinitely many times
    Pick a point $p$ uniformly at random from $S$ (with replacement);
    Mark $p$;
    if All points in $S$ are marked then
        $\ell_{ext} \leftarrow k$;
        Unmark all points $q \in S$ other than $p$;
    else
        $\ell_{ext} \leftarrow k - 1$;
    Call strategy($k - 1, S \setminus \{p\}, \ell_{ext}$);
```

Procedure 3 strategy(ℓ, P, ℓ_{ext}) (Promise: $|P| = n_\ell$ and $\ell_{ext} \geq \ell$).

```
if $\ell = 0$ (and therefore, $|P| = n_0 = 1$) then
    Output $(p, \ell_{ext})$, where $p$ is the unique point in $P$;
else
    Construct the set-system $\mathcal{Q}_\ell \subseteq 2^P$ using Fact 20;
    repeat $(\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1)$ times
        Pick a set $P'$ uniformly at random from $\mathcal{Q}_\ell$ (with replacement);
        Call strategy($\ell - 1, P', \ell_{ext}$);
        $\ell_{ext} \leftarrow \ell - 1$;
```

The following claim bounds the expected cost of an arbitrary online algorithm for every strategy call made by adversary. Its proof is identical to the proof of Corollary 6 in [2]. It is noteworthy that all the arguments involved in that proof go through even in the revealed service pattern setting.

▶ **Lemma 21.** *For $\ell_{ext} = k$ or $k-1$, the expected cost of the algorithm per strategy$(k-1, P, \ell_{ext})$ call made by adversary is $(\beta - 1)^{k-1}/(n_{k-1} + 1)$.*

We now show that the service pattern $\mathcal{I} = (\mathcal{I}^1, \ldots, \mathcal{I}^k)$ created by the procedure adversary is feasible, that is, it can be labeled in a way that all requests get served. We start by noting the following.

▶ **Observation 22.** *Let $(p_{t_1}, \ell_{t_1}), \ldots, (p_{t_2}, \ell_{t_2})$ be the input generated by a* strategy(ℓ, P, ℓ_{ext}) *call which starts at time t_1 and ends at time t_2. Then $\ell_{t_1} = \ell_{ext} \geq \ell$ and $\ell_{t_1+1}, \ldots, \ell_{t_2}$ are all less than ℓ. As a consequence, the following statements about the adversary's service pattern $\mathcal{I} = (\mathcal{I}^1, \ldots, \mathcal{I}^k)$ hold.*

1. *For all $\ell' \geq \ell$, a single interval in $\mathcal{I}^{\ell'}$ covers the interval $[t_1, t_2 + 1)$.*
2. *For all $\ell' \leq \ell$, the interval in $\mathcal{I}^{\ell'}$ covering t_1 starts at t_1, and the interval in $\mathcal{I}^{\ell'}$ covering t_2 ends at $t_2 + 1$.*

In particular, $[t_1, t_2 + 1)$ is an interval in $\mathcal{I}^\ell$.

▶ **Lemma 23.** *Consider an arbitrary* strategy(ℓ, P, ℓ_{ext}) *call which starts at time t_1, ends at time t_2, and generates the input $(p_{t_1}, \ell_{t_1}), \ldots, (p_{t_2}, \ell_{t_2})$. Suppose for all $\ell' > \ell$, the single interval in $\mathcal{I}^{\ell'}$ covering $[t_1, t_2+1)$ is labeled with some point $p_{\ell'}$, such that $P \cap \{p_{\ell+1}, \ldots, p_k\} \neq \emptyset$. Then the intervals in $\mathcal{I}^1, \ldots, \mathcal{I}^\ell$ that intersect $[t_1, t_2 + 1)$ (and therefore, are subsets of $[t_1, t_2+1)$) can be labeled in such a way that for all $t \in \{t_1, \ldots, t_2\}$ there exists $i \in \{1, \ldots, k\}$ such that the unique interval in $\mathcal{I}^i$ covering t is labeled with p_t.*

Proof. We prove by induction on ℓ. For $\ell = 0$, the set P contains a single point, which gets requested. Since $P \cap \{p_1, \ldots, p_k\} \neq \emptyset$, the claim holds.

For $\ell > 0$, consider a point $p \in P \cap \{p_{\ell+1}, \ldots, p_k\}$. From the third property in Fact 20, there exists a point $q \in P$ such that every set in the set system $\mathcal{Q}_\ell$ contains at least one of the points p or q. Label the interval $[t_1, t_2 + 1)$ in $\mathcal{I}^\ell$ by such a point q. Consider an arbitrary recursive call strategy$(\ell - 1, P', \ell_{ext})$ which starts at time $t_1' \geq t_1$ and ends at time $t_2' \leq t_2$. We have $P' \cap \{q, p_{\ell+1}, \ldots, p_k\} \neq \emptyset$. By induction hypothesis, the intervals in $\mathcal{I}^1, \ldots, \mathcal{I}^{\ell-1}$ that intersect $[t_1', t_2' + 1)$ can be labeled in such a way that for all $t \in \{t_1', \ldots, t_2'\}$ there exists $i \in \{1, \ldots, k\}$ such that the unique interval in $\mathcal{I}^i$ covering t is labeled with p_t. ◀

▶ **Lemma 24.** *The service pattern $\mathcal{I} = (\mathcal{I}^1, \ldots, \mathcal{I}^k)$ created by the procedure* adversary *is feasible.*

Proof. Every interval in $\mathcal{I}^k$ starts exactly when all points in S are found to be marked. For every interval I^k in $\mathcal{I}^k$ do the following. Label it by the point q whose marking results in the beginning of the next interval in $\mathcal{I}^k$. In other words, q is the last point to get marked after the unmarking step in the beginning of I^k. Thus, q is never sampled by adversary during the interval I^k, and therefore q belongs to the set $S \setminus \{p\}$ passed to every strategy call made by adversary during the interval I^k. Thus, by Lemma 23 for $\ell = k - 1$, the intervals in $\mathcal{I}^1, \ldots, \mathcal{I}^{k-1}$ that are subsets of I^k can be labeled in such a way that all requests given during the interval I^k are served. Thus, the service pattern $\mathcal{I}$ is feasible. ◀

Now, we bound the cost of the service pattern $\mathcal{I} = (\mathcal{I}^1, \ldots, \mathcal{I}^k)$ created by the procedure adversary. We start by bounding the total cost of intervals created during a strategy(ℓ, P, ℓ_{ext}) call.

▶ **Lemma 25.** *Define the sequence $c_0, c_1, \ldots$ inductively as follows: $c_0 = 0$ and for $\ell > 0$*

$$c_\ell = \beta^{\ell-1} + \beta \cdot (\lceil n_{\ell-1}/2 \rceil + 1) \cdot c_{\ell-1}$$

For an arbitrary $\ell \in \{0, \ldots, k-1\}$ and $\ell_{ext} \geq \ell$, consider a call of strategy(ℓ, P, ℓ_{ext}) *which starts at time t_1 and ends at time t_2. The total cost of the intervals in layers $\mathcal{I}_1, \ldots, \mathcal{I}_\ell$ that intersect the interval $[t_1, t_2 + 1)$ (equivalently, are subsets of $[t_1, t_2 + 1)$, by Observation 22) is at most c_ℓ.*

Proof. We prove this by induction on ℓ. The claim is trivially true for $\ell = 0$. For $\ell > 0$, the $\mathsf{strategy}(\ell, P, \ell_{ext})$ makes $(\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1)$ recursive calls of $\mathsf{strategy}(\ell - 1, P', \ell_{ext})$. For each of these calls, the following holds by induction hypothesis. If the call starts at time $t'_1 \geq t_1$ and ends at time $t'_2 \leq t_2$, then the total cost of intervals in layers $\mathcal{I}_1, \ldots, \mathcal{I}_{\ell-1}$ that intersect the interval $[t'_1, t'_2 + 1)$ is at most $c_{\ell-1}$. Since there are $(\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1)$ such recursive calls the total cost of intervals in layers $\mathcal{I}_1, \ldots, \mathcal{I}_{\ell-1}$ that intersect the interval $[t_1, t_2 + 1)$ is at most $(\beta - 1) \cdot (\lceil n_{\ell-1}/2 \rceil + 1) \cdot c_{\ell-1}$. Adding to this the cost $\beta^{\ell-1}$ of the interval $[t_1, t_2 + 1) \in \mathcal{I}_\ell$ gives the required bound. ◀

▶ **Theorem 3.** *The randomized competitive ratio of WkS-RSP is* $\Omega(2^k)$.

Proof. Consider the service pattern $\mathcal{I} = (\mathcal{I}^1, \ldots, \mathcal{I}^k)$ of the adversary. Every interval in $\mathcal{I}^k$ starts with one marked point and ends just before all the points in the set S are marked in the procedure $\mathsf{adversary}$. Using the standard coupon collector argument, we get that the expected number of $\mathsf{strategy}(k - 1, S \setminus \{p\}, \ell_{ext})$ made during an interval in $\mathcal{I}^k$ is $(n_{k-1}+1)H(n_{k-1})$. Thus, the amortized cost of intervals in $\mathcal{I}^k$ per $\mathsf{strategy}(k-1, S \setminus \{p\}, \ell_{ext})$ call is $\beta^{k-1}/((n_{k-1} + 1)H(n_{k-1}))$. From Lemma 25, we get that the total cost of intervals in $\mathcal{I}^1, \ldots, \mathcal{I}^{k-1}$ per $\mathsf{strategy}(k - 1, S \setminus \{p\}, \ell_{ext})$ call is at most c_{k-1}. The cost of the revealed service pattern per $\mathsf{strategy}(k-1, S \setminus \{p\}, \ell_{ext})$ call is at most $\beta^{k-1}/((n_{k-1}+1)H(n_{k-1}))+c_{k-1}$. The rest of the proof is identical to the proof of Theorem 2 in [2]. Essentially, for a large β, the dominant term in the adversary's cost is $\beta^{k-1}/((n_{k-1}+1)H(n_{k-1}))$, while the algorithm's cost is $\beta^{k-1}/(n_{k-1} + 1)$ modulo a lower order term due to Lemma 21, thus implying a lower bound of $H(n_{k-1}) = \Omega(2^k)$ on the competitive ratio. ◀

Minimum Consistent Subset in Trees and Interval Graphs

Aritra Banik
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, India

Sayani Das
Theoretical Computer Science,
The Institute of Mathematical Sciences,
Chennai, India

Anil Maheshwari
School of Computer Science, Carleton University, Ottawa, Canada

Bubai Manna
Department of Mathematics, Indian Institute of Technology Kharagpur, India

Subhas C. Nandy
Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India

Krishna Priya K. M.
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, India

Bodhayan Roy
Department of Mathematics, Indian Institute of Technology Kharagpur, India

Sasanka Roy
Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India

Abhishek Sahu
National Institute of Science, Education and Research, An OCC of Homi Bhabha National Institute, Bhubaneswar, India

Abstract

In the Minimum Consistent Subset (MCS) problem, we are presented with a connected simple undirected graph G, consisting of a vertex set $V(G)$ of size n and an edge set $E(G)$. Each vertex in $V(G)$ is assigned a color from the set $\{1, 2, \ldots, c\}$. The objective is to determine a subset $V' \subseteq V(G)$ with minimum possible cardinality, such that for every vertex $v \in V(G)$, at least one of its nearest neighbors in V' (measured in terms of the hop distance) shares the same color as v. The decision problem, indicating whether there exists a subset V' of cardinality at most l for some positive integer l, is known to be NP-complete even for planar graphs.

In this paper, we establish that the MCS problem is NP-complete on trees. We also provide a fixed-parameter tractable (FPT) algorithm for MCS on trees parameterized by the number of colors (c) running in $O(2^{6c}n^6)$ time, significantly improving the currently best-known algorithm whose running time is $O(2^{4c}n^{2c+3})$. In an effort to comprehensively understand the computational complexity of the MCS problem across different graph classes, we extend our investigation to interval graphs. We show that it remains NP-complete for interval graphs, thus enriching graph classes where MCS remains intractable.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Graph algorithms analysis; Theory of computation → Dynamic graph algorithms

Keywords and phrases Nearest-Neighbor Classification, Minimum Consistent Subset, Trees, Interval Graphs, Parameterized complexity, NP-complete

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.7

Related Version *Full Version*: `https://arxiv.org/abs/2404.15487`

Funding *Aritra Banik*: Supported by the Science and Engineering Research Board (SERB) via the project MTR/2022/000253.

Anil Maheshwari: Supported by Natural Sciences and Engineering Research Council of Canada (NSERC)

Bodhayan Roy: Supported by the Science and Engineering Research Board (SERB) via the project MTR/2021/000474.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 7; pp. 7:1–7:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

For many supervised learning algorithms, the input comprises a colored training dataset T in a metric space $(\mathcal{X}, d)$ where each element $t \in T$ is assigned a color $C(t)$ from the set of colors $[c]$. The objective is to preprocess T in a manner that enables rapid assignment of color to any uncolored element in $\mathcal{X}$, satisfying specific optimality criteria. One commonly used optimality criterion is the nearest neighbor rule, where each uncolored element x is assigned a color based on the colors of its k nearest neighbors in the training dataset T (where k is a fixed integer). The efficiency of such an algorithm relies on the size of the training dataset. Therefore, it is crucial to reduce the size of the training set while preserving distance information. This concept was formalized by Hart [11] in 1968 as the minimum consistent subset (MCS) problem. In this problem, given a colored training dataset T, the objective is to find a subset $S \subseteq T$ of minimum cardinality such that for every point $t \in T$, the color of t is the same as the color of one of its nearest neighbors in S. Since its inception, the MCS problem has found numerous applications, as can be judged by over 2800 citations to [11] in Google Scholar.

The MCS problem for points in $\Re^2$ under the Euclidean norm is shown to be NP-complete if at least three colors color the input points. Furthermore, it remains NP-complete even for two colors [15, 12]. Recently, it has been shown that the MCS problem is W[1]-hard when parameterized by the output size [5]. For some algorithmic results on the MCS problem in $\Re^2$, see [3, 15].

In this paper, we explore the minimum consistent subset problem when $(\mathcal{X}, d)$ is a graph metric. Without loss of generality, we will use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For any graph G, we denote the set of vertices of G by $V(G)$ and the set of edges by $E(G)$. Consider any graph G and an arbitrary vertex coloring function $C : V(G) \to [c]$. For a subset of vertices U, let $C(U)$ represent the set of colors of the vertices in U, formally denoted as $C(U) = \{C(u) : u \in U\}$. For any two vertices $u, v \in V(G)$, the shortest path distance between u and v in G is denoted by $d(u, v)$. For a vertex $v \in V(G)$ and any subset of vertices $U \subseteq V(G)$, let $d(v, U) = \min_{u \in U} d(v, u)$. The nearest neighbors of v in the set U are denoted as $\mathrm{NN}(v, U)$, formally defined as $\mathrm{NN}(v, U) = \{u \in U : d(v, u) = d(v, U)\}$. A subset of vertices $S \subseteq V(G)$, is called a consistent subset for (G, C) if for every $v \in V(G)$, $C(v) \in C(\mathrm{NN}(v, S))$. The consistent subset problem on graphs is defined as follows:

Consistent Subset Problem on Graphs(CSPG)

Input: A graph G, a coloring function $C : V(G) \to [c]$, and an integer ℓ.
Question: Does there exist a consistent subset of size $\leq \ell$ for (G, C)?

A consistent subset of minimum size is called a minimum consistent subset (MCS). Banerjee et al. [2] proved that the CSPG is W[2]-hard [6] when parameterized by the size of the minimum consistent set, even when limited to two colors, thus refuting the possibility of an FPT algorithm parameterized by $(c + \ell)$ under standard complexity-theoretic assumptions for general graphs. This naturally raises the question of determining the simplest graph classes where the problem remains computationally intractable. Dey et al. [8] presented a polynomial-time algorithm for CSPG on simple graph classes such as paths, spiders, combs, and caterpillars. The CSPG has gained significant research attention in recent years when the underlying graph is a tree. Dey et al. [7] presented a polynomial-time algorithm for bi-colored trees, and Arimura et al. [1] presented an XP algorithm parameterized by the number of colors c, with a running time of $\mathcal{O}(2^{4c} n^{2c+3})$. Very recently, the paper [4] demonstrated a polynomial time algorithm for the *minimum consistent spanning subset* (a variant of MCS) in trees.

New Results. The most intriguing question yet to be answered is whether CSPG remains NP-hard for trees [1, 7]. In this paper, we decisively answer this question in the affirmative. This is particularly noteworthy given the scarcity of naturally occurring problems known to be NP-hard on trees. Our contribution includes a reduction from the MAX-2SAT problem, detailed in Section 3. Next, we show that CSPG is fixed-parameter tractable (FPT) for trees on n vertices, significantly improving the results presented in Arimura et al. [1]. Our intricate dynamic programming algorithm runs in $\mathcal{O}(2^{6c}n^6)$ time, whereas [1] requires $\mathcal{O}(2^{4c}n^{2c+3})$ time; see Section 4.

Moreover, in Section 5, we show that CSPG on interval graphs is NP-hard. While interval graphs are unrelated to trees, our hardness result for interval graphs raises new questions about the fixed-parameter tractability of CSPG on this distinct graph class.

2 Notation and Preliminary Results

Notations. For any graph G and any vertex $v \in V(G)$, let $N(v) = \{u : u \in V(G),\ (u,v) \in E(G)\}$ denotes the set of neighbours of v and $N[v] = \{v\} \cup N(v)$. We denote the distance between two subgraphs G_1 and G_2 in G by $d(G_1, G_2) = \min\{d(v_1, v_2) : v_1 \in V(G_1), v_2 \in V(G_2)\}$. For any subset of vertices $U \subseteq V(G)$ in a graph G, $G[U]$ denotes the subgraph of G induced on U. Most of the symbols and notations of graph theory used are standard and taken from [9].

As an elementary result, we prove that MCS is log-APX-hard [14]. We reduce the dominating set problem to the consistent subset problem (CSPG). In the *dominating set problem* given a graph G and an integer k the objective is to find out whether there exists a subset $U \subseteq V(G)$ of size k such that for any vertex $v \in V(G)$, $N[v] \cap U \neq \emptyset$. It is known that the set cover problem is log-APX-hard, or in other words, it is NP-hard to approximate the set cover problem within a $c \cdot \log n$ factor [13]. As there exists an L-reduction from set cover to dominating set problem, the dominating set problem is known to be log-APX-hard. Let (G, k) be any arbitrary instance of the dominating set problem with a graph G and an integer k. We construct an instance $(H, C, k+1)$ of CSPG as follows. $V(H) = V(G) \cup \{x\}$ and $E(H) = E(G) \cup \{(x, v) : v \in V(G)\}$. For all $v \in V(G)$, we set $C(v) = 1$, and set $C(x) = 2$. For the sake of completeness, we state the following lemma.

▶ **Lemma 1.** *There exists a dominating set for G of size at most k if and only if there is a consistent subset of size at most $k+1$ for the graph H.*

Proof. Let D be a dominating set of size k for the graph G. We claim that $D' = \{x\} \cup D$ is a consistent subset of H. If not, then there is a vertex $v_i \in V(H) \setminus D'$ such that $d(v_i, D) > d(v_i, x) = 1$. This contradicts the assumption that D is a dominating set for G and the claim holds.

On the other hand, suppose D' is a consistent subset of size $k+1$ in the graph H. Observe that $x \in D'$ as x is the only vertex with color 2. We claim that $D = D' \setminus \{x\}$ is a dominating set of G. If not, then there is a vertex $v \in V(G) \setminus D$ such that $N(v) \cap D = \emptyset$. Thus $d(v, D') > 1$ but $d(v, x) = 1$ and $C(v) \neq C(x)$. This contradicts the assumption that C is a consistent subset and hence the claim holds. ◀

From Lemma 1, we have the following theorem.

▶ **Theorem 2.** *There exists a constant $c > 0$ such that it is NP-hard to approximate the Minimum Consistent Set problem within a factor of $c \cdot \log n$.*

3 NP-hardness of MCS for Trees

In this section, we prove that CSPG is NP-hard even when the input graph is a tree. We present a reduction from MAX-2SAT problem to CSPG. Let θ be a given MAX-2SAT formula with n variables $\{x_1, \ldots, x_n\}$ and m clauses $\{c_1, \ldots, c_m\}$, $n, m \geq 50$. We construct an instance (T_θ, C_θ) of the MCS problem from θ as follows.

Construction of (T_θ, C_θ).

The constructed tree T_θ is composed of variable gadgets, clause gadgets, and central vertex gadgets.

Variable Gadget.

A variable gadget X_i for the variable $x_i \in \theta$ has two components where each component has a literal path and M pairs of stabilizer vertices, as described below (see Figure 1), where M is very large (we will define the exact value of M later).

Literal paths: The two literal paths of the variable gadget X_i are $P_i^\ell = \langle x_i^1, x_i^2, x_i^3, x_i^4 \rangle$ and $\overline{P}_i^\ell = \langle \overline{x}_i^1, \overline{x}_i^2, \overline{x}_i^3, \overline{x}_i^4 \rangle$, each consisting of four vertices; these are referred to as *positive literal path*, and *negative literal path* respectively. Here, by a path of k (> 2) vertices, we mean a connected graph with $k - 2$ nodes having degree 2 and the remaining two nodes having degree 1. All the vertices on the path P_i^ℓ are of color c_i^ℓ and all the vertices on the path $\overline{P}_i^\ell$ are of color $\overline{c}_i^\ell$.

Stabilizer vertices: M pairs of vertices $\{s_i^1, \overline{s}_i^1\}, \ldots, \{s_i^M, \overline{s}_i^M\}$, where the color of each pair of vertices $\{s_i^j, \overline{s}_i^j\}$ is $c^s(i, j)$. We denote the set of vertices $S_i = \{s_i^1, \ldots, s_i^M\}$ as *positive stabilizer vertices* and the set of vertices $\overline{S}_i = \{\overline{s}_i^1, \ldots, \overline{s}_i^M\}$ as *negative stabilizer vertices*. Each vertex in S_i is connected to x_i^1 and each vertex in $\overline{S}_i$ is connected to $\overline{x}_i^1$.

The intuition behind this set of stabilizer vertices is that by setting a large value of M we ensure that either $\{s_i^1, \ldots, s_i^M\}$ or $\{\overline{s}_i^1, \ldots, \overline{s}_i^M\}$ is present in any "small" sized solution.

Clause Gadget.

For each clause $c_i = (y_i \vee z_i)$, where y_i and z_i are two (positive/negative) literals, we define the clause gadget C_i as follows. It consists of three paths, namely *left occurrence path* $P_i^Y = \langle y_i^1, \cdots y_i^7 \rangle$, *right occurrence path* $P_i^Z = \langle z_i^1, \cdots z_i^7 \rangle$, and *clause path* $P_i^W = \langle w_i^1, \cdots w_i^7 \rangle$ (see Figure 1). All the vertices in P_i^Y (resp. P_i^Z) have the same color as that of the corresponding literal path in their respective variable gadgets, i.e. for any literal, say y_i in C_i, if $y_i = x_i$ (resp. $\overline{x_i}$) then we set the color of the vertices of P_i^Y as c_i^x (resp. $\overline{c}_i^x$). The color of the vertices on the path P_i^Z is set in the same manner. We color the vertices in P_i^W by c_i^w, which is different from that of the vertices in P_i^Y and P_i^Z. We create the clause gadget C_i by connecting y_i^1 with w_i^2 and z_i^1 with w_i^6 by an edge (see Figure 1).

Central Vertex Gadget.

We also have a central path $P^v = \langle v_1, v_2, v_3 \rangle$. The color of all the vertices in P^v is the same, say c^v, which is different from the color of all other vertices in the construction. For each variable gadget X_i, x_i^1 and $\overline{x}_i^1$ are connected with the vertex v_1 (see Figure 1). For each clause C_i, w_i^4 is connected with v_1. The color of the vertices of P^v is c^v.

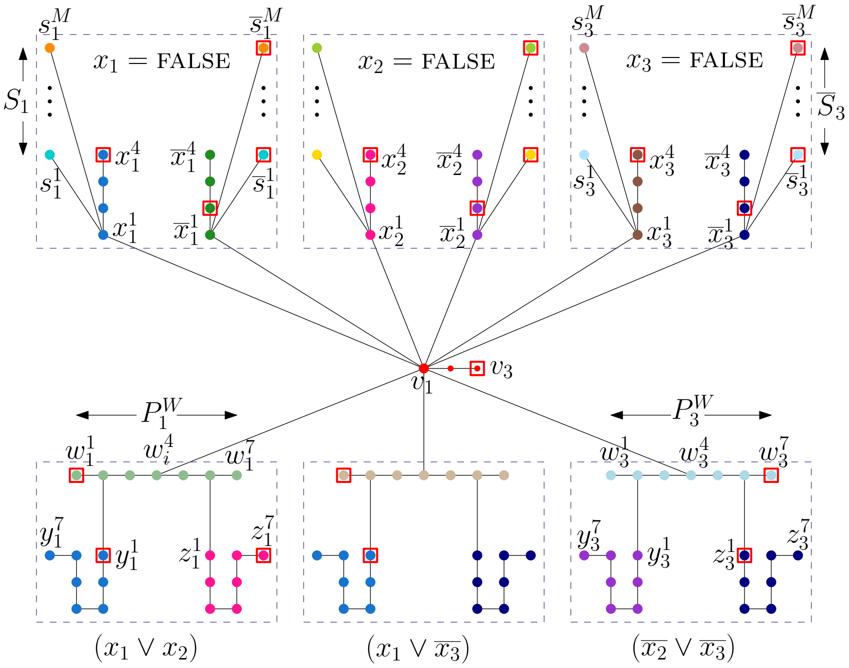


Figure 1 An example of construction of (T_θ, C_θ) where $\theta = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3})$. For the assignment $x_1 = x_2 = x_3 = \text{FALSE}$, we have shown the corresponding CS with a red box around the vertices. For the assignment $x_1 = x_2 = x_3 = \text{FALSE}$, $(x_1 \vee x_2)$ is not satisfied whereas the rest of the clauses are satisfied.

Our objective is to show that there exists an MCS of size at most $N(k) = n(M+2) + 2k + 3(m-k) + 1$ in the tree T_θ if at least k clauses of θ are satisfied; otherwise, the size is strictly greater than $N(k)$. We now prove a set of auxiliary claims about a minimum consistent subset for (T_θ, C_θ).

▶ **Lemma 3.** *For any consistent subset V_C of size at most $N(k) = n(M+2)+2k+3(m-k)+1$ in the tree T_θ, the following are true.*

- *For any variable x_i, exactly one of the following is true.*
  - *$S_i \subset V_C$, $\overline{S}_i \cap V_C = \emptyset$, and $x_i^2, \overline{x}_i^4 \in V_C$.*
  - *$\overline{S}_i \subset V_C$, $S_i \cap V_C = \emptyset$, and $\overline{x}_i^2, x_i^4 \in V_C$*

  , and
- *$v_3 \in V_C$.*

Lemma 3, states that by strategically choosing the vertices in a variable gadget, the vertices of the tree corresponding to that variable gadget can be consistently covered by choosing its only one set of stabilizer vertices.

Proof. For a variable x_i, let $S_i \cap V_C \neq \emptyset$. Let $s_i^j \in S_i \cap V_C$. But then every vertex $v \in S_i \setminus \{s_i^j\}$ must have a vertex within distance 2 of its own color, since $dist(s_i^j, v) = 2$. Hence $S_i \subset V_C$. One can similarly prove that $\overline{S}_i \cap V_C \neq \emptyset \implies \overline{S}_i \subseteq V_C$. Also, every variable gadget contains M uniquely colored vertices and hence has at least M vertices in V_C. So, if $M >> n$, we have that $N(k) < (n+1)M$, and there exists no variable gadget that contains vertices from both S_i and $\overline{S}_i$. In other words exactly one of the following holds for every variable gadget corresponding to a variable x_i:

- $S_i \subset V_C$, $\overline{S}_i \cap V_C = \emptyset$
- $\overline{S}_i \subset V_C$, $S_i \cap V_C = \emptyset$,

Below, we look into one of these cases, and a similar argument may be made for the other case as well.

Case 1: $S_i \subset V_C$, $\overline{S}_i \cap V_C = \emptyset$. Notice that there must be a vertex in the literal path $\{x_i^1, x_i^2, x_i^3, x_i^4\}$ from $\{x_i^1, x_i^2\}$ of the variable gadget X_i since $dist(S_i, x_i^1) = 1$ and the distance to any other vertex of the same color (other than these two vertices) is more than 1. But $x_i^1 \notin V_C$, as $\overline{S}_i \cap V_C = \emptyset$ and $dist(x_i^1, \overline{S}_i) < dist(S_i, \overline{S}_i)$. Hence $x_i^2 \in V_C$.

Similarly there must be a vertex in the literal path $\{\overline{x}_i^1, \overline{x}_i^2, \overline{x}_i^3, \overline{x}_i^4\}$ of the variable gadget X_i since $dist(S_i, \overline{x}_i^1) = 3$ and the distance to any other vertex of the same color (other than $\{\overline{x}_i^2, \overline{x}_i^3, \overline{x}_i^4\}$) is more than 3. And the distance of 4 between S_i and $\overline{S_i}$ eliminates the possibility of belonging any of $\overline{x}_i^1, \overline{x}_i^2$ or $\overline{x}_i^3$ belonging in V_C. Thus, $\overline{x}_i^4 \in V_C$.

Case 2: $\overline{S}_i \subset V_C$, $S_i \cap V_C = \emptyset$. Case 2 may be argued in a manner similar to Case 1.

Moreover, V_C must contain at least one veretx from the set $\{v_1, v_2, v_3\}$. However, the distance of 4 between S_i and $\overline{S_i}$ rules out the possibility of either v_1 or v_2 being in V_C. Consequently, $v_3 \in V_C$. ◀

To satisfy the inequality in the above lemma, we now set the value of M as n^3. In the next lemma, we present a bound on the vertices from each clause gadget that are contained in a consistent subset of size at most $N(k)$. For any clause C_i, denote the corresponding clause gadget by $T_i^C = G[\{w_i^a, y_i^a, z_i^a | 1 \le a \le 7\}]$.

▶ **Lemma 4.** *In any consistent subset V_C of the tree T_θ, for each clause C_i, $2 \le |V(T_i^C) \cap V_C|$.*

Proof. There needs to be a vertex among the vertices $\{w_i^a \mid 1 \le a \le 7\}$ since they are distinctly colored from all other vertices. If this vertex belongs to $\{w_i^a \mid 1 \le a \le 4\}$, then there must also be a vertex in $\{y_i^a \mid 1 \le a \le 7\}$ since the nearest vertex of the same color (any y_i^a) is farther away than the vertex with the color of any w_i^a. Similarly, if this vertex belongs to $\{w_i^a \mid 4 \le a \le 7\}$, then there must be a vertex in $\{z_i^a \mid 1 \le a \le 7\}$. Therefore, $2 \le |V(T_i^C) \cap V_C|$. ◀

▶ **Theorem 5.** *There exists a truth assignment of the variables in θ which satisfies at least k clauses if and only if there exists a consistent subset of size at most $N(k)$ for (T_θ, C_θ).*

Proof. ($\Rightarrow$) For the forward direction, let there exist an assignment A to the variables of θ that satisfies k clauses. Consider the following set of vertices V_A. For each variable x_i if x_i is TRUE, include all the vertices of S_i in V_A. Also include x_i^2 and $\overline{x}_i^4$ in V_A. If $x_i =$ FALSE then include all the vertices of $\overline{S}_i$ in V_A. Also include x_i^4 and $\overline{x}_i^2$ in V_A.

For every satisfied clause $C_i = (y_i \vee z_i)$ with respect to A we include the following vertices in V_A. Without loss of generality assume that $y_i =$ TRUE. We include w_i^7 and z_i^1 in V_A. For every unsatisfied clause $C_i = (y_i \vee z_i)$, we include w_i^1, y_i^1 and z_i^7 in V_A. We also include v_3 in V_A.

Observe that the cardinality of V_A is $N(k) = n(M + 2) + 2k + 3(m - k) + 1$. Next, we prove that V_A is a consistent subset for T_θ. Observe that for any pair of vertices $(s_i^j, \overline{s}_i^j)$, exactly one of them is in V_A. Without loss of generality assume that $s_i^j \in V_A$. Observe that $d(s_i^j, \overline{s}_i^j) = d(\overline{s}_i^j, V_A) = 4$. If $x_i =$ TRUE then $d(x_i^j, x_i^2) \le d(x_i^j, V_A)$ and $d(\overline{x}_i^j, \overline{x}_i^4) \le d(\overline{x}_i^j, V_A)$. The case when $x_i =$ FALSE is symmetric.

For any clause gadget either w_i^1 or w_i^7 is in V_A. Without loss of generality assume that $w_i^1 \in V_A$. Observe that for every vertex w_i^j, $d(w_i^j, w_i^1) = d(w_i^j, V_A)$. Let $C_i = (y_i \vee z_i)$ be a satisfied clause and without loss of generality assume that $y_i = x_j =$ TRUE. Observe that $d(y_i^1, x_j^2) = 6$, and $d(y_i^3, V_A) = 6$, and $d(y_i^a, x_j^2) = d(y_i^a, V_A)$. For any unsatisfied clause $C_i = (y_i \vee z_i)$ observe that $d(y_i^a, x_j^2) = d(y_i^a, V_A)$. Also for any v_j where $1 \leq j \leq 3$ $d(v_j, v_3) = d(v_j, V_A)$. Therefore V_A is a consistent subset for T_θ.

($\Leftarrow$) In the backward direction, let there be a consistent subset V_C of size at most $N(k)$ for (T_θ, C_θ). We know from Lemma 3 that either $S_i \subset V_C$ or $\overline{S}_i \subset V_C$. From Lemma 3 any such solution has at least $n(M+2)+1$ many vertices from V_C outside the clause gadgets leaving at most $2k + 3(m-k)$ that may be chosen from the clause gadgets.

Each clause gadget comprises of vertices of three distinct colors: one color exclusive to the clause itself and two colors dedicated to literals. An essential insight is that if there are no vertices in V_C of colors specific to the literals from a clause in the variable gadgets, then such a clause gadget must contain at least three vertices from V_C. This assertion is valid because the distance between two sets of vertices of the same color (corresponding to the same literal in two clauses) across any two clauses is at least 8, while vertices in V_C of clause-specific colors are at a distance of at most 6.

This fact, coupled with Lemma 4 implies that there are at least k clauses for whom colors specific to at least one of their literals have the vertices in V_C of the same color from the variable gadgets. Making the same literals true and setting other variables arbitrarily gives us an assignment that satisfies at least k clauses. ◀

4 MCS for Trees: A Parameterized Algorithm

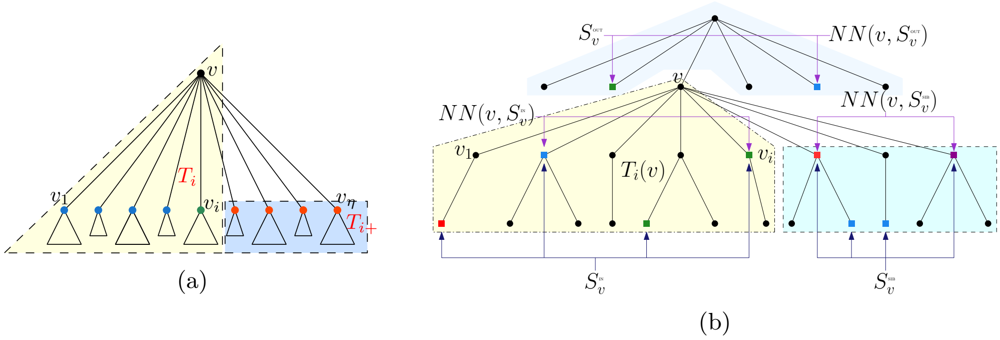
In this section, we consider the optimization version of the MCS problem for the trees.

Minimum Consistent Subset for Trees **Parameter:** c
Input: A rooted tree T, whose vertices $V(T)$ are coloured with a set C of c colours.
Question: Find the minimum possible size of a consistent subset (MCS) for T?

We consider T as a rooted tree by taking an arbitrary vertex r as its root. We use $V(T')$ to denote the vertices of a subtree T' of T, and $C(U) \subseteq C$ to denote the subset of colors assigned to subset of vertices $U \subseteq V$, and $C(u)$ to denote the color attached to the vertex $u \in V(T)$. For any vertex v, let η_v denote the number of children of v and we denote the children of v by $v_1, v_2, \cdots, v_{\eta_v}$. We denote the subtree rooted at a vertex v by $T(v)$. For any vertex v and any integer $i < \eta_v$, we use $T_{i+}(v)$ to denote the union of subtrees rooted at v_{i+1} to v_{η_v}, and $T_i(v)$ to denote the subtree rooted at v and containing first i many children of v. Thus, $T_{i+}(v) = \cup_{i+1 \leq j \leq \eta_v} T(v_j)$, which is a forest, and $T_i(v) = T(v) \setminus T_{i+}(v)$. In Figure 2(a), the light yellow part is $T_i(v)$, and the light sky-colored part is $T_{i+}(v)$. We define $T^{out}(v) = T \setminus T(v)$.

For any positive integer d and for any vertex $v \in V(T)$, a set of vertices $U \subset V(T)$ is called d-equidistant from v if $d(u_i, v) = d$ for all $u_i \in U$. Any subset of vertices U spans a set of colors $C' \subseteq C$ if $C(U) = C'$. For any vertex $v \in V(T)$, we use $\mathcal{E}_i^{\text{SIB}}(v, d, C')$ (resp. $\mathcal{E}_i^{\text{OUT}}(v, d, C')$) to denote the set of subsets of vertices in $T_{i+}(v)$ (resp. $T \setminus T(v)$), which are d-equidistant from v and span the colors in C'. Next, we define a *(partial) consistent subset* for a subtree $T_i(v)$.

Intuition. Our dynamic programming (DP) routine exploits the key observation that a (partial) consistent subset (formally defined below) for a subtree $T(v)$ can be computed in FPT time. This computation is possible given the distance to the closest vertex in the consistent subset that lies outside $T(v)$ and the colors of those vertices. The entries in our



(a)

(b)

Figure 2 Illustration of the bottom-up dynamic programming routine, ■ vertices denotes consistent subset. $\delta_S^{\textsc{in}} = 1, \delta_S^{\textsc{out}} = 2, \delta_S^{\textsc{sib}} = 1$.

DP table store the minimum size of partial consistent subsets for all subtrees $T_i(v)$. These subsets are defined based on six parameters: distance to the closest vertex from v in the consistent subset in $T_i(v)$, $T^{out}(v)$ and $T_{i+}(v)$ and colors of these three set of closest vertices.

▶ **Definition 6.** *Let $d^{\textsc{in}} \in \mathbb{Z}_0^+$ and $d^{\textsc{out}}, d^{\textsc{sib}} \in \mathbb{Z}^+$, and let three subsets of colors $C^{\textsc{in}}, C^{\textsc{out}}, C^{\textsc{sib}} \subseteq C$. A (partial) consistent subset of the subtree $T_i(v)$ with respect to the parameters $d^{\textsc{in}}$, $d^{\textsc{out}}$, $d^{\textsc{sib}}$, $C^{\textsc{in}}$, $C^{\textsc{out}}$, $C^{\textsc{sib}}$ is defined as a set of vertices $W \subseteq V(T_i(v))$ such that for any arbitrary subset $X \in \mathcal{E}_i^{SIB}(v, d^{\textsc{sib}}, C^{\textsc{sib}})$ and $Y \in \mathcal{E}_i^{OUT}(v, d^{\textsc{out}}, C^{\textsc{out}})$ (assuming they exist), W satisfies the following (see Figure 2(b)):*

- *$d(v, W) = d^{\textsc{in}}$. (i.e., the distance of v to its nearest member(s) in W is $d^{\textsc{in}}$)*
- *$C(\mathrm{NN}(v, W)) = C^{\textsc{in}}$. (i.e., $C^{\textsc{in}}$ is the set of colors of the nearest members of v in the set W)*
- *For every vertex $u \in T_i(v)$, $C(u) \in C(\mathrm{NN}(u, W \cup X \cup Y))$.*

Note that for some values of $d^{\textsc{in}}, d^{\textsc{out}}, d^{\textsc{sib}}, C^{\textsc{in}}, C^{\textsc{out}}, C^{\textsc{sib}}$ there may not exist any (partial) consistent subset for $T_i(v)$; in such a case we define it is undefined. Also note that, for some values, the (partial) consistent subset can be empty as well, such as when $d^{\textsc{in}} = \infty$ and $C(u) \in C(\mathrm{NN}(u, X \cup Y))$ for every vertex $u \in T_i(v)$. For ease of notation, we will denote a (partial) consistent subset for $T_i(v)$ as a consistent subset with respect to the parameters $d^{\textsc{in}}, d^{\textsc{out}}, d^{\textsc{sib}}, C^{\textsc{in}}, C^{\textsc{out}}, C^{\textsc{sib}}$.

Consider an arbitrary consistent subset S_T of T, an arbitrary vertex $v \in V(T)$ and an integer $i \in [\eta_v]$ (see Figure 2(b)). For any vertex $v \in V(T)$ and $1 \leq i \leq \eta_v$, define $S_v^{\textsc{in}} = S_T \cap V(T_i(v))$, $S_v^{\textsc{sib}} = S_T \cap V(T_{i+}(v))$, and $S_v^{\textsc{out}} = S_T \cap V(T \setminus T(v))$. Also define $\delta_S^{\textsc{in}} = d(v, S_v^{\textsc{in}})$, $C_S^{\textsc{in}} = C(\mathrm{NN}(v, S_v^{\textsc{in}}))$, $\delta_S^{\textsc{sib}} = d(v, S_v^{\textsc{sib}})$, $C_S^{\textsc{sib}} = C(\mathrm{NN}(v, S_v^{\textsc{sib}}))$, $\delta_S^{\textsc{out}} = d(v, S_v^{\textsc{out}})$, $C_S^{\textsc{out}} = C(\mathrm{NN}(v, S_v^{\textsc{out}}))$. Let W be any arbitrary (partial) consistent subset with respect to the parameters $\delta_S^{\textsc{in}}, \delta_S^{\textsc{out}}, \delta_S^{\textsc{sib}}, C_S^{\textsc{in}}, C_S^{\textsc{out}}, C_S^{\textsc{sib}}$ (see Definition 6). Next, we have the following lemma.

▶ **Lemma 7.** *$S_W = (S_T \setminus S_v^{\textsc{in}}) \cup W$ is a consistent subset for T.*

Proof. Suppose that $A = W \cup \mathrm{NN}(v, S_v^{\textsc{sib}}) \cup \mathrm{NN}(v, S_v^{\textsc{out}})$ is the set of vertices which are either in W or in the nearest neighbor of v outside $T_i(v)$ in S_W. We will show that for any vertex $u \in T_i(v)$, $NN(u, S_W) \subseteq A$ and there is a vertex in $NN(u, A)$ of the color same as u. Similarly, let $B = (S_W \setminus W) \cup \mathrm{NN}(v, W)$. We show that for any vertex w outside $T_i(v)$, $NN(w, S_W) \subseteq B$ and there is a vertex in $NN(w, B)$ of the color same as w. Please note that A and B are not necessarily disjoint.

Consider a vertex $u \in T_i(v)$ and $w \in T \setminus T_i(v)$. Since $\mathrm{NN}(v, S_v^{\text{SIB}}) \in \mathcal{E}_i^{\text{SIB}}(v, \delta_S^{\text{SIB}}, C_S^{\text{SIB}})$, $\mathrm{NN}(v, S_v^{\text{OUT}}) \in \mathcal{E}_i^{\text{OUT}}(v, \delta_S^{\text{OUT}}, C_S^{\text{OUT}})$, and W is a consistent subset with respect to the parameters $\delta_S^{\text{IN}}, \delta_S^{\text{OUT}}, \delta_S^{\text{SIB}}, C_S^{\text{IN}}, C_S^{\text{OUT}}, C_S^{\text{SIB}}$, we have $C(u) \in C(\mathrm{NN}(u, W \cup \mathrm{NN}(v, S_v^{\text{SIB}}) \cup \mathrm{NN}(v, S_v^{\text{OUT}}))) = C(\mathrm{NN}(u, A))$. Also, as S_T is a consistent subset, we have $C(w) \in C(\mathrm{NN}(w, S_T \setminus S_v^{\text{IN}}) \cup C_S^{\text{IN}})$. From the properties of W, we have $C(\mathrm{NN}(v, W)) = C_S^{\text{IN}}$. Hence, $C(w) \in C(\mathrm{NN}(w, B))$. Thus, to prove the result, it is enough to show that (i) no vertex from $B \setminus A$ can be the closest to the vertex u in the set S_W, and (ii) no vertex from $A \setminus B$ can be the closest vertex of w in the set S_W. We prove these two claims by contradiction.

Assume that *Claim (i)* is false. Then, there will be a vertex $x \in B \setminus A$, which is closest to u. Note that, $(B \setminus A) \cap ((T_i(v) \cup \mathrm{NN}(v, S_v^{\text{SIB}}) \cup \mathrm{NN}(v, S_v^{\text{OUT}})) = \emptyset$. Hence we have $d(u, x) = d(u, v) + d(v, x)$, $d(v, x) > min(\delta_S^{\text{SIB}}, \delta_S^{\text{OUT}})$. This is a contradiction as the closest vertex from v in $S_W \cup (T \setminus T_i(v))$ (if it exists) is at distance $min(\delta_S^{\text{SIB}}, \delta_S^{\text{OUT}}) = d(v, \mathrm{NN}(v, S_v^{\text{SIB}}) \cup \mathrm{NN}(v, S_v^{\text{OUT}}))$. Hence, x cannot be the closest vertex of u in S_W. Thus, *Claim (i)* follows.

Now, assume that *Claim (ii)* is false. Then there is a vertex $y \in A \setminus B$, which is closest to w. As $w \notin T_i(v)$ and $y \in T_i(v)$, we have $d(w, y) = d(w, v) + d(v, y)$. Since $d(v, \mathrm{NN}(v, W)) = \delta_S^{\text{IN}}$ and $w \in (A \setminus B = W \setminus \mathrm{NN}(v, W))$, we have $d(v, y) > \delta_S^{\text{IN}}$. This contradicts the fact that the closest vertex from v in $S_W \cup T_i(v)$ (if it exists) is at distance δ^{IN} from v. Hence, *Claim (ii)* is true. ◀

Motivated by Lemma 7, we design the following algorithm based on the dynamic programming technique. For each choice of $v \in V(T)$, $i \in [\eta_v]$, $\delta_v^{\text{IN}} \in [n] \cup \{0, \infty\}$, $\delta_v^{\text{OUT}} \in [n] \cup \{\infty\}$, $\delta_v^{\text{SIB}} \in [n] \cup \{\infty\}$, and $C_v^{\text{IN}}, C_v^{\text{OUT}}, C_v^{\text{SIB}} \subseteq C$, we define a subproblem which computes the cardinality of a minimum sized (partial) consistent subset for the subtree $T_i(v)$ with respect to the parameters $\langle \delta_v^{\text{IN}}, \delta_v^{\text{OUT}}, \delta_v^{\text{SIB}}, C_v^{\text{IN}}, C_v^{\text{OUT}}, C_v^{\text{SIB}} \rangle$, and denote its size by $P(T_i(v), \delta_v^{\text{IN}}, \delta_v^{\text{OUT}}, \delta_v^{\text{SIB}}, C_v^{\text{IN}}, C_v^{\text{OUT}}, C_v^{\text{SIB}})$. Let us use $\delta_v^{\text{MIN}} = \min(\delta_v^{\text{IN}}, \delta_v^{\text{OUT}}, \delta_v^{\text{SIB}})$.

$$A = \begin{cases} C_v^{\text{IN}} & \text{if } \delta_v^{\text{IN}} = \delta_v^{\text{MIN}} \\ \emptyset & \text{otherwise} \end{cases}, \quad B = \begin{cases} C_v^{\text{SIB}} & \text{if } \delta_v^{\text{SIB}} = \delta_v^{\text{MIN}} \\ \emptyset & \text{otherwise} \end{cases}, \quad D = \begin{cases} C_v^{\text{OUT}} & \text{if } \delta_v^{\text{OUT}} = \delta_v^{\text{MIN}} \\ \emptyset & \text{otherwise} \end{cases}$$

We define $C_v^{\text{MIN}} = A \cup B \cup D$. Note that δ_v^{MIN} is the distance of the closest vertex to v in any $X \in \mathcal{E}_i^{\text{SIB}}(v, \delta_v^{\text{SIB}}, C_v^{\text{SIB}})$, any $Y \in \mathcal{E}_i^{\text{OUT}}(v, \delta_v^{\text{OUT}}, C_v^{\text{OUT}})$ or the consistent subset, and that C_v^{MIN} denotes the colors of all such vertices.

To compute any DP entry, we take into account the following six cases. The first two cases are for checking whether a DP entry is valid. The third case considers the scenario in which v is part of the solution; the fourth, fifth, and sixth cases collectively consider the scenario in which v is not in the solution.

Case 1: If $C(v) \notin C_v^{\text{MIN}}$, return undefined.
Case 2: $\delta_v^{\text{IN}} = 0$ and $C_v^{\text{IN}} \neq \{C(v)\}$. Here, return ∞.
Case 3: $\delta_v^{\text{IN}} = 0$ and $C_v^{\text{IN}} = \{C(v)\}$. Here, return $P(T_i(v), \delta_v^{\text{IN}}, \delta_v^{\text{OUT}}, \delta_v^{\text{SIB}}, C_v^{\text{IN}}, C_v^{\text{OUT}}, C_v^{\text{SIB}}) =$

$$1 + \sum_{1 \leq j \leq i} \left\{ \min_{\delta, C'} P\Big(T_{\eta_{v_j}}(v_j), \delta, 1, \infty, C', \{C(v)\}, \emptyset\Big) \right\}$$

Explanation. Case 1 and Case 2 are self-explanatory. Case 3 implies that the vertex v is included in the consistent subset. Consequently, for the optimal solution, we need to determine a consistent subset for each tree rooted at a child v_j of v, independently of each other, assuming that v is part of the consistent subset (refer to Figure 2(a)). For every child v_j of v, we iterate through all possible choices of $C' \subseteq C$ and $\delta_{v_j}^{\text{IN}} = \delta \in \{1, \dots, h(T(v_j))\} \cup \{\infty\}$

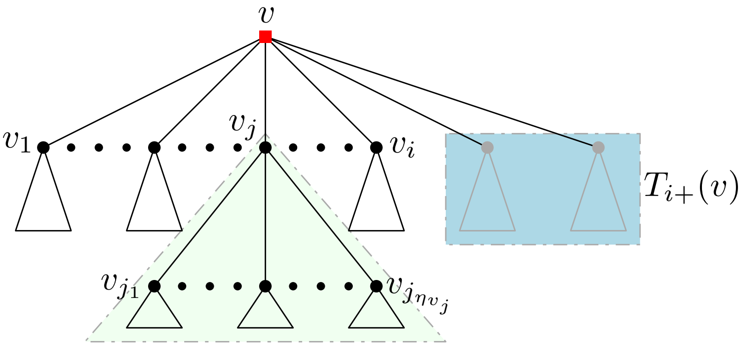


Figure 3 Illustration of the Case 3, where $\delta_v^{\text{IN}} = 0$.

where $h(T(v_j))$ is the height of the tree rooted at v_j, to identify the minimum consistent subset for $T_{\eta_{v_j}}(v_j)$. This is done with the constraints that the closest vertex in the consistent subset inside $T_{\eta_{v_j}}(v_j)$ is at a distance of δ and spans C'. For any vertex in $T(v_j)$, the path of the closest vertex of its own color outside $T_i(v)$ has to pass through v, which is considered to be in the consistent subset and has color $C(v)$. Thus, $\delta_v^{\text{OUT}} = 1$ is taken for the tree $T_{\eta_{v_j}}(v_j)$. Since we are solving for the complete tree rooted at v_j with no siblings, we set the distance to the closest sibling vertex as $\delta_v^{\text{SIB}} = \infty$ and the corresponding color set as $\emptyset$.

Notations for Subsequent Cases. In the rest of the section, we consider three more cases where $\delta_v^{\text{IN}} > 0$, and hence $\delta_v^{\text{IN}}, \delta_v^{\text{OUT}}, \delta_v^{\text{SIB}} > 0$. Intuitively, while solving the problem recursively, we will recursively solve MCS in $T_{i-1}(v)$ and $T_{\eta_{v_i}}(v_i)$. We try all possible sets of choices of C_a, C_b with $C_v^{\text{IN}} = C_a \cup C_b$, and recursively solve for a solution assuming that the nodes of colors in C_a are present in $T_{i-1}(v)$ at a distance of δ_v^{IN} (if $C_a \neq \emptyset$ and such choices are feasible) and nodes of colors in $C_b \subseteq C_v^{\text{IN}}$ are present in $T_{\eta_{v_i}}(v_i)$ at a distance of $\delta_v^{\text{IN}} - 1$ from v_i (if $C_b \neq \emptyset$ and such choices are feasible).

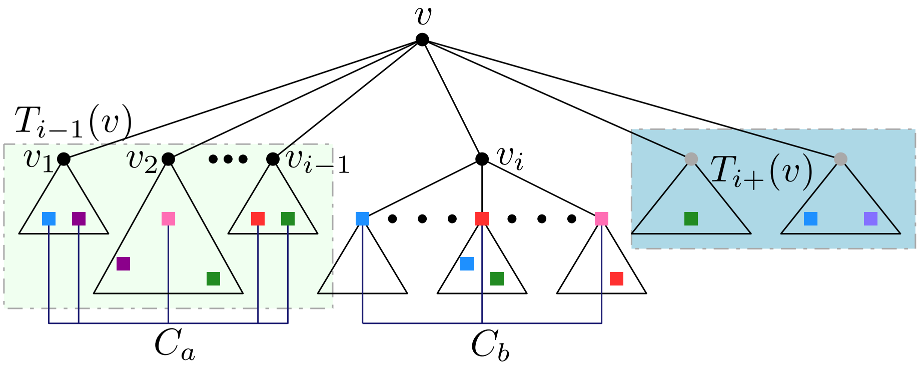


Figure 4 Illustration of the Case 4.

Case 4: $C_a, C_b \neq \emptyset$. In this case, the closest vertices in the consistent subset from v in both $T_{i-1}(v)$ and $T_{\eta(v_i)}(v_i)$ are located at a distance of precisely δ_v^{IN}. We start by defining the following. Let $\delta_x = \min(\delta_v^{\text{IN}}, \delta_v^{\text{SIB}})$ and recall $\delta_v^{\text{MIN}} = \min(\delta_v^{\text{IN}}, \delta_v^{\text{OUT}}, \delta_v^{\text{SIB}})$.

Observe that both $T_{i+}(v)$ and $T_{\eta_{v_i}}(v_i)$ contains siblings of $T_{i-1}(v)$. Thus, in a hypothetical consistent subset, which is compatible with the current partial consistent subset, for the tree T, the closest vertices from v in $T_{(i-1)+}(v)$ are either in $T_{i+}(v)$ or $T_{\eta_{v_i}}(v_i)$. Here, δ_x is trying to capture this distance information, and C^{SIB}_{i-1} represents the colors of such vertices. Similarly, from v_i in a hypothetical consistent subset CS for T, which is compatible with the current partial consistent subset, $\delta^{\text{MIN}}_v + 1$ denotes the distance to the vertices in CS contained in $T \setminus T(v_i)$. Note that these vertices can be either in $T_{(i-1)}(v)$ or in $T_{i+}(v)$ or in $T \setminus T(v)$. C^{OUT}_i represents the colors of such vertices.

$$C^{\text{SIB}}_{i-1} = \begin{cases} C_b & \text{if } \delta^{\text{IN}}_v < \delta^{\text{SIB}}_v \text{ and } C_b \neq \emptyset \\ C_b \cup C^{\text{SIB}}_v & \text{if } \delta^{\text{IN}}_v = \delta^{\text{SIB}}_v \\ C^{\text{SIB}}_v & \text{otherwise} \end{cases}$$

Also, define $C^{\text{OUT}}_i = A \cup B \cup D$, where

$$A = \begin{cases} C_a \text{ if } \delta^{\text{IN}}_v = \delta^{\text{MIN}}_v \\ \emptyset \quad \text{otherwise} \end{cases}, \; B = \begin{cases} C^{\text{SIB}}_v \text{ if } \delta^{\text{SIB}}_v = \delta^{\text{MIN}}_v \\ \emptyset \quad \text{otherwise} \end{cases}, \quad D = \begin{cases} C^{\text{OUT}}_v \text{ if } \delta^{\text{OUT}}_v = \delta^{\text{MIN}}_v \\ \emptyset \quad \text{otherwise} \end{cases}$$

Now we can safely assume that there is a δ_x-equidistant set from v contained in $T_{(i-1)+}(v)$ that spans C^{SIB}_{i-1}.

$$\text{Return } P(T_i(v), \delta^{\text{IN}}_v, \delta^{\text{OUT}}_v, \delta^{\text{SIB}}_v, C^{\text{IN}}_v, C^{\text{OUT}}_v, C^{\text{SIB}}_v) = \min_{C_a, C_b} \Big(P\Big(T_{i-1}(v), \delta^{\text{IN}}_v, \delta^{\text{OUT}}_v, \delta_x, C_a, C^{\text{OUT}}_v, C^{\text{SIB}}_{i-1}\Big) + P\Big(T_{\eta_{v_i}}(v_i), \delta^{\text{IN}}_v - 1, \delta^{\text{MIN}}_v + 1, \infty, C_b, C^{\text{OUT}}_i, \emptyset\Big)\Big)$$

Explanation. In this case we iterate over all possible choices of C_a and C_b, assuming $C_a, C_b \neq \emptyset$ and $C_a \cup C_b = C^{\text{IN}}_v$. In the first part of the recursive formula, we recursively solve the problem for the tree $T_{i-1}(v)$ with the restriction that we have to include a set of vertices of color C_a in $T_{i-1}(v)$ at distance δ^{IN}_v from v (see Figure 2(b)). The restriction on C^{OUT}_v and δ^{OUT}_v among the vertices in $T \setminus T(v)$ remains the same as that of the parent problem. Regarding C^{SIB}_v and δ^{SIB}_v, observe that vertices in $T(v_i)$ and T_{i+} are part of $T_{(i-1)+}$. Therefore the parameters for the sibling depend on the value of δ^{IN}_v and δ^{SIB}_v, and accordingly, we have defined C^{SIB}_{i-1}.

In the second part of the recursive formula, we are solving the problem recursively for the tree $T_{\eta_{v_i}}(v_i)$, with the restriction that, in the consistent set in the consistent set we have to include a set of vertices from $T_{\eta_{v_i}}(v_i)$ which are of colors C_b, and at distance $\delta^{\text{IN}}_v - 1$ from v_i. Observe that the vertices in $T_{i-1}(v)$, $T_{i+}(v)$ and $T \setminus T(v)$ are all outside $T(v_i)$. Thus the restriction on the distance to the vertices on the consistent subset outside $T(v_i)$ and their colors depend on the values of $\delta^{\text{IN}}_v, \delta^{\text{SIB}}_v$ and δ^{OUT}_v. Thus the distance δ^{OUT}_v of this subproblem is defined as $\delta^{\text{MIN}}_v = \min(\delta^{\text{IN}}_v, \delta^{\text{SIB}}_v, \delta^{\text{OUT}}_v)$, and the set of colors C^{OUT}_i is defined accordingly. As we are solving for the whole tree rooted at v_i, there are no siblings; so $\delta^{\text{SIB}}_v = 0$ and $C^{\text{SIB}}_i = \emptyset$.

Case 5: $C_a = \emptyset$ and $C_b = C^{\text{IN}}_v$. Note that in this case, the closest vertices in the consistent subset from v in $T_{\eta(v_i)}(v_i)$ are located at a distance of δ^{IN}_v while in $T_{i-1}(v)$, they are located at a distance of at least $\delta \geq \delta^{\text{IN}}_v + 1$

We iterate over all values $\delta > \delta^{\text{IN}}_v$ and all possible choices of colors to find the size of a minimum consistent subset. We define δ_x and C^{SIB}_{i-1} the same as in Case 4. For any values $\delta > \delta^{\text{IN}}_v$ and $C \subseteq [c]$, we define $\delta^{\text{MIN}}_v(\delta, C) = \min(\delta, \delta^{\text{SIB}}_v, \delta^{\text{OUT}}_v)$, and $C^{\text{OUT}}_i(\delta, C) = A \cup B \cup D$ where

$$A = \begin{cases} C \text{ if } \delta = \delta^{\text{MIN}}_v \\ \emptyset \text{ otherwise} \end{cases}, \quad B = \begin{cases} C^{\text{SIB}}_v \text{ if } \delta^{\text{SIB}}_v = \delta^{\text{MIN}}_v \\ \emptyset \quad \text{otherwise} \end{cases}, \quad D = \begin{cases} C^{\text{OUT}}_v \text{ if } \delta^{\text{OUT}}_v = \delta^{\text{MIN}}_v \\ \emptyset \quad \text{otherwise} \end{cases}$$

From v_i in a hypothetical consistent subset CS for T, which is compatible with the current partial consistent subset, $\delta_v^{\textsc{min}}(\delta, C)$ denotes the distance to the vertices in CS contained in $T \setminus T(v_i)$. Note that these vertices can be either in $T_{(i-1)}(v)$ or in $T_{i+}(v)$ or in $T \setminus T(v)$. $C_i^{\textsc{out}}(\delta, C)$ represents the colors of such vertices.

$$\text{Return } P(T_i(v), \delta_v^{\textsc{in}}, \delta_v^{\textsc{out}}, \delta_v^{\textsc{sib}}, C_v^{\textsc{in}}, C_v^{\textsc{out}}, C_v^{\textsc{sib}}) = \min_{\delta > \delta_v^{\textsc{in}}, C \subseteq [c]} \Big(P\Big(T_{i-1}(v), \delta, \delta_v^{\textsc{out}}, \delta_x, C, C_v^{\textsc{out}}, C_{i-1}^{\textsc{sib}}\Big) + P\Big(T_{\eta_{v_i}}(v_i), \delta_v^{\textsc{in}} - 1, \delta_v^{\textsc{min}}(\delta, C) + 1, \infty, C_b, C_i^{\textsc{out}}(\delta, C), \emptyset\Big)\Big)$$

Explanation. The explanation for this case is the same as Case 4 except for the fact that we have to make sure that the closest vertex chosen in the consistent subset from $T_{i-1}(v)$ is at distance at least $\delta_v^{\textsc{in}} + 1$.

Case 6: $C_b = \emptyset$ and $C_a = C_v^{\textsc{in}}$.

Here we consider the case when $C_b = \emptyset$ and $C_a = C_v^{\textsc{in}}$. Note that in this case, the closest vertices in the consistent subset from v in $T_{i-1}(v)$ are located at a distance of $\delta_v^{\textsc{in}}$ while in $T_{\eta(v_i)}(v_i)$, they are located at a distance of at least $\delta \geq \delta_v^{\textsc{in}} + 1$

We define $\delta_v^{\textsc{min}}(\delta, C) = \min(\delta_v^{\textsc{in}}, \delta_v^{\textsc{sib}}, \delta_v^{\textsc{out}})$. We define $C_i^{\textsc{out}}(\delta, C) = A \cup B \cup D$ where

$$A = \begin{cases} \mathcal{C}_a \text{ if } \delta_v^{\textsc{in}} = \delta_v^{\textsc{min}} \\ \emptyset \text{ otherwise} \end{cases}, \quad B = \begin{cases} C_v^{\textsc{sib}} \text{ if } \delta_v^{\textsc{sib}} = \delta_v^{\textsc{min}} \\ \emptyset \quad \text{otherwise} \end{cases}, \quad D = \begin{cases} C_v^{\textsc{out}} \text{ if } \delta_v^{\textsc{out}} = \delta_v^{\textsc{min}} \\ \emptyset \quad \text{otherwise} \end{cases}$$

For any values $\delta > \delta_v^{\textsc{in}}$ and $C \subseteq [c]$ we define $\delta_x(\delta, C) = \min(\delta_v^{\textsc{sib}}, \delta)$ We define $C_{i-1}^{\textsc{sib}}(\delta, C) = E \cup F$ where

$$E = \begin{cases} \mathcal{C} \text{ if } \delta = \delta_x(\delta, C) \\ \emptyset \text{ otherwise} \end{cases}, \quad F = \begin{cases} C_v^{\textsc{sib}} \text{ if } \delta_v^{\textsc{sib}} = \delta_x(\delta, C) \\ \emptyset \quad \text{otherwise} \end{cases}$$

The meaning and the reasoning behind the defining of $\delta_v^{\textsc{min}}(\delta, C)$ and $C_i^{\textsc{out}}(\delta, C)$ remains the same as in previous cases. Thus, in a hypothetical consistent subset, which is compatible with the current partial consistent subset, for the tree T, the closest vertices from v in $T_{(i-1)+}(v)$ are either in $T_{i+}(v)$ or $T_{\eta_{v_i}}(v_i)$. Here, $\delta_x(\delta, C)$ is trying to capture this distance information, and $C_{i-1}^{\textsc{sib}}(\delta, C)$ represents the colors of such vertices.

In this case we return:

$$\text{Return } P(T_i(v), \delta_v^{\textsc{in}}, \delta_v^{\textsc{out}}, \delta_v^{\textsc{sib}}, C_v^{\textsc{in}}, C_v^{\textsc{out}}, C_v^{\textsc{sib}}) = \min_{\delta > \delta_v^{\textsc{in}},\ C \subseteq [c]} \Big(P\Big(T_{i-1}(v), \delta_v^{\textsc{in}}, \delta_v^{\textsc{out}}, \delta_x(\delta, C), C_a, C_v^{\textsc{out}}, C_{i-1}^{\textsc{sib}}(\delta, C)\Big) + P\Big(T_{\eta_{v_i}}(v_i), \delta, \delta_v^{\textsc{min}} + 1, \infty, C_b, C, \emptyset\Big)\Big)$$

Explanation. The explanation for this case is the same as the previous two cases.

Running time of the algorithm. The total number of choices of $\delta_v^{\textsc{in}}, \delta_v^{\textsc{out}}, \delta_v^{\textsc{sib}}, C_v^{\textsc{in}}, C_v^{\textsc{out}}$ and $C_v^{\textsc{sib}}$ is bounded by $n^3 2^{3c}$. For each choice C_a and C_b, the algorithm takes at-most $n2^c$ time to go through all possible entries of δ and C (in **case 5** and **case 6**) and there are at-most 2^{2c} choices of C_a and C_b. The recursion runs for at most n^2 times. Hence, the worst-case running time of the algorithm is $\mathcal{O}(2^{6c} n^6)$.

5 NP-hardness of MCS for Interval Graphs

A graph H is said to be an interval graph if there exists an interval layout of the graph H, or in other words, for each node, $v_i \in V(H)$ one can assign an interval α_i on the real line such that $(v_i, v_j) \in E(H)$ if and only if α_i and α_j (completely or partially) overlap in the layout of those intervals.

We prove that the Minimum Consistent Subset problem is NP-complete even when the input graph is an interval graph. We present a reduction from the Vertex Cover problem for cubic graphs. It is known that Vertex Cover remains NP-complete even for cubic graphs [10]. For any set of intervals $\mathcal{I}$, let $G(\mathcal{I})$ be the interval graph corresponding to the set of intervals $\mathcal{I}$.

Interval Graph Construction.

Let G be any cubic graph, where $V(G) = \{v_1, \ldots, v_n\}$ is the set of vertices, and $E(G) = \{e_1, \ldots, e_m\}$ is the set of edges in G. We create the set of intervals $\mathcal{I}_G$ for G on a real line $\mathcal{L}$. The set of intervals in $\mathcal{I}_G$ is represented by intervals of three different sizes, *medium*, *small* and *large*, where each medium interval is of unit length, each small interval is of length $\epsilon << \frac{1}{2n^3}$ and the length of the large interval is $\ell >> 2n$.

We define $\mathcal{I}_G = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{I}_3 \cup \mathcal{I}_4$ where $\mathcal{I}_1$ contains $2m$ *medium* size intervals (two intervals for each edge) and defined as $\mathcal{I}_1 = \{I(e_i, v_j) : e_i = (v_j, y) \in E(G), y \in V(G)\}$. We set color c_i to the interval $I(e_i, v_j)$. $\mathcal{I}_2$ contain $n \cdot n^3$ *small* intervals of color c_{m+1}, and $\mathcal{I}_3$ contain $n \cdot n^4$ *small* intervals of color c_{m+1}. $\mathcal{I}_4$ contains one large interval I_ℓ of color c_1.

We create the following vertex gadget X_i for each vertex $v_i \in V(G)$. X_i contains the following *medium* size intervals $\{I(e, v_i) : e = (v_i, x) \in E(G)\}$ corresponding to the edges that are incident on v_i. These intervals span the same region s_i of unit length on the real line $\mathcal{L}$; hence they are mutually completely overlapping. In the vertex gadget X_i, we also include a total of n^3 mutually non-overlapping small intervals in the set $\mathcal{I}_2$. Span of all the small intervals in X_i is contained in the span s_i of the *medium* sized intervals in X_i (see Figure 5).

Each vertex gadget is placed one after another (in a non-overlapping manner) along the line $\mathcal{L}$ in an arbitrary order, such that a total of n^4 mutually non-overlapping small intervals can be drawn between two consecutive vertex gadgets. Thus, $\mathcal{I}_3$ contains n sets of n^4 non-overlapping small intervals. Finally, $\mathcal{I}_4$ contains a single large interval I_ℓ that contains all the intervals in $\mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{I}_3$. This completes the construction.

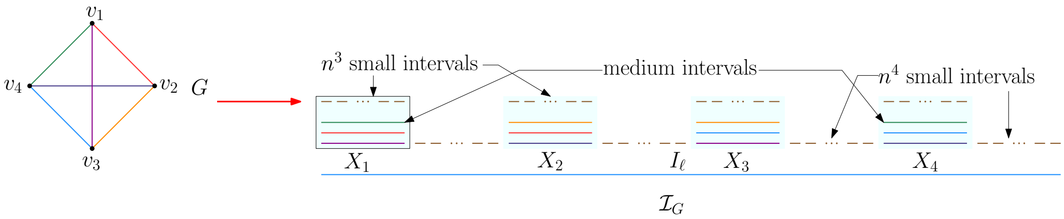


Figure 5 An example reduction.

▶ **Lemma 8.** *The graph G has a vertex cover of size at most k if and only if the corresponding interval graph $G(\mathcal{I}_G)$ has a consistent subset of size at most $K = k(3 + n^3)$.*

Proof. With a slight abuse of notations, we will denote the vertex in $G(\mathcal{I}_G)$ corresponding to an interval $I \in \mathcal{I}_G$ by I. ($\Rightarrow$) Let $A \subseteq V(G)$ be a vertex cover of G. Consider the set of intervals $\mathcal{I}_A = \bigcup_{v_i \in A} X_i$. We prove that $\mathcal{I}_A$ is a consistent subset of $G(\mathcal{I}_G)$. Note that as $|X_i| = 3 + n^3$, $|\mathcal{I}_A| = k(3 + n^3)$.

As the vertices in A cover all the edges in G, $\mathcal{I}_A$ must contain at least one interval of each color $\{c_1, \cdots, c_m\}$. As the unit intervals in X_i associated with a vertex $v_i \in A$ are of colors different from the color of the small intervals in X_i, $\mathcal{I}_A$ contains at least n^3 small intervals. Therefore $\mathcal{I}_A$ contains at least one interval from each color in $\{c_1, \ldots c_{m+1}\}$. Observe that, (i) the interval $I_\ell \in \mathcal{I}_4$ of color c_1 contains an interval of color c_1 that corresponds to the edge e_1, and (ii) the distance between any two nodes corresponding to medium intervals in two different vertex gadgets of $G(\mathcal{I}_G)$ is 2 (via the node corresponding to the interval I_ℓ in $G(\mathcal{I}_G)$). Thus, $\mathcal{I}_A$ is a consistent subset.

($\Leftarrow$) Let $\mathcal{I}_B \subseteq \mathcal{I}_G$ be any consistent subset of $G(\mathcal{I}_G)$ of cardinality $(3 + n^3)k$. Now, if $\mathcal{I}_B$ contains I_ℓ then $|\mathcal{I}_G| - 2 \leq |\mathcal{I}_B| \leq |\mathcal{I}_G|$ because if $e_1 = (v_i, v_j)$ be the edge of color c_1, then we can only do not take the medium intervals of color c_1 from the vertex gadget X_i and X_j in $\mathcal{I}_B$ because they are covered by I_ℓ, which contradicts the fact that $|\mathcal{I}_B| = (3 + n^3)k$. Thus, we have $I_\ell \notin \mathcal{I}_B$.

By the definition, $\mathcal{I}_B$ contains at least one color from $\{c_1, \cdots, c_m, c_{m+1}\}$. Also, if $\mathcal{I}_B$ contains one interval from the gadget X_i of any vertex v_i, then it must contain all the intervals from X_i; otherwise, it can not be a consistent subset. Thus $\mathcal{I}_B$ is the union $\mathcal{X}$ of at most k sets from $\{X_i : i \in [n]\}$, and it contains at least one interval from each color $\{c_1, \cdots, c_m\}$, and a few intervals of color c_{m+1}. Now, consider the set $V_B = \{v_i : X_i \in \mathcal{X}\}$, which is a vertex cover for the graph G of size at most k.

Hence, the lemma is proved. ◀

6 Conclusion

We have shown that MCS is NP-complete for trees and interval graphs, and have given an exact algorithm parameterized w.r.t. the number of colors for trees. As a direction for future research, possibilities of approximating MCS can be explored for interval graphs and related graph classes like circle graphs, circular arc graphs, etc. Parameterized algorithms may also be explored where, in addition to a parameter for the number of colours, there is also a parameter specifying the structural properties of the input graph.

References

1. Hiroki Arimura, Tatsuya Gima, Yasuaki Kobayashi, Hiroomi Nochide, and Yota Otachi. Minimum consistent subset for trees revisited. *CoRR*, abs/2305.07259, 2023. doi:10.48550/arXiv.2305.07259.
2. Sandip Banerjee, Sujoy Bhore, and Rajesh Chitnis. Algorithms and hardness results for nearest neighbor problems in bicolored point sets. In Michael A. Bender, Martín Farach-Colton, and Miguel A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics*, pages 80–93, 2018. doi:10.1007/978-3-319-77404-6_7.
3. Ahmad Biniaz, Sergio Cabello, Paz Carmi, Jean-Lou De Carufel, Anil Maheshwari, Saeed Mehrabi, and Michiel Smid. On the minimum consistent subset problem. *Algorithmica*, 83(7):2273–2302, 2021. doi:10.1007/S00453-021-00825-8.
4. Ahmad Biniaz and Parham Khamsepour. The minimum consistent spanning subset problem on trees. *Journal of Graph Algorithms and Applications*, 28(1):81–93, 2024. doi:10.7155/JGAA.V28I1.2929.

5 Rajesh Chitnis. Refined lower bounds for nearest neighbor condensation. In Sanjoy Dasgupta and Nika Haghtalab, editors, *International Conference on Algorithmic Learning Theory, 29 March - 1 April 2022, Paris, France*, volume 167 of *Proceedings of Machine Learning Research*, pages 262–281. PMLR, 2022. URL: https://proceedings.mlr.press/v167/chitnis22a.html.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

7 Sanjana Dey, Anil Maheshwari, and Subhas C. Nandy. Minimum consistent subset problem for trees. In Evripidis Bampis and Aris Pagourtzis, editors, *Fundamentals of Computation Theory - 23rd International Symposium, FCT 2021, Athens, Greece, September 12-15, 2021, Proceedings*, volume 12867 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2021. doi:10.1007/978-3-030-86593-1_14.

8 Sanjana Dey, Anil Maheshwari, and Subhas C. Nandy. Minimum consistent subset of simple graph classes. *Discret. Appl. Math.*, 338:255–277, 2023. doi:10.1016/J.DAM.2023.05.024.

9 Reinhard Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, page 7, 2012.

10 Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. doi:10.1016/0304-3975(76)90059-1.

11 Peter E. Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968. doi:10.1109/TIT.1968.1054155.

12 Kamyar Khodamoradi, Ramesh Krishnamurti, and Bodhayan Roy. Consistent subset problem with two labels. In B.S. Panda and Partha P. Goswami, editors, *Algorithms and Discrete Applied Mathematics*, pages 131–142, 2018. doi:10.1007/978-3-319-74180-2_11.

13 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 475–484, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258641.

14 Vijay V. Vazirani. *Approximation Algorithms*. Springer Publishing Company, Incorporated, 2010.

15 Gordon Wilfong. Nearest neighbor problems. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, pages 224–233, 1991. doi:10.1145/109648.109673.

Beyond Decisiveness of Infinite Markov Chains

Benoît Barbot
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Patricia Bouyer
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles,
91190 Gif-sur-Yvette, France

Serge Haddad
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles,
91190 Gif-sur-Yvette, France

Abstract

Verification of infinite-state Markov chains is still a challenge despite several fruitful numerical or statistical approaches. For *decisive* Markov chains, there is a simple numerical algorithm that frames the reachability probability as accurately as required (however with an unknown complexity). On the other hand when applicable, statistical model checking is in most of the cases very efficient. Here we study the relation between these two approaches showing first that decisiveness is a necessary and sufficient condition for almost sure termination of statistical model checking. Afterwards we develop an approach with application to both methods that substitutes to a non decisive Markov chain a decisive Markov chain with the same reachability probability. This approach combines two key ingredients: abstraction and importance sampling (a technique that was formerly used for efficiency). We develop this approach on a generic formalism called layered Markov chain (LMC). Afterwards we perform an empirical study on probabilistic pushdown automata (an instance of LMC) to understand the complexity factors of the statistical and numerical algorithms. To the best of our knowledge, this prototype is the first implementation of the deterministic algorithm for decisive Markov chains and required us to solve several qualitative and numerical issues.

2012 ACM Subject Classification Mathematics of computing → Markov processes; Theory of computation → Concurrency

Keywords and phrases Markov Chains, Infinite State Systems, Numerical and Statistical Verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.8

Related Version *Full Version*: `https://doi.org/10.48550/arXiv.2409.18670` [3]

Supplementary Material *Software (Source Code)*: `https://git.lacl.fr/barbot/Cosmos`
archived at `swh:1:dir:36e5a6f37d834251fb9a96becb481e96f5c68e6c`

Funding This work has been partly supported by ANR projects MAVeriQ (ANR-20-CE25-0012) and BisoUS (ANR-22-CE48-0012).

1 Introduction

Infinite-state discrete time Markov chains. In finite Markov chains, computing reachability probabilities can be performed in polynomial time using linear algebra techniques [14]. The case of infinite Markov chains is much more difficult, and has initiated several complementary proposals:

- A first approach consists in analyzing the high-level probabilistic model that generates the infinite Markov chains. For instance in [5], the authors study probabilistic pushdown automata and show that the reachability probability can be expressed in the first-order theory of the reals. Thus (by a dichotomous algorithm) this probability can be approximated within an arbitrary precision.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 8; pp. 8:1–8:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- A second approach consists in designing algorithms, whose correctness relies on a semantic property of the Markov chains, and which outputs an interval of the required precision that contains the reachability probability. *Decisiveness* [1] is such a property: given a target set T, it requires that almost surely, a random path either visits T or some state from which T is unreachable. In order to be effective, this algorithm needs the decidability of the (qualitative) reachability problem. For instance finite Markov chains are decisive w.r.t. any set of states. Several other classes of denumerable Markov chains are decisive by construction: Petri nets (or equivalently VASS) with constant weights[1] on transitions w.r.t. any upward-closed target set [1], lossy channel systems with constant weights and constant message loss probability [1] w.r.t. any finite target set, regular Petri nets with arbitrary weights w.r.t. any finite target set [7]. Also a critical associated decision problem is the decidability of decisiveness in the high-level model that generates the Markov chains. Decisiveness is decidable for several classes of systems: probabilistic pushdown automata with constant weights [5], random walks with polynomial weights [7] which can be generalized to probabilistic homogeneous one-counter machines with polynomial weights [7]. This class is particularly interesting since it extends the well-known model of quasi-birth death processes (QBDs).
- The statistical model checking (SMC) [17, 16] approach consists in generating numerous random paths and computing an interval of the required precision that contains the reachability probability with an arbitrary high probability (the confidence level). As we will show later on, the effectiveness of SMC also requires some semantic property, which will happen to be decisiveness.

SMC and importance sampling. When the reachability probability is very small, the SMC approach requires a huge number of random paths, which prohibits its use. In order to circumvent this problem (called the *rare event problem*), several approaches have been proposed (see for instance [13]), among which the *importance sampling* method. This seems to be in practice, one of the most efficient approaches to tackle this problem. Importance sampling consists in sampling the paths in a biased[2] Markov chain (w.r.t. the original one) that increases the reachability probability. In order to take into account the bias, a likelihood for any path is computed (on-the-fly) and the importance sampling algorithm returns the empirical average value of the likelihood. While the expected returned value is equal to the reachability probability under evaluation, the confidence interval returned by the algorithm is, without further assumption, only “indicative” (i.e., it does not necessary fulfill the features of a confidence interval) – boundedness of the likelihood is indeed required (but hard to ensure). In [4], a simple relation between the biased and the original finite Markov chain is stated that (1) ensures that the confidence interval returned by the algorithm is a “true” interval and that (2) the variance of the estimator (here the likelihood) is reduced w.r.t. the original estimator, entailing an increased efficiency of the SMC.

Related work when the Markov chain is not decisive. Very few works have addressed the effectiveness of SMC for infinite non decisive Markov chains. The main proposal [15] consists in stopping the computation at each step with some fixed (small) probability. The successful paths are equipped with a numerical value, whose average over the paths is returned by the

[1] That is, each transition is assigned a weight, and the probability for a transition to be fired is its relative weight w.r.t. all enabled transitions.

[2] In the sense that probability values in the biased chain differ from the original chain.

algorithm. It turns out that it is an importance sampling method, which has surprisingly not been pointed out by the authors. However here again, since the likelihood is (in general) not bounded, the interval returned by the algorithm is not a confidence interval. An alternative notion called divergence has been proposed in [8] to partly cover the case of non-decisive Markov chains.

Our contributions. We introduce the *reward reachability problem* (a slight generalization of the reachability problem) by associating a reward with every successful path and looking for the expected reward. We first establish that decisiveness is a necessary and sufficient condition for the almost-sure termination of SMC for bounded rewards.

- Our major contribution consists in establishing a relation between a non-decisive Markov chain and an auxiliary Markov chain, called an *abstraction*, with the following property: they can be combined into a biased Markov chain, which happens to be **decisive**; the SMC with importance sampling on this chain provides a confidence interval for the reachability probability of the original Markov chain.
- We furthermore show that importance sampling can be applied to adapt (based on the abstraction) the deterministic algorithm of [1].
- Afterwards we illustrate the interest of this approach, by exhibiting a generic model called *layered Markov chains* (LMC), which can be instantiated for instance by probabilistic pushdown automata with polynomial weights. These automata cannot be handled with the technique of [5].
- Finally we present several experiments, based on the tool Cosmos [2], which compare the SMC and the deterministic approaches. It allows to identify how various factors impact the efficiency of the algorithms. We provide within the tool Cosmos the first implementation of the deterministic approach for decisive Markov chains, which required us to solve several numerical issues. As a rough summary, at the price of a confidence level against certainty, the computing time of SMC is generally several magnitude orders smaller than the one of the deterministic algorithm.

Organization. In section 2, we introduce the numerical and statistical specification of the reward reachability problem, and we recall the notion of decisiveness. In section 3, we focus on the decisiveness property establishing that decisiveness is a necessary and sufficient condition for almost sure termination of statistical model checking. Section 4 contains our main contribution: the specification of an abstraction of a Markov chain, its use for solving the reward reachability problems for non decisive Markov chains via importance sampling and the development of this method for LMCs. Afterwards in Section 5, we present some implementation details and experimentally compare the deterministic and the statistical approaches. We conclude and give some perspectives to this work in Section 6.

Some missing proofs and more details on the implementation can be found in the appendix. Full proofs are given in [3].

2 Preliminaries

In this preliminary section we define Markov chains, and the decisiveness property.

▶ **Definition 1.** *A* discrete-time Markov chain *(or simply* Markov chain*)* $\mathcal{C} = (S, P)$ *is defined by a countable set of states* S *and a transition probability matrix* P *of size* $S \times S$*. Given an initial state* $s_0 \in S$*, the state of the chain at time* n *is a random variable (r.v. in short)* $X_n^{\mathcal{C},s_0}$ *defined by:* $\mathbf{Pr}\big(X_0^{\mathcal{C},s_0} = s_0\big) = 1$ *and* $\mathbf{Pr}\big(X_{n+1}^{\mathcal{C},s_0} = s' \mid \bigwedge_{i\leq n} X_i^{\mathcal{C},s_0} = s_i\big) = P(s_n, s')$*.*

If $\mathcal{C} = (S, \mathrm{P})$ is a Markov chain, we write $E_\mathcal{C} = \{(s, s') \in S \times S \mid \mathrm{P}(s, s') > 0\}$ for the set of edges of $\mathcal{C}$, and $\to_\mathcal{C}$ for the corresponding edge relation. A state s is *absorbing* if $\mathrm{P}(s, s) = 1$. A Markov chain $\mathcal{C}$ is said *effective* whenever for every $s \in S$, the support of $\mathrm{P}(s, \cdot)$ is finite and computable, and for every $s, s' \in S$, $\mathrm{P}(s, s')$ is computable. A target set $T \subseteq S$ is said *effective* whenever its membership problem is decidable. In the following, we will always consider effective Markov chains and effective target sets when speaking of algorithms, without always specifying it.

A finite (resp. infinite) path is a finite (resp. infinite) sequence of states $\rho = s_0 s_1 s_2 \ldots \in S^+$ (resp. S^ω) such that for every $0 \leq i$, $(s_i, s_{i+1}) \in E_\mathcal{C}$. We write *first*$(\rho)$ for s_0, and whenever $\rho \in S^+$, we write *last*(ρ) for the last state of ρ. For every $n \in \mathbb{N}$, we write $\rho[n] \overset{\text{def}}{=} s_n$ and $\rho_{\leq n} \overset{\text{def}}{=} s_0 s_1 s_2 \ldots s_n$. If $\rho = s_0 \ldots s_n$, $\mathbf{Pr}(\rho)$ is equal to $\prod_{i<n} \mathrm{P}(s_i, s_{i+1})$ and corresponds to the probability that this path is followed when starting from its initial state s_0.

The random infinite path generated by process $(X_n^{\mathcal{C},s_0})_{n\in\mathbb{N}}$ will be denoted $\varrho^{\mathcal{C},s_0}$. Note that $s \to_\mathcal{C}^* s'$ if and only if $\mathbf{Pr}\big(\varrho^{\mathcal{C},s} \models \Diamond\{s'\}\big) > 0$ (we use the $\Diamond$ modality of temporal logics, which expresses *Eventually*, and later, we will also write $\Diamond_{>0}$ for the *strict Eventually* modality – eventually but not now –, as well as $\Diamond_{\leq n}$ for *n-steps Eventually*). Finally, for every $s, s' \in S$, we define the time from s to s' as the random variable $\tau^{\mathcal{C},s,s'} = \min\{i \in \mathbb{N} \mid i > 0 \text{ and } X_i^{\mathcal{C},s} = s'\}$, with values in $\mathbb{N}_{>0} \cup \{+\infty\}$. To ease the reading, we will omit subscripts $_\mathcal{C}$ and $_T$, or superscripts $^\mathcal{C}$ in the various notations, whenever it is obvious in the context.

In this paper we are interested in evaluating the probability to reach a designed target set T from an initial state s_0 in a Markov chain $\mathcal{C}$, that is, $\mu_{\mathcal{C},T}(s_0) \overset{\text{def}}{=} \mathbf{Pr}\big(\varrho^{\mathcal{C},s_0} \models \Diamond T\big)$. In general, it might be difficult to compute such a value, which will often not even be a rational number. That is why like many other research works we will show how to compute accurate approximations (surely or with a high level of confidence). We present our solutions in a more general setting which would anyway be necessary in the following developments.

▶ **Definition 2.** *Let $T \subseteq S$ and $\rho \in S^\omega$. We let first$_T(\rho) := \min\{i \in \mathbb{N} \mid \rho[i] \in T\} \in \mathbb{N} \cup \{\infty\}$. Let $L : S^+ \to \mathbb{R}$ be a function. The function $f_{L,T} : S^\omega \to \mathbb{R}$ is then defined by:*[3]

$$f_{L,T}(\rho) := \begin{cases} L\big(\rho_{\leq first_T(\rho)}\big) & \text{if } first_T(\rho) \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases}$$

We say that $f_{L,T}(\rho)$ is the *reward* of ρ. The function $f_{L,T}$ is called the *T-function for L*; let $B \in \mathbb{R}_{>0}$, $f_{L,T}$ is said *B-bounded* whenever $\max(|f_{L,T}(\rho)| \mid \rho \in S^\omega) \leq B$. Observe that $f_{L,T}$ could be B-bounded for some B even if L is unbounded.

We will be interested in evaluating the expected reward $\nu_{\mathcal{C},L,T}(s_0) \overset{\text{def}}{=} \mathbf{E}\big(f_{L,T}(\varrho^{\mathcal{C},s_0})\big)$.[4] Note that if L is constant equal to 1, then $f_{L,T} = \mathbb{1}_{\Diamond T}$ is the indicator function for paths that visit T, in which case $\nu_{\mathcal{C},L,T}(s_0) = \mu_{\mathcal{C},T}(s_0)$.

We define two problems related to the accurate estimation of these values:

- The EvalER problem (EvalER stands for "Evaluation of the Expected Reward") asks for *a deterministic algorithm*, which:
  1. takes as input a Markov chain $\mathcal{C}$, an initial state s_0, a computable function $L : S^+ \to \mathbb{R}_{\geq 0}$, a target set T, a precision $\varepsilon > 0$, and
  2. outputs an interval $I \subseteq \mathbb{R}$ of length bounded by ε such that $\nu_{\mathcal{C},L,T}(s_0) \in I$.

  The particular case of the reachability probability (when L is constant equal to 1) is denoted EvalRP.

[3] This function is measurable as a pointwise limit of measurable functions.
[4] $\mathbf{E}$ denotes the expectation.

- The EstimER problem (EstimER stands for "Estimation of the Expected Reward") asks for a *probabilistic Las Vegas algorithm*, which:
  1. takes as input a Markov chain $\mathcal{C}$, an initial state s_0, a computable function $L : S^+ \to \mathbb{R}_{\geq 0}$, a target set T, a precision $\varepsilon > 0$, a confidence value $\delta > 0$, and
  2. outputs a random interval $I \subseteq \mathbb{R}$ of length bounded by ε such that $\mathbf{Pr}\big(\nu_{\mathcal{C},L,T}(s_0) \notin I\big) \leq \delta$, and $\mathbf{E}(\mathsf{mid}(I)) = \nu_{\mathcal{C},L,T}(s_0)$, where $\mathsf{mid}(I)$ is the middle of interval I.[5]

  The particular case of the reachability probability (when L is constant equal to 1) is denoted EstimRP.

In [1], the concept of decisiveness for Markov chains was introduced. Roughly, decisiveness allows to lift some "good" properties of finite Markov chains to countable Markov chains. We recall this concept here. Let $T \subseteq S$ and denote the "avoid set" of T by $\mathsf{Av}_{\mathcal{C}}(T) \overset{\text{def}}{=} \{s \in S \mid \mathbf{Pr}(\varrho^{\mathcal{C},s} \models \Diamond T) = 0\}$.

▶ **Definition 3.** *The Markov chain $\mathcal{C}$ is* decisive w.r.t. T from s_0 *if* $\mathbf{Pr}\big(\varrho^{\mathcal{C},s_0} \models \Diamond T \vee \Diamond \mathsf{Av}_{\mathcal{C}}(T)\big) = 1$.

3 Analysis of decisive Markov chains

We fix for this section a Markov chain $\mathcal{C} = (S, \mathrm{P})$, an initial state s_0, a computable function $L : S^+ \to \mathbb{R}_{\geq 0}$ and a target set T, and we assume w.l.o.g. that T is a set of absorbing states. We present two approaches (extended from the original ones) to compute the expected value of the function $f_{L,T}$ that require $\mathcal{C}$ to be decisive w.r.t. T from s_0.

3.1 Decisiveness and approximation algorithm

In the original paper proposing the concept of decisiveness [1], "theoretical" approximation schemes were designed. We slightly extend the one designed for reachability objectives in our more general setting, see Algorithm 1.

Algorithm 1 Approximation scheme for the EvalER problem; the fair_extract operation ensures that any element put in the set cannot stay forever in an execution including an infinite number of extractions; a simple implementation can be done with a queue.

```
input : $\mathcal{C} = (S, \mathrm{P})$ a countable Markov chain, $s_0 \in S$ an initial state, $L : S^+ \to \mathbb{R}_{\geq 0}$
        a computable function, $T \subseteq S$ a target set s.t. $\mathsf{Av}_{\mathcal{C}}(T)$ is effective and $f_{L,T}$
        is $B$-bounded, $\varepsilon > 0$ a precision.
$e := 0$, $p_{\text{fail}} := 0$, $p_{\text{succ}} := 0$; $set := \{(1, s_0)\}$;
while $1 - (p_{succ} + p_{fail}) > \varepsilon/2B$ do
    $(p, \rho) :=$ fair_extract$(set)$; $s := last(\rho)$;
    if $s \in T$ then $e := e + p \cdot L(\rho)$; $p_{\text{succ}} := p_{\text{succ}} + p$;
    else if $s \in \mathsf{Av}(T)$ then $p_{\text{fail}} := p_{\text{fail}} + p$;
    else
        for $s \to_{\mathcal{C}} s'$ do insert$(set, (p \cdot \mathrm{P}(s, s'), \rho s'))$;
end
return $[e - \varepsilon/2, e + \varepsilon/2]$
```

[5] The last condition on the middle of I means that the estimator is unbiased.

The termination and correctness of this algorithm is established by the following proposition, whose proof is given in [3] and a special case of which is given in [1].

▶ **Proposition 4** (Termination and correctness of Algorithm 1). *Algorithm 1 solves the* EvalER *problem if and only if $\mathcal{C}$ is decisive w.r.t. T from s_0.*

Up to our knowledge, the version of this algorithm for computing reachability probabilities has not been implemented, hence the terminology "theoretical" scheme above. Also, there is no known convergence speed. Later in section 5, we briefly describe an efficient implementation of this scheme by designing some tricks.

3.2 Decisiveness and (standard) statistical model-checking

The standard *statistical model-checking* (SMC in short) consists in sampling a large number of paths to simulate the random variables $X^{s_0} = (X^{s_0}_n)_{n\geq 0}$; a sampling is stopped when it hits T or $\mathsf{Av}(T)$, and a value 1 (resp. 0) is assigned when T (resp. $\mathsf{Av}(T)$) is hit; finally the average of all the values is computed. This requires that almost-surely a path hits T or $\mathsf{Av}(T)$, which is precisely decisiveness of the Markov chain w.r.t. T from s_0. This allows to compute an *estimate* of the probability to reach T. We describe more precisely the approach and extend the context to allow the estimation of the expected value of $f_{L,T}$.

Algorithm 2 Statistical model-checking for the EstimER problem.

```
input :$\mathcal{C} = (S, \mathrm{P})$ a countable Markov chain, $s_0 \in S$ an initial state, $L : S^+ \to \mathbb{R}$ a
       computable function, $T \subseteq S$ a target set s.t. $\mathsf{Av}_{\mathcal{C}}(T)$ is effective and $f_{L,T}$ is
       $B$-bounded, $\varepsilon > 0$ a precision, $\delta > 0$ a confidence value.
$N := \left\lceil \frac{8B^2}{\varepsilon^2} \log\left(\frac{2}{\delta}\right) \right\rceil$; $\hat{f} := 0$;
for $i$ from 1 to $N$ do
    $\rho := s_0$; $s := s_0$;
    while $s \notin T \cup \mathsf{Av}(T)$ do  $s' :=$ sample($\mathrm{P}(s, \cdot)$); $\rho := \rho s'$; $s := s'$ ;
    if $s \in T$ then $\hat{f} := \hat{f} + L(\rho)$;
end
$\hat{f} := \frac{\hat{f}}{N}$; return $[\hat{f} - \varepsilon/2, \hat{f} + \varepsilon/2]$
```

The SMC approach is presented as Algorithm 2 (where $\mathsf{Av}(T)$ is assumed to be effective). This is in general a semi-algorithm, since it may happen that the **while** loop is never left at some iteration i. Nevertheless, decisiveness ensures almost sure (a.s.) termination:

▶ **Lemma 5.** *The **while** loop a.s. terminates if and only if $\mathcal{C}$ is decisive w.r.t. T from s_0.*

The correctness of the algorithm will rely on this proposition that can be straightforwardly deduced from the Hoeffding inequality [10].

▶ **Proposition 6.** *Let $V_1, \ldots, V_N$ be B-bounded independent random variables and let $V = \frac{1}{N}\sum_{i=1}^{N} V_i$. Let $\varepsilon, \delta > 0$ be such that $N \geq \frac{8B^2}{\varepsilon^2}\log\left(\frac{2}{\delta}\right)$. Then:* $\mathbf{Pr}\left(\left|V - \mathbf{E}(V)\right| \geq \frac{\varepsilon}{2}\right) \leq \delta$.

We can now state the following important result.

▶ **Proposition 7** (Termination and correctness of Algorithm 2). *Algorithm 2 solves the* EstimER *problem if and only if $\mathcal{C}$ is decisive w.r.t. T from s_0.*

Proof. The termination is a consequence of Lemma 5. The correctness is a consequence of Proposition 6, by taking random variable V_i as $f_{L,T}\big(\varrho^{\mathcal{C},s_0}\big)$. In this case, V is equal to the value of $\hat{f}$ at the end of the algorithm, which completes the argument. ◀

While termination is guaranteed by the previous corollary the (time) efficiency of the simulation remains a critical factor. In particular, the expected value D of the random time $\tau^{s_0,T\cup\mathsf{Av}(T)}$ to reach $T \cup \mathsf{Av}(T)$ from s_0 should be finite; in this case, the average simulation time will be D and therefore the complexity of the whole approach will be linear in the number of simulations. Decisiveness does not ensure this; so a dedicated analysis needs to be done to ensure efficiency of the approach.

4 Beyond decisiveness

In the previous section, we have presented two generic approaches for analyzing infinite (denumerable) Markov chains. They both only apply to **decisive** Markov chains. In this section, we twist the previous approaches, so that they will be applicable to analyze some **non decisive** Markov chains as well. Our proposition follows the following steps:

- based on the *importance sampling* approach, we explain how the analysis of the original Markov chain can be transferred to that of a *biased* Markov chain (Subsection 4.1);
- we explain how a biased Markov chain can be automatically constructed via an *abstraction*, and give conditions ensuring that the obtained biased Markov chain can be analyzed (Subsection 4.2);
- we give a generic framework based on *layered Markov chains* and *random walks*, with conditions on various parameters to safely apply the designed approach (Subsection 4.3).

We fix an effective countable Markov chain $\mathcal{C} = (S, \mathrm{P})$, $s_0 \in S$ an initial state, $L : S^+ \to \mathbb{R}_{\geq 0}$ a computable function, and $T \subseteq F \subseteq S$ two effective sets, with both $\mathsf{Av}_{\mathcal{C}}(F)$ and $\mathsf{Av}_{\mathcal{C}}(T)$ being effective (note that $\mathsf{Av}_{\mathcal{C}}(F) \subseteq \mathsf{Av}_{\mathcal{C}}(T)$). Since we are interested in the probability to reach T, from now on, we assume that T is absorbing in $\mathcal{C}$ and that $s_0 \notin \mathsf{Av}_{\mathcal{C}}(F)$.

4.1 Model-checking via a biased Markov chain

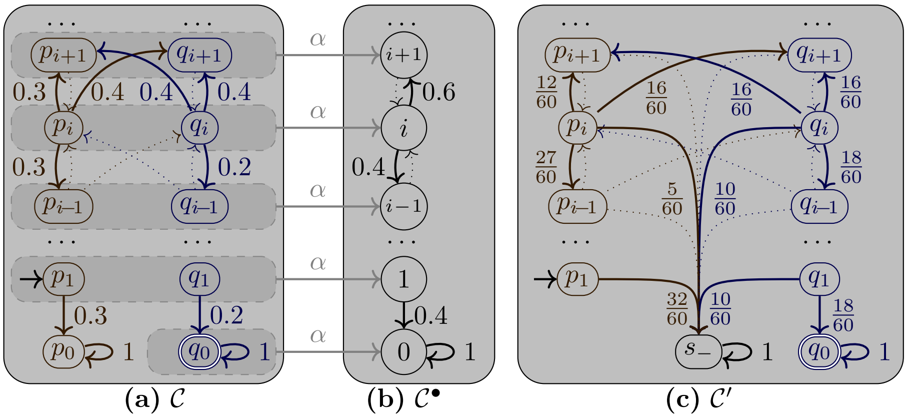
Importance sampling has been introduced in the fifties [12] to evaluate rare-event probabilities (see the book [13] for more details). We revisit the approach in our more general setting of reward reachability, with the extra set F.[6] The role of F will be discussed page 10. This approach applies the standard SMC approach with a correction factor, called *likelihood*, to another Markov chain.

▶ **Definition 8** (biased Markov chain and likelihood). *Let $\mathcal{C} = (S, P)$ with $T \subseteq F \subseteq S$ and $\mathcal{C}' = (S', P')$ be Markov chains such that:*

- *$S' = (S \setminus \mathsf{Av}_{\mathcal{C}}(F)) \uplus \{s_-\}$, where $s_- \notin S$;*
- *In $\mathcal{C}'$ all states of $T \cup \{s_-\}$ are absorbing;*
- $\forall s, s' \in S \setminus \mathsf{Av}_{\mathcal{C}}(F),\ P(s, s') > 0 \ \Rightarrow\ P'(s, s') > 0$ (1)

Then $\mathcal{C}'$ is a biased Markov chain *of $(\mathcal{C}, T, F)$ and the* likelihood *$\gamma_{\mathcal{C},\mathcal{C}'}$ is the non negative function defined for finite paths $\rho \in S'^+$ s.t. $\mathbf{Pr}'(\rho) > 0$ by: $\gamma_{\mathcal{C},\mathcal{C}'}(\rho) \overset{def}{=} \frac{\mathbf{Pr}(\rho)}{\mathbf{Pr}'(\rho)}$ if ρ does not visit s_-, and $\gamma_{\mathcal{C},\mathcal{C}'}(\rho) \overset{def}{=} 0$ otherwise.*

[6] The standard importance sampling method is recovered when $F = T$.



(a) $\mathcal{C}$ **(b)** $\mathcal{C}^\bullet$ **(c)** $\mathcal{C}'$

Figure 1 $\mathcal{C}$,$\mathcal{C}^\bullet$,$\mathcal{C}'$ are three Markov chains and α is defined by $\alpha(q_0) = 0$ and for all $n > 0$, $\alpha(p_n) = \alpha(q_n) = n$. $\mathcal{C}'$ is a biased Markov chain of $(\mathcal{C}, \{q_0\}) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet, \{0\})$.

Eqn. (1) ensures that this modification cannot remove transitions between states of $S \setminus \mathsf{Av}_{\mathcal{C}}(F)$, but it can add transitions. So, $\mathsf{Av}_{\mathcal{C}'}(F) = \{s_-\}$. We fix a biased Markov chain $\mathcal{C}'$ for the rest of this subsection and omit the subscripts for the likelihood function γ. The likelihood can be computed greedily from the initial state: if $\rho \cdot s$ is a finite path of $\mathcal{C}'$ such that $\gamma(\rho)$ has been computed, then $\gamma(\rho \cdot s)$ is equal to 0 if $s = s_-$, and $\gamma(\rho) \cdot \frac{\mathrm{P}\left(last(\rho), s\right)}{\mathrm{P}'\left(last(\rho), s\right)}$ otherwise.

▶ **Example 9.** Figure 1(c) depicts a Markov chain which is a biased Markov chain of Figure 1(a) with $T = F = \{q_0\}$ and $\mathsf{Av}_{\mathcal{C}}(F) = \mathsf{Av}_{\mathcal{C}}(T) = \{p_0\}$.

Using the likelihood, we can define the new function of interest in Markov chain $\mathcal{C}'$. We let $L' \overset{\text{def}}{=} L \cdot \gamma$ and we realize that the expected reward of $f_{L',T}$ in $\mathcal{C}'$ from s_0 coincides with the expected reward of $f_{L,T}$ in $\mathcal{C}$ from s_0, as stated below.

▶ **Proposition 10.** $\mathbf{E}\big(f_{L',T}\big(\varrho^{\mathcal{C}',s_0}\big)\big) = \mathbf{E}\big(f_{L,T}\big(\varrho^{\mathcal{C},s_0}\big)\big)$.

The proof of this proposition is given in Appendix A.1. The idea is that the likelihood in $\mathcal{C}'$ compensates for the bias in the probabilities in $\mathcal{C}'$ w.r.t. original probabilities in $\mathcal{C}$. Thanks to this result, the computation of the expected value of $f_{L,T}$ in $\mathcal{C}$ can be reduced to the computation of the expected value of $f_{L',T}$ in $\mathcal{C}'$. Thus, as soon as $\mathcal{C}'$ and $f_{L',T}$ satisfy the hypotheses of Proposition 4 (resp. Proposition 7) for the EvalER (resp. EstimER) problem, Algorithm 1 (resp. Algorithm 2) can be applied to $\mathcal{C}'$, which will solve the corresponding problem in $\mathcal{C}$. Specifically, the second method is what is called the *importance sampling* of $\mathcal{C}$ via $\mathcal{C}'$. Observe the following facts:

- the decisiveness hypothesis only applies to the biased Markov chain $\mathcal{C}'$, not to the original Markov chain $\mathcal{C}$;
- the requirement that $f_{L',T}$ be B-bounded (for some B) does not follow from any hypothesis on $f_{L,T}$ since the likelihood might be unbounded.

4.2 Construction of a biased Markov chain via an abstraction

The approach designed in the previous subsection requires the decisiveness of the biased Markov chain and the effective boundedness of the function which is evaluated. We now deport these various assumptions on another Markov chain, for which numerical (or symbolical) computations can be done, and which will serve as an *abstraction*. This approach generalizes [4] in several directions: first, [4] was designed for finite Markov chains; then, we consider a superset F of T which will allow us to relax conditions over $S \setminus T$ to its subset $S \setminus F$.

▶ **Definition 11.** *A Markov chain* $\mathcal{C}^\bullet = (S^\bullet, P^\bullet)$ *together with a set* $F^\bullet$ *is called an* abstraction *of* $\mathcal{C}$ *with set* F *by function* $\alpha\colon S \setminus \mathsf{Av}_{\mathcal{C}}(F) \to S^\bullet$ *whenever, the following conditions hold:*

(A) *for all* $s \in F$, $\alpha(s) \in F^\bullet$*;*

(B) *for all* $s \in S \setminus (F \cup \mathsf{Av}_{\mathcal{C}}(F))$, $\displaystyle\sum_{s' \notin \mathsf{Av}_{\mathcal{C}}(F)} \mathrm{P}(s,s') \cdot \mu_{\mathcal{C}^\bullet,F^\bullet}(\alpha(s')) \leq \mu_{\mathcal{C}^\bullet,F^\bullet}(\alpha(s))$.

Condition (B) is called *monotony* and is only required outside $F \cup \mathsf{Av}_{\mathcal{C}}(F)$. We write more succinctly that $(\mathcal{C}^\bullet, F^\bullet)$ is an α-abstraction of $(\mathcal{C}, F)$, denoted $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet, F^\bullet)$ and $\mu_{F^\bullet} \overset{\text{def}}{=} \mu_{\mathcal{C}^\bullet,F^\bullet}$ and $\mu_F \overset{\text{def}}{=} \mu_{\mathcal{C},F}$.

▶ **Example 12.** We claim that the Markov chain $\mathcal{C}^\bullet$ in Figure 1(b) with $F^\bullet = \{0\}$ is an abstraction of $\mathcal{C}$ in Figure 1(a) with $s_0 = p_1$. Indeed, the monotony condition is satisfied: for all $n > 0$:

- in p_n : $0.3\left(\frac{2}{3}\right)^{n+1} + 0.4\left(\frac{2}{3}\right)^{n+1} + 0.3\left(\frac{2}{3}\right)^{n-1} = \frac{55}{60}\left(\frac{2}{3}\right)^{n} < \left(\frac{2}{3}\right)^{n}$;
- in q_n : $0.4\left(\frac{2}{3}\right)^{n+1} + 0.4\left(\frac{2}{3}\right)^{n+1} + 0.2\left(\frac{2}{3}\right)^{n-1} = \frac{25}{30}\left(\frac{2}{3}\right)^{n} < \left(\frac{2}{3}\right)^{n}$.

Observe that $\mu_{F^\bullet}(n) = \left(\frac{0.4}{0.6}\right)^n = \left(\frac{2}{3}\right)^n$.

As will be explicit in the next lemma, an abstraction is a stochastic bound of the initial Markov chain outside $\mathsf{Av}_{\mathcal{C}}(F)$.

▶ **Lemma 13.** *Let* $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet, F^\bullet)$*. Then for all* $s \in S \setminus \mathsf{Av}_{\mathcal{C}}(F)$, $\mu_F(s) \leq \mu_{F^\bullet}(\alpha(s))$*. In particular, for all* $s \in S \setminus \mathsf{Av}_{\mathcal{C}}(F)$, $\mu_{F^\bullet}(\alpha(s)) > 0$.

Proof. Let $\mu_F^{(n)}(s) \overset{\text{def}}{=} \mathbf{Pr}(s \models \Diamond_{\leq n} F)$. Observe that $\mu_F(s) = \lim_{n\to+\infty} \mu_F^{(n)}(s)$. We show by induction on n that for all $s \in S$ and all $n \in \mathbb{N}$, $\mu_F^{(n)}(s) \leq \mu_{F^\bullet}(\alpha(s))$.

- Case $n = 0$:
  - $s \in F$ implies $\alpha(s) \in F^\bullet$ (condition (A)). Hence $\mu_{F^\bullet}(\alpha(s)) = 1 = \mu_F^{(0)}(s)$.
  - $s \in S \setminus F$: $\mu_F^{(0)}(s) = 0 \leq \mu_{F^\bullet}(\alpha(s))$.
- Inductive case:
  - $s \in F$ implies $\alpha(s) \in F^\bullet$ (condition (A)). Hence $\mu_{F^\bullet}(\alpha(s)) = 1 = \mu_F^{(n+1)}(s)$.
  - $s \in S \setminus F$: $\mu_F^{(n+1)}(s) = \sum_{s'} \mathrm{P}(s,s') \cdot \mu_F^{(n)}(s') = \sum_{s' \notin \mathsf{Av}_{\mathcal{C}}(F)} \mathrm{P}(s,s') \cdot \mu_F^{(n)}(s') \leq \sum_{s' \notin \mathsf{Av}_{\mathcal{C}}(F)} \mathrm{P}(s,s') \cdot \mu_{F^\bullet}(\alpha(s'))$ by induction hypothesis. Hence $\mu_F^{(n+1)}(s) \leq \mu_{F^\bullet}(\alpha(s))$ by condition (B). ◀

Given an abstraction $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet, F^\bullet)$ and $s \in S \setminus \mathsf{Av}_{\mathcal{C}}(F)$, let $h(s)$ be the *decreasing ratio* at s: $h(s) \overset{\text{def}}{=} \dfrac{1}{\mu_{F^\bullet}(\alpha(s))} \cdot \displaystyle\sum_{s' \in S \setminus \mathsf{Av}_{\mathcal{C}}(F)} \mathrm{P}(s,s') \cdot \mu_{F^\bullet}(\alpha(s'))$. For all $s \in S$, $h(s) \leq 1$: this is obvious when $s \in S \setminus F \cup \mathsf{Av}_{\mathcal{C}}(F)$ by the monotony condition (B); if $s \in F$, then $\alpha(s) \in F^\bullet$ by condition (A), and hence $\mu_{F^\bullet}(\alpha(s)) = 1$.

We now define a biased Markov chain based on the above abstraction, which will be interesting for both methods (approximation and estimation).

▶ **Definition 14.** *Let $(\mathcal{C},F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet,F^\bullet)$. Then $\mathcal{C}' = \big((S \setminus \mathsf{Av}_{\mathcal{C}}(F)) \uplus \{s_-\}, P'\big)$ is the Markov chain, where s_- is absorbing and for all $s,s' \in S \setminus \mathsf{Av}_{\mathcal{C}}(F)$, $P'(s,s') = P(s,s') \cdot \dfrac{\mu_{F^\bullet}(\alpha(s'))}{\mu_{F^\bullet}(\alpha(s))}$ and $P'(s,s_-) = 1 - h(s)$.*

By assumption, for all $s \in T$, s is absorbing in $\mathcal{C}$. This implies in particular that s is also absorbing in $\mathcal{C}'$. Also, notice that P' coincides with P within F, which means that there is no bias in the zone F in $\mathcal{C}'$ w.r.t. $\mathcal{C}$.

▶ **Lemma 15.** *Let $(\mathcal{C},F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet,F^\bullet)$. Then the Markov chain $\mathcal{C}'$ defined in Definition 14 is a biased Markov chain of $(\mathcal{C},T,F)$.*

Proof. First probabilities are well-defined, thanks to the remark on h being bounded by 1. The only thing which needs to be checked is the following: if $s,s' \notin \mathsf{Av}_{\mathcal{C}}(F)$ and $\mathrm{P}(s,s') > 0$, then $\mathrm{P}'(s,s') > 0$. Since $\mathrm{P}'(s,s') = \mathrm{P}(s,s') \cdot \frac{\mu_{F^\bullet}(\alpha(s'))}{\mu_{F^\bullet}(\alpha(s))}$ and $s' \notin \mathsf{Av}_{\mathcal{C}}(F)$ using Lemma 13, $\mu_{F^\bullet}(\alpha(s')) \geq \mu_F(s') > 0$. So $\mathcal{C}'$ is a biased Markov chain of $(\mathcal{C},T,F)$. ◀

Since the only transitions added to $\mathcal{C}$, when defining $\mathcal{C}'$, lead to s_-, the (qualitative) reachability of T is unchanged and so $\mathsf{Av}_{\mathcal{C}'}(T) = (\mathsf{Av}_{\mathcal{C}}(T) \setminus \mathsf{Av}_{\mathcal{C}}(F)) \cup \{s_-\}$. Furthermore $\mathcal{C}'$ does not depend on T. So we call $\mathcal{C}'$ the *biased Markov chain of* $(\mathcal{C},F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet,F^\bullet)$. As above, we define the likelihood γ, and accordingly the function $L' = L \cdot \gamma$. So the approach of Subsection 4.1 can be applied, provided $\mathcal{C}'$ satisfies the required properties (decisiveness and boundedness of the evaluated function). In subsection 4.3, we will be more specific and give a generic framework guaranteeing those properties.

Role of F. In the original importance sampling method, there was no superset $F \supseteq T$, and the monotony condition was imposed on $S \setminus T$. However, in practice, the monotony condition may not be satisfied in $F \setminus T$ while being satisfied in $S \setminus F$; hence the formulation with a superset $F \supseteq T$ widens the applicability of the approach. It should be noted that once a set F has been found, which ensures the monotony condition, any of its supersets will also do the work. Its choice will impact the efficiency of the approach, as will be illustrated in Section 5, and will therefore serve as a parameter of the approach that can be adjusted for improving efficiency.

We end up this subsection with some property of the reward function that is to be analyzed in the biased Markov chain obtained using an abstraction.

▶ **Proposition 16.** *Let $(\mathcal{C},F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet,F^\bullet)$ and L a computable function from S^+ to $\mathbb{R}$ such that $f_{L,T}$ is B-bounded. Let $\mathcal{C}'$ be the biased Markov chain of $(\mathcal{C},F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet,F^\bullet)$ and $L' = L \cdot \gamma_{\mathcal{C},\mathcal{C}'}$. Let $s_0 \in S$, then for every infinite path ρ in $\mathcal{C}'$ starting at s_0:*

$$f_{L',T}(\rho) = \begin{cases} L(\rho_{\leq \mathit{first}_T(\rho)}) \cdot \mu_{F^\bullet}(\alpha(s_0)) & \text{if } \rho \models \Diamond T \\ 0 & \text{otherwise} \end{cases}$$

Thus $f_{L',T}$ is B-bounded.

The proof of this proposition is given in Appendix A.2. Thus in addition to be a biased Markov chain of $\mathcal{C}$, $\mathcal{C}'$ preserves a necessary condition for applying algorithms of Section 3: the boundedness of the reward function. Furthermore, when $f_{L,T} = \mathbb{1}_{\Diamond T}$ (corresponding to the standard reachability property), $f_{L',T}$ for paths starting at s_0 is a bivaluated function: $f_{L',T} = \mu_{F^\bullet}(\alpha(s_0)) \cdot \mathbb{1}_{\Diamond T}$ which does not need to be computed on the fly by the algorithms.

4.3 A generic framework based on random walks

Our objective is to apply the algorithms of Section 3 to the biased Markov chain $\mathcal{C}'$ defined in the previous subsection via an abstraction, and to exploit Proposition 16. This requires $\mathcal{C}'$ to be effective and to be decisive w.r.t. T. The effectiveness will be obtained via the numerical or symbolic computation (since $\mathcal{C}^\bullet$ is infinite) of $\mu_{F^\bullet}(\alpha(s))$. To that purpose, we use random walks as abstractions since they have closed forms for the reachability probabilities and *layered Markov chains* as generic models. The proofs of this section are either omitted or sketched and full proofs can be found in Appendix. The proofs of this section are only partly given in the core and in the appendix of this paper, but are fully given in [3].

▶ **Definition 17.** *A* layered Markov chain *(LMC in short) is a tuple $(\mathcal{C}, \lambda)$ where $\mathcal{C} = (S, P)$ is a countable Markov chain, $\lambda : S \to \mathbb{N}$ is a mapping such that for all $s \to_{\mathcal{C}} s'$, $\lambda(s) - \lambda(s') \leq 1$, and for all $n \in \mathbb{N}$, $\lambda^{-1}(n)$ is finite.*

Given $s \in S$, $\lambda(s)$ is the *level* of s. In words there are two requirements on λ: (1) after one step the level can be decreased by at most one unit while it can be arbitrarily increased, and (2) for any level ℓ, the set of states with level ℓ is finite. We define $\mathrm{P}^+(s)$, $\mathrm{P}^-(s)$ and $\mathrm{P}^=(s)$ (with $\mathrm{P}^+(s) + \mathrm{P}^-(s) + \mathrm{P}^=(s) = 1$) as follows:

$$\mathrm{P}^+(s) = \sum_{\substack{s' \in S \text{ s.t.}\\ \lambda(s') \geq \lambda(s)+1}} \mathrm{P}(s, s'), \quad \mathrm{P}^-(s) = \sum_{\substack{s' \in S \text{ s.t.}\\ \lambda(s') = \lambda(s)-1}} \mathrm{P}(s, s'), \quad \mathrm{P}^=(s) = \sum_{\substack{s' \in S \text{ s.t.}\\ \lambda(s') = \lambda(s)}} \mathrm{P}(s, s')$$

In the sequel we fix an LMC $(\mathcal{C}, \lambda)$ and we consider a finite target set T. We want to apply the previous approach to $\mathcal{C}$ using an α-abstraction $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet, F^\bullet)$, where $\mathcal{C}^\bullet$ is the random walk $\mathcal{W}^p = (\mathbb{N}, \mathrm{P}_p)$ with some probability parameter $0 < p < 1$ defined as follows: $\mathrm{P}_p(0, 0) = 1$; for every $i > 0$, $\mathrm{P}_p(i, i+1) = p$ and $\mathrm{P}_p(i, i-1) = 1 - p$ (it is depicted in Figure 1(b) for $p = 0.6$). We define $\kappa \overset{\text{def}}{=} \frac{1-p}{p}$ and recall this folk result.

▶ **Proposition 18.** *In $\mathcal{W}^p$, the probability to reach state 0 from state m is 1 when $p \leq \frac{1}{2}$ and κ^m otherwise.*

Here we introduce a subclass of LMC useful for our aims.

▶ **Definition 19.** *A LMC $(\mathcal{C}, \lambda)$ is said (p^+, N_0)-* divergent *with $p^+ > \frac{1}{2}$ and $N_0 \in \mathbb{N}$ if letting $F \overset{def}{=} \lambda^{-1}([0, N_0])$, for every $s \in S \setminus F$, $P^=(s) < 1$ implies $\frac{P^+(s)}{P^-(s)+P^+(s)} \geq p^+$.*

The (p^+, N_0)-divergence constrains states of levels larger than N_0, and imposes that, from those states that do not stay at the same level, the relative proportion of successors increasing their levels compared with those decreasing their levels is at least the value p^+ (itself larger than $\frac{1}{2}$). Note that a (p^+, N_0)-divergent LMC is also (p'^+, N_0')-divergent for all $\frac{1}{2} < p'^+ \leq p^+$ and $N_0' \geq N_0$. This will allow to adjust the corresponding set F that will be used in the approach, as will be seen in the experiments (Section 5).

To be able to apply the previous approach, it remains to examine under which conditions starting from a (p^+, N_0)-*divergent* LMC $\mathcal{C}$: (1) $\mathcal{W}^p$ is an abstraction, and (2) $\mathcal{C}'$ obtained via this abstraction is decisive w.r.t. F from s_0. The next proposition shows that $\mathcal{W}^p$ is an abstraction as soon as $1/2 < p < p^+$.

▶ **Proposition 20.** *Let $(\mathcal{C}, \lambda)$ be a (p^+, N_0)-divergent LMC and write $F \overset{def}{=} \lambda^{-1}([0, N_0])$. We define α as the restriction of λ to $S \setminus \mathsf{Av}(F)$, and we let $\frac{1}{2} < p < p^+$. Then $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ is an abstraction.*

The only point that needs to be checked is the monotony condition defining an abstraction. The proof is given in Appendix A.3 and distinguishes the states that almost-surely stay within the same level, and the other states; the rest is just calculation. The condition which is satisfied is even stronger than monotony: for all $s \in S \setminus (F \cup \mathsf{Av}(F))$ such that $\alpha(s) = n > N_0$ and $\mathrm{P}^{=}(s) < 1$: $1 - h(s) \geq \frac{2p-1}{(1-p)p} \cdot (\mathrm{P}^{-}(s) + \mathrm{P}^{+}(s)) \cdot (p^{+} - p)$, where $h(s)$ is the decreasing ratio at s, see page 9.

It remains to understand under which conditions the biased Markov chain of $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ is decisive w.r.t. T. To do that, let us introduce the key notion of *attractor* [1]: given a Markov chain $\mathcal{C} = (S, \mathrm{P})$ and $R \subseteq S$, R is an attractor if for all $s \in S$, $\mathbf{Pr}\big(\varrho^{\mathcal{C},s} \models \Diamond R\big) = 1$. There is a relation between attractor and decisiveness, stated as follows: if R is a *finite* attractor and $B \subseteq R$, then $\mathcal{C}$ is decisive w.r.t. B.

The next theorem gives a simple condition for a set R to be an attractor in a Markov chain, using a *Lyapunov function.*

▶ **Theorem 21.** *Let $\mathcal{C} = (S, P)$ be a Markov chain and $R \subseteq S$ s.t. for all $s \in S$, $\mathbf{Pr}\left(\varrho^{\mathcal{C},s} \models \Diamond R\right) > 0$, and let $\mathcal{L} : S \to \mathbb{R}^{+}$ be a Lyapunov function s.t. (1) for all $n \in \mathbb{N}$, $\mathcal{L}^{-1}([0, n])$ is finite, and (2) for all $s \in S \setminus R$, $\sum_{s' \in S} P(s, s') \cdot \mathcal{L}(s') \leq \mathcal{L}(s)$. Then for all $s \in S$, $\mathbf{Pr}\left(\varrho^{\mathcal{C},s} \models \Diamond R\right) = 1$.*

The full proof is rather involved and partly relies on martingale theory; it is given in Appendix A.3.

Using the previous theorem, we show that choosing $\mathcal{W}^p$ as an abstraction with $1/2 < p < p^{+}$ ensures decisiveness of $\mathcal{C}'$. The Lyapunov function will be obtained via the level function.

▶ **Proposition 22.** *Let $(\mathcal{C}, \lambda)$ be a (p^{+}, N_0)-divergent LMC, write $F \overset{def}{=} \lambda^{-1}([0, N_0])$, let α be the restriction of λ to $S \setminus \mathsf{Av}(F)$, and fix $\frac{1}{2} < p < p^{+}$. Then the biased Markov chain $\mathcal{C}'$ of $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ is decisive w.r.t. any $T \subseteq F$.*

The detailed proof is given in Appendix A.3; we explain here the rough idea. This proposition will be an application of Theorem 21 to $\mathcal{C}'$ with Lyapunov function $\mathcal{L}$ given by α (and additionally $\mathcal{L}(s_{-}) = 0$). So there will be some $N_1 \geq N_0$ such that $R \overset{\text{def}}{=} \mathcal{L}^{-1}\left([0, N_1]\right)$ is a finite attractor in $\mathcal{C}'$. Condition (2) of Theorem 21 is ensured by the fact that the level is unchanged after a transition from s if $\mathrm{P}^{=}(s) = 1$, and by the stronger condition given after Proposition 20 otherwise.

This proposition allows to apply the analysis of Subsection 4.1 to the biased Markov chain of $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$, yielding approximation and estimation algorithms for the original Markov chain. Nevertheless, as argued in Subsection 3.2, decisiveness is enough to ensure correctness of the SMC, but not enough for efficiency. Efficiency can be ensured, if the expected time for reaching $T \cup \mathsf{Av}(T)$ is finite. We will do so by strengthening the divergence condition of LMC.

To do so we present another theorem for the existence of an attractor, inspired by Foster's theorem [9], whose proof is given in Appendix A.3. Observe that here the requirement becomes: the average level decreases by some fixed $\varepsilon > 0$, and the other requirements are no more necessary.

▶ **Theorem 23.** *Let $\mathcal{C} = (S, P)$ be a Markov chain and $R \subseteq S$. If there exists $\mathcal{L} : S \to \mathbb{R}_{\geq 0}$ and $\varepsilon > 0$ such that for all $s \notin R$, $\mathcal{L}(s) - \sum_{s' \in S} P(s, s') \cdot \mathcal{L}(s') \geq \varepsilon$, then for all $s \notin R$ the expected time to reach R is finite and bounded by $\frac{\mathcal{L}(s)}{\varepsilon}$; in particular, R is an attractor of $\mathcal{C}$.*

We are now in a position to establish a sufficient condition for the biased LMC $\mathcal{C}'$ of $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ to be decisive with finite expected time to reach some finite target T.

▶ **Proposition 24.** *Let $(\mathcal{C}, \lambda)$ be a (p^+, N_0)-divergent LMC such that $\inf_{s \in \lambda^{-1}(]N_0,\infty[)} P^+(s) > 0$, and write $F \overset{def}{=} \lambda^{-1}([0, N_0])$. We define α as the restriction of λ to $S \setminus \mathsf{Av}(F)$, and we fix $\frac{1}{2} < p < p^+$. Then the biased Markov chain $\mathcal{C}'$ of $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ is decisive w.r.t. $T \subseteq F$ with finite expected time to reach $T \cup \mathsf{Av}_{\mathcal{C}'}(T)$.*

The full proof is given in Appendix A.3; the idea is as follows. We use the same Lyapunov function as before, and the stronger condition mentioned after Proposition 22 together with the constraint on P^+: applying Theorem 23, we are able to find a finite attractor $R \overset{\text{def}}{=} \mathcal{L}^{-1}([0, N_1]) \cup \{s_-\}$ for some $N_1 \geq N_0$, reachable in finite expected time (given by α). By analyzing the successive visits of R before reaching $T \cup \mathsf{Av}_{\mathcal{C}'}(T)$, we derive a bound on the expected time to reach $T \cup \mathsf{Av}_{\mathcal{C}'}(T)$, which (linearly) depends on the level of the initial state.

5 Applications and experiments

Probabilistic pushdown automata. Our method is applied to the setting of probabilistic pushdown automaton (pPDA) using the height of the stack as the level function λ. We only provide an informal definition for pPDA (see [5] for a formal definition).

A pPDA configuration consists of a stack of letters from an alphabet Σ and a state of an automaton. A set of rules describes how the top of the stack is modified. A rule $(q, a) \xrightarrow{w} (q', u)$ applies if the top of the stack matches the letter a and the current state is q. Then it replaces a by the word u and q by q'. The weight w of the rule is a polynomial in n, the size of the stack. Probability rules are defined with the relative weight of the rule which applies w.r.t. all rules that could apply. If the target T is defined as a regular language on the stack $\mathsf{Av}(T)$ is also a regular language (see [5]) that can be computed: the membership of a configuration to T and $\mathsf{Av}(T)$ is effective and not costly.

▶ **Example 25.** We consider the pPDA with a single (omitted) state with stack alphabet $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$ defined by the set of rules: $\{\mathtt{A} \xrightarrow{1} \mathtt{C}, \mathtt{A} \xrightarrow{n} \mathtt{BB}, \mathtt{B} \xrightarrow{5} \varepsilon, \mathtt{B} \xrightarrow{n} \mathtt{AA}, \mathtt{C} \xrightarrow{1} \mathtt{C}\}$. Starting with the stack containing only A, the target set $T = \{\varepsilon\}$ is the configuration with the empty stack and $\mathsf{Av}(T)$ is the set of configurations containing a C. Let us describe some possible evolutions. From the initial configuration two rules apply by reading A: the new stack is C with probability $\frac{1}{2}$ or BB with probability $\frac{1}{2}$. From the stack BB, two rules apply by reading the first B: the new stack is then B with probability $\frac{5}{7}$ (7 is the sum of the weight of $\mathtt{B} \xrightarrow{5} \varepsilon$ and of $\mathtt{B} \xrightarrow{n} \mathtt{AA}$, with $n = 2$), and BAA with probability $\frac{2}{7}$.

The approach described previously applies to pPDA, as soon as the LMC defined by the pPDA can be proven to be (p^+, N_0)-divergent for some $p^+ > \frac{1}{2}$ and $N_0 \in \mathbb{N}$. This condition can be ensured by some syntactical constraints on the pPDA.

Implementation. Since SMC with importance sampling is already present in the tool Cosmos [4], we only added the mapping function λ in order to apply our method. We focus here on the implementation details of Algorithm 1, which (to the best of our knowledge) has never been done.

Algorithm 1 requires to sum up a large number of probabilities accurately while those probabilities are of different magnitudes. We have experimentally observed that without dedicated summation algorithms, the implementation of this algorithm does not converge. We therefore propose a data structure with better accuracy when summing up positive values at the cost of increased memory consumption and time. This data structure encodes a floating point number r as a table of integers of size 512 where the cell c at index i stores the value $c2^{-i}$, with c being a small enough integer to be represented exactly. The probability r is the sum of the values of the table.

We specialized Algorithm 1 (called AlgoDec in the following), when the function to be evaluated satisfies the following *monoidal property*: for all $\rho = \rho_1\rho_2$, $f(\rho) = f(\rho_1) \cdot f(\rho_2)$; this property is in particular satisfied by the likelihood related to an importance sampling. It is thus possible to merge paths leading to the same state and store only for each state the probability to reach it and the weighted average likelihood of the merged paths. In practice, this leads to a large improvement. Another improvement is the use of a heap where states are ordered by their probability to be reached: the algorithm will converge faster. The termination of the algorithm still holds as the heap management happens to be fair, see [3] for details.

Experimental studies. We first ran experiments[7] on the example depicted on Figure 1. As there are only two states per level, the numerical algorithm (AlgoDec) with important sampling is very efficient and computes an interval of 0.0258657 ± 10^{-8} in 10ms. The SMC approach computes a confidence interval of 0.02586 ± 10^{-4} in 135s. As expected the SMC approach is much slower on such a small toy example.

The pPDA of Example 25 is both decisive and a (p, N_0)-divergent LMC for $1/2 < p \leq \frac{N_0}{N_0+5}$ so that $(\mathcal{W}^p, [0, N_0])$ defines an abstraction. We compare the use of importance sampling with different values of p to standard SMC and AlgoDec. In Figure 2 each point is the result of a computation with or without importance sampling. The value 0.3151 is contained in the intervals returned by all numerical computations and all but one confidence intervals of SMC (consistent with 120 experiments and a confidence level of 0.99).

Figure 2a depicts the computation time w.r.t. the width of the confidence interval for the two algorithms over three Markov chains: the initial Markov chain, the importance sampling using $\mathcal{W}^{0.6}$ as abstraction and the importance sampling with $\mathcal{W}^{0.51}$ as abstraction. Looking only at SMC (dotted line on the figure) the computation time scales the same way on the three curves with the standard SMC taking more time. Looking at the AlgoDec curves (solid line) with a well-chosen value of $p = 0.6$ this algorithm is very fast but with another value of p or without importance sampling the performance quickly degrades.

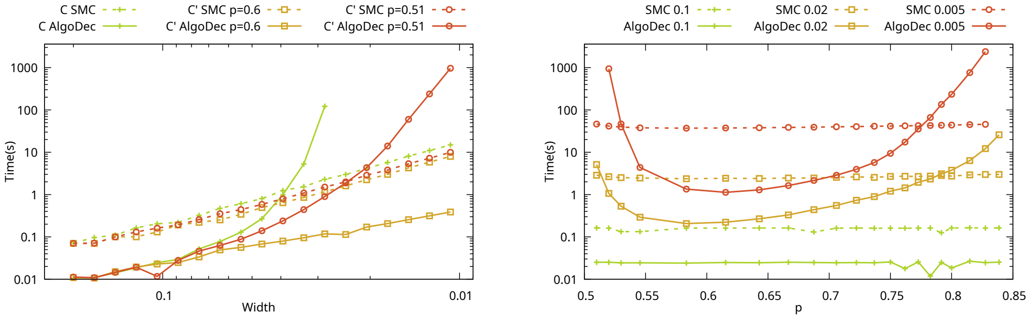
To better understand how the computation time increases w.r.t. p we plot it in Figure 2b. The SMC is barely sensible to the value of p while the computation time of AlgoDec reaches a minimum at around $p = 0.6$ and becomes intractable when p moves away from this value.

▶ **Example 26.** We consider the pPDA with a single state with stack alphabet $\{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$ defined by the set of rules: $\{\mathtt{A} \xrightarrow{1} \mathtt{B}, \mathtt{A} \xrightarrow{1} \mathtt{C}, \mathtt{B} \xrightarrow{10} \varepsilon, \mathtt{B} \xrightarrow{10+n} \mathtt{AA}, \mathtt{C} \xrightarrow{10} \mathtt{A}, \mathtt{C} \xrightarrow{10+n} \mathtt{BB}\}$ starting with stack $\mathtt{A}$, target configuration $T = \{\varepsilon\}$ and $\mathsf{Av}(T) = \emptyset$.

Example 26 is not decisive but is a (p, N_0)-divergent LMC for $1/2 < p \leq \frac{10+N_0}{20+N_0}$ thus $(\mathcal{W}^p, [0, N_0])$ defines an abstraction. In Figure 3 we plot the computation time w.r.t. p. The probability 0.516318 is contained in all the results. As in Example 25, AlgoDec is very sensitive to the value of p while SMC is not. In this example SMC is always faster than AlgoDec with similar computation times for a well-chosen value of p.

From our experiments we observe that while importance sampling can be applied both to AlgoDec and SMC, as soon as the size of the state space grows, AlgoDec is not tractable.

[7] All the experiments are run with a timeout of 1 hour and a confidence level set to 0.99.



(a) Computation time as a function of the precision, the width is given in logarithmic scale.

(b) Computation time as a function of p.

Figure 2 Computation time for Example 25 in logarithmic scale. Given a value for p, the threshold N_0 is chosen as the smallest integer such that $(\mathcal{W}^p, [0, N_0])$ defines an α-abstraction.

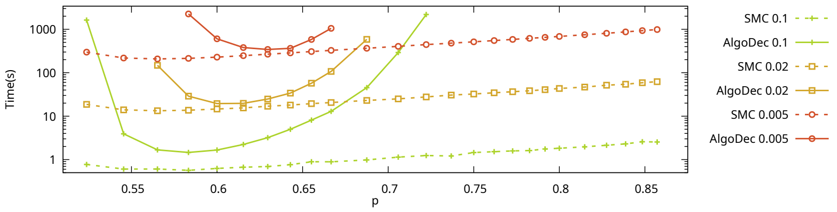


Figure 3 Computation time as a function of p for Example 26.

Additionally, the few experiments that we have conducted suggest the following methodology to analyze Markov chains: apply SMC with importance sampling for various values of p; find the "best" p; apply AlgoDec with that value of p (when possible).

6 Conclusion

We have recalled two standard approaches to the analysis of reachability properties in infinite Markov chains, a deterministic approximation algorithm, and a probabilistic algorithm based on statistical model checking. For their correctness or termination, they both require the Markov chain to satisfy a *decisiveness* property. Analyzing non decisive Markov chains is therefore a challenge.

In this work, we have introduced the notion of abstraction for a Markov chain and developed a theoretical method based on importance sampling to "transform" a non decisive Markov chain into a decisive one, allowing to transfer the analysis of the non decisive Markov chain to the decisive one. Then we have presented a concrete framework where the Markov chain is a layered Markov chain (LMC), the abstraction is done via a random walk, and given conditions that ensure that this abstract chain is decisive. Finally we have implemented the two algorithms within the tool Cosmos, and compared their respective performances on some examples given as probabilistic pushdown automata (which are specific LMCs).

There are several further research directions that could be investigated. First while (one-dimensional) random walks have closed forms for reachability probabilities, other (more complex) models also enjoy such a property, and could therefore be used for abstractions. Second, the divergence requirements are based on conditions for one-step transitions and could be relaxed to an arbitrary (but fixed) number of steps. Finally, more systematic, and even automatic, approaches could be investigated, that would compute adequate abstractions to adequate classes of Markov chains allowing to use our approach.

References

1 P. A. Abdulla, N. B. Henda, and R. Mayr. Decisive Markov chains. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:7)2007.

2 P. Ballarini, B. Barbot, M. Duflot, S. Haddad, and N. Pekergin. HASL: A new approach for performance evaluation and model checking from concepts to experimentation. *Performance Evaluation*, 90:53–77, 2015. doi:10.1016/j.peva.2015.04.003.

3 B. Barbot, P. Bouyer, and S. Haddad. Beyond decisiveness of infinite markov chains. *CoRR arXiV*, 2024. doi:10.48550/arXiv.2409.18670.

4 B. Barbot, S. Haddad, and C. Picaronny. Coupling and importance sampling for statistical model checking. In *18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 331–346. Springer, 2012. doi:10.1007/978-3-642-28756-5_23.

5 J. Esparza, A. Kucera, and R. Mayr. Model Checking Probabilistic Pushdown Automata. *Logical Methods in Computer Science*, 2(1), 2006. doi:10.2168/LMCS-2(1:2)2006.

6 G. Fayolle, V.A. Malyshev, and M. V. Menshikov. *Topics in the Constructive Theory of Countable Markov Chains.* Cambridge University Press, 1995.

7 A. Finkel, S. Haddad, and L. Ye. About decisiveness of dynamic probabilistic models. In *34th International Conference on Concurrency Theory (CONCUR'23)*, volume 279 of *LIPIcs*, pages 14:1–14:17. Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CONCUR.2023.14.

8 A. Finkel, S. Haddad, and L. Ye. Introducing divergence for infinite probabilistic models. In *17th International Conference on Reachability Problems (RP'23)*, volume 14235 of *LNCS*, pages 1–14. Springer, 2023. doi:10.1007/978-3-031-45286-4_10.

9 F. G. Foster. On the Stochastic Matrices Associated with Certain Queuing Processes. *The Annals of Mathematical Statistics*, 24(3):355–360, 1953. doi:10.1214/aoms/1177728976.

10 W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58:13–30, 1963.

11 W. Kahan. Pracniques: further remarks on reducing truncation errors. *Comm. ACM*, 8(1):40, 1965. doi:10.1145/363707.363723.

12 H. Kahn and T. E. Harris. Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30, 1951.

13 G. Rubino and B. Tuffin, editors. *Rare Event Simulation using Monte Carlo Methods.* Wiley, 2009. doi:10.1002/9780470745403.

14 J. M. Rutten, M. Z. Kwiatkowska, G. Norman, D. Parker, and P. Panangaden. *Mathematical techniques for analyzing concurrent and probabilistic systems*, volume 23 of *CRM monograph series.* American Mathematical Society, 2004. URL: http://www.ams.org/publications/authors/books/postpub/crmm-23.

15 H. L. S. Younes, E. M. Clarke, and P. Zuliani. Statistical verification of probabilistic properties with unbounded until. In *13th Brazilian Symposium on Formal Methods (SBMF'10)*, volume 6527 of *LNCS*, pages 144–160. Springer, 2010. doi:10.1007/978-3-642-19829-8_10.

16 H. L. S. Younes and R. G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368–1409, 2006. doi:10.1016/J.IC.2006.05.002.

17 H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *14th International Conference on Computer-Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 223–235. Springer, 2022. doi:10.1007/3-540-45657-0_17.

We report here some of the missing proofs and discussions. More details are given in [3].

A Some missing proofs of Section 4

A.1 Proofs of results of Section 4.1

We first state a useful lemma proved in a simple way.

▶ **Lemma 27.** $\mathbf{Pr}(\rho) > 0$ *and* $\mathit{last}(\rho) \notin \mathsf{Av}_{\mathcal{C}}(F)$ *imply* $\mathbf{Pr}'(\rho) > 0$.

Proof of Proposition 10. We proceed by a sequence of equalities:

$$\begin{aligned}
&\mathbf{E}\big(f_{L',T}\big(\varrho^{\mathcal{C}',s_0}\big)\big) \quad = \quad \mathbf{E}\big(\big(f_{L',T}\cdot \mathbb{1}_{\neg\Diamond T}\big)\big(\varrho^{\mathcal{C}',s_0}\big) + \big(f_{L',T}\cdot \mathbb{1}_{\Diamond T}\big)\big(\varrho^{\mathcal{C}',s_0}\big)\big)\\
&= \mathbf{E}\big(\big(f_{L',T}\cdot \mathbb{1}_{\Diamond T}\big)\big(\varrho^{\mathcal{C}',s_0}\big)\big) \quad = \sum_{\substack{\rho\in(S\setminus\mathsf{Av}_{\mathcal{C}}(F))^+ \text{ s.t.}\\ \mathbf{Pr}'(\rho)>0 \text{ and } \mathit{last}(\rho)=\mathit{first}_T(\rho)}} L'(\rho)\cdot\mathbf{Pr}'(\rho)\\
&= \sum_{\substack{\rho\in(S\setminus\mathsf{Av}_{\mathcal{C}}(F))^+ \text{ s.t.}\\ \mathbf{Pr}'(\rho)>0 \text{ and } \mathit{last}(\rho)=\mathit{first}_T(\rho)}} L(\rho)\cdot\frac{\mathbf{Pr}(\rho)}{\mathbf{Pr}'(\rho)}\cdot\mathbf{Pr}'(\rho)\\
&= \sum_{\substack{\rho\in(S\setminus\mathsf{Av}_{\mathcal{C}}(F))^+ \text{ s.t.}\\ \mathbf{Pr}'(\rho)>0,\ \mathbf{Pr}(\rho)>0 \text{ and } \mathit{last}(\rho)=\mathit{first}_T(\rho)}} L(\rho)\cdot\mathbf{Pr}(\rho)\\
&= \sum_{\substack{\rho\in(S\setminus\mathsf{Av}_{\mathcal{C}}(F))^+ \text{ s.t.}\\ \mathbf{Pr}(\rho)>0, \text{ and } \mathit{last}(\rho)=\mathit{first}_T(\rho)}} L(\rho)\cdot\mathbf{Pr}(\rho) \ {\scriptstyle\text{(by Lemma 27)}}\\
&= \sum_{\substack{\rho\in S^+ \text{ s.t.}\\ \mathbf{Pr}(\rho)>0, \text{ and } \mathit{last}(\rho)=\mathit{first}_T(\rho)}} L(\rho)\cdot\mathbf{Pr}(\rho) \ {\scriptstyle\textit{(since } \mathsf{Av}_{\mathcal{C}}(F)\subseteq\mathsf{Av}_{\mathcal{C}}(T))} \quad = \quad \mathbf{E}\big(f_{L,T}\big(\varrho^{\mathcal{C},s_0}\big)\big)
\end{aligned}$$

◀

A.2 Proofs of results of Section 4.2

Before going to the proof of Proposition 16, we state a useful lemma.

▶ **Lemma 28.** *Let* $\mathcal{C}'$ *be the biased Markov chain of* $(\mathcal{C},F) \overset{\alpha}{\hookrightarrow} (\mathcal{C}^\bullet,F^\bullet)$. *Then for all* $s\in S\setminus\mathsf{Av}_{\mathcal{C}}(F)$,

- *for all* ρ *starting from* s *with* $\mathbf{Pr}'(\rho)>0$ *and* $\mathit{last}(\rho)\neq s_-$, $\mathbf{Pr}(\rho)=\mathbf{Pr}'(\rho)\cdot\frac{\mu_{F^\bullet}(\alpha(s))}{\mu_{F^\bullet}(\alpha(\mathit{last}(\rho)))}$;
- $\mu_{\mathcal{C},T}(s)=\mu_{F^\bullet}(\alpha(s))\cdot\mu_{\mathcal{C}',T}(s)$ *and* $\mu_{\mathcal{C},F}(s)=\mu_{F^\bullet}(\alpha(s))\cdot\mu_{\mathcal{C}',F}(s)$.

Proof. We establish the first property by induction. Let ρ be a path starting from s with $\mathbf{Pr}'(\rho)>0$ and $\mathit{last}(\rho)\neq s_-$. If ρ is the single state s then $\mathbf{Pr}'(\rho)=\mathbf{Pr}(\rho)=1$. Since $\mathit{last}(\rho)=s$, the base case is proved. Assume now that $\rho=\rho's''$ with $\mathbf{Pr}'(\rho)>0$ and $s'\overset{\text{def}}{=}\mathit{last}(\rho')\neq s_-$. Then: $\mathbf{Pr}(\rho)=\mathbf{Pr}(\rho')\cdot\mathrm{P}(s',s'')=\mathbf{Pr}(\rho')\cdot\mathrm{P}'(s',s'')\cdot\frac{\mu_{F^\bullet}(\alpha(s'))}{\mu_{F^\bullet}(\alpha(s''))}$. It is well-defined due to Lemma 13. Applying the induction hypothesis, we getthe induction step:

$$\mathbf{Pr}(\rho)=\mathbf{Pr}'(\rho')\cdot\frac{\mu_{F^\bullet}(\alpha(s))}{\mu_{F^\bullet}(\alpha(s'))}\cdot\mathrm{P}'(s',s'')\cdot\frac{\mu_{F^\bullet}(\alpha(s'))}{\mu_{F^\bullet}(\alpha(s''))}=\mathbf{Pr}'(\rho)\cdot\frac{\mu_{F^\bullet}(\alpha(s))}{\mu_{F^\bullet}(\alpha(s''))}.$$

The paths that reach T (resp. F) from s are the same in $\mathcal{C}$ and $\mathcal{C}'$, and they do not reach s_-. Pick such a path ρ. Then due to the previous property: $\mathbf{Pr}(\rho)=\mathbf{Pr}'(\rho)\cdot\mu_{F^\bullet}(\alpha(s))$. Summing over all such paths establishes the second property. ◀

While the previous property allows to solve the reachability problem in $\mathcal{C}$ using $\mathcal{C}'$, the next proposition extends it to the reward reachability problem.

Proof of Proposition 16. By definition, $f_{L',T}$ assigns 0 to infinite paths not visiting T. Assume now that ρ is an infinite path visiting T. Then $f_{L',T}(\rho) = (L \cdot \gamma)\big(\rho_{\leq \mathit{first}_T(\rho)}\big)$. Let $\rho' = \rho_{\leq \mathit{first}_T(\rho)}$. It does not visit s_-, hence $\gamma(\rho') = \frac{\mathbf{Pr}(\rho')}{\mathbf{Pr}'(\rho')} = \frac{\mu_{F^\bullet}(\alpha(s))}{\mu_{F^\bullet}(\alpha(\mathit{last}(\rho')))}$ by Lemma 28. Since $\mathit{last}(\rho') \in T$, $\gamma(\rho') = \mu_{F^\bullet}(\alpha(s))$. This implies the first part of the proposition. The restriction to the case of the indicator function is immediate. ◀

A.3 Proofs of results of Section 4.3

We first establish that random walks parametrized by $\frac{1}{2} < p < p^+$ are abstractions for a (p^+, N_0)- divergent LMC and give useful information on the decreasing ratio for states s with $\mathrm{P}^+(s) < 1$.

Proof of Proposition 20. We denote $\mu_{\mathcal{W}^p,[0,N_0]}$ more simply by $\mu^\bullet_{[0,N_0]}$. We pick $s \in S \setminus (F \cup \mathsf{Av}(F))$ such that $\alpha(s) = n > N_0$, and we distinguish between two cases. If $\mathrm{P}^=(s) = 1$, since $\alpha(s') = n$ for all s' s.t. $\mathrm{P}(s,s') > 0$, we easily infer that $\sum_{s' \notin \mathsf{Av}_\mathcal{C}(F)} \mathrm{P}(s,s') \cdot \mu^\bullet_{[0,N_0]}(\alpha(s')) \geq \mu^\bullet_{[0,N_0]}(\alpha(s))$. Otherwise, we can compute (all details are given in [3]):

$$\begin{array}{rcl}
1 - h(s) & \geq & 1 - \dfrac{1}{\mu^\bullet_{[0,N_0]}(n)} \big(\mu^\bullet_{[0,N_0]}(n-1) \cdot \mathrm{P}^-(s) + \mu^\bullet_{[0,N_0]}(n) \cdot \mathrm{P}^=(s) + \mu^\bullet_{[0,N_0]}(n+1) \cdot \mathrm{P}^+(s)\big) \\
 & & (\text{since } \mathrm{P}^+(s) + \mathrm{P}^-(s) + \mathrm{P}^=(s) = 1 \text{ and } \mu^\bullet_{[0,N_0]}(x) \text{ is non increasing w.r.t. } x) \\
 & & \\
 & = & (1 - \dfrac{1}{\kappa}) \cdot \mathrm{P}^-(s) + (1-\kappa) \cdot \mathrm{P}^+(s) \\
 & = & \frac{2p-1}{(1-p)p}(\mathrm{P}^-(s) + \mathrm{P}^+(s)) \left(-p \cdot (1 - \dfrac{\mathrm{P}^+(s)}{\mathrm{P}^-(s) + \mathrm{P}^+(s)}) + (1-p) \cdot \dfrac{\mathrm{P}^+(s)}{\mathrm{P}^-(s) + \mathrm{P}^+(s)}\right) \\
 & \geq & \frac{2p-1}{(1-p)p}(\mathrm{P}^-(s) + \mathrm{P}^+(s)) \big(p^+ - p\big) > 0 \quad (\text{since } \frac{1}{2} < p < p^+)
\end{array}$$

This concludes the proof of monotony, implying that $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ is an abstraction. ◀

Out of the above proof, a stronger condition than monotony happens to be satisfied.

▶ **Corollary 29.** *Let $(\mathcal{C}, \lambda)$ be a (p^+, N_0)-divergent LMC. We define α as the restriction of λ to $S \setminus \mathsf{Av}(F)$, and we let $\frac{1}{2} < p < p^+$. Then the monotony condition satisfied by the abstraction $(\mathcal{C}, F) \overset{\alpha}{\hookrightarrow} (\mathcal{W}^p, [0, N_0])$ can be strengthened as follows. For all $s \in S \setminus (F \cup \mathsf{Av}(F))$ such that $\alpha(s) = n > N_0$ and $P^=(s) < 1$: $1 - h(s) \geq \frac{2p-1}{(1-p)p} \cdot (P^-(s) + P^+(s)) \cdot (p^+ - p) > 0$, where $h(s)$ is the decreasing ratio at s, see page 9.*

Before turning to the proof of Theorem 21, we first establish a sufficient condition to be an attractor in a Markov chain.

▶ **Lemma 30.** *Let $\mathcal{C} = (S, P)$ be a Markov chain, $s_0 \in S$ and $R \subseteq S$ s.t. for all $s \in S$, $\mathbf{Pr}\left(\varrho^{\mathcal{C},s} \models \Diamond R\right) > 0$. Assume that for every $\delta > 0$, there exists a finite set $S_\delta \subseteq S$ and $m_\delta \in \mathbb{N}$ such that $\mathbf{Pr}\left(\bigwedge_{i \geq m_\delta} X_i^{\mathcal{C},s_0} \in S_\delta\right) > 1 - \delta$. Then $\mathbf{Pr}\left(\varrho^{\mathcal{C},s_0} \models \Diamond R\right) = 1$.*

Proof. Fix some $\delta > 0$. Let $\ell \in \mathbb{N}$ be the maximal length over $s \in S_\delta$ of a shortest path from s to R and $p_{\min} > 0$ the minimal probability of these paths. Let $k \in \mathbb{N}$. Then $\mathbf{Pr}\left(\bigwedge_{m_\delta \leq i \leq m_\delta + k\ell} X_i^{\mathcal{C},s_0} \notin R \mid \bigwedge_{m_\delta \leq i} X_i^{\mathcal{C},s_0} \in S_\delta\right) \leq (1 - p_{\min})^k$.

Letting k go to ∞, one gets $\mathbf{Pr}\left(\bigwedge_{m_\delta \leq i} X_i^{\mathcal{C},s_0} \notin R \mid \bigwedge_{m_\delta \leq i} X_i \in S_\delta\right) = 0$ implying $\mathbf{Pr}\left(\varrho^{\mathcal{C},s_0} \models \Diamond R \mid \bigwedge_{m_\delta \leq i} X_i^{\mathcal{C},s_0} \in S_\delta\right) = 1$. Thus $\mathbf{Pr}\left(\varrho^{\mathcal{C},s_0} \models \Diamond R\right) > 1 - \delta$. Since δ is arbitrary, one gets $\mathbf{Pr}\left(\varrho^{\mathcal{C},s_0} \models \Diamond R\right) = 1$. ◀

Then using both martingale theory and the previous lemma we establish another sufficient condition based on a non negative state function non increasing on average (i.e. the expected next value). A similar proof for recurrence of irreducible Markov chains can be found in [6].

Proof of Theorem 21. Since we are interested in reachability of R, w.l.o.g. we assume that all $s \in R$ are absorbing and thus $\sum_{s' \in S} \mathrm{P}(s, s') \cdot \mathcal{L}(s') = \mathcal{L}(s)$.

We fix some initial state s_0 and consider the random sequence of states $(X_n^{\mathcal{C},s_0})_{n \in \mathbb{N}}$, which we simply write $(X_n)_{n \in \mathbb{N}}$. Define $\mathcal{F}_n$ the σ-algebra generated by $(X_m)_{m \leq n}$ and $Y_n = \mathcal{L}(X_n)$. Due to the inequation $\sum_{s' \in S} \mathrm{P}(s, s')\mathcal{L}(s') \leq \mathcal{L}(s)$ for all $s \in S$ and the memoryless property of Markov chain $\mathbf{E}\left(Y_{n+1} \mid \mathcal{F}_n\right) = \mathbf{E}\left(Y_{n+1} \mid X_n\right) = \sum_{s' \in S} \mathrm{P}(X_n, s') \cdot \mathcal{L}(s') \leq \mathcal{L}(X_n) = Y_n$. Thus $(Y_n)_{n \in \mathbb{N}}$ is a supermartingale. Consider the limit Y_∞ of this supermartingale: it satisfies $\mathbf{E}\left(Y_\infty\right) \leq \mathcal{L}(s_0) < \infty$.

We use Lemma 30 to conclude that R is an attractor. Towards a contradiction assume that the sufficient condition of Lemma 30 is not satisfied. There is some $\delta > 0$ such that for all finite set S^* and $m \in \mathbb{N}$, $\mathbf{Pr}\left(\bigvee_{m \leq i} X_i \notin S^*\right) \geq \delta$. For every $n \in \mathbb{N}$, choose $S_n^* = \mathcal{L}^{-1}([0, n])$. The event $E_1 = \left\{\bigwedge_{m \in \mathbb{N}} \bigvee_{m \leq i} X_i \notin S_n^*\right\}$ is the limit of decreasing events with probability larger than or equal to δ. So $\mathbf{Pr}(E_1) \geq \delta$. Consider the event $E_2 = \{Y_\infty \geq n\}$: then $E_1 \subseteq E_2$. Thus $\mathbf{Pr}\left(E_2\right) \geq \delta$. Now, by Markov's inequality applied to the random variable Y_∞, $\mathbf{E}\left(Y_\infty\right) \geq n \cdot \mathbf{Pr}\left(Y_\infty \geq n\right) \geq n\delta$. Since this is true for all n, $\mathbf{E}\left(Y_\infty\right) = \infty$, which is a contradiction. The sufficient condition of Lemma 30 is then satisfied, which implies that R is an attractor. ◀

The next proposition shows that choosing $\mathcal{W}^p$ as an abstraction with $1/2 < p < p^+$ ensures decisiveness of $\mathcal{C}'$.

Proof of Proposition 22. From Proposition 20, $(\mathcal{W}^p, [0, N_0])$ is a α-abstraction of $(\mathcal{C}, F)$, so $\mathcal{C}'$ is well-defined. We exhibit some $N_1 \geq N_0$ s.t. $\alpha^{-1}([0, N_1]) \cup \{s_-\}$ is a finite attractor of $\mathcal{C}'$, which implies decisiveness of $\mathcal{C}'$ w.r.t. T (thanks to Lemma 3.4 of [1]).

To do so, we apply Theorem 21 to the Markov chain $\mathcal{C}$, using the layered function $\mathcal{L}$, which coincides with α on $S \setminus \mathsf{Av}(F)$ and extended by $\mathcal{L}(s_-) = 0$ as the Lyapunov function. It remains to show the inequation on $\mathcal{L}$.

Let $s \in S' \setminus \{s_-\}$ with $\alpha(s) = n > N_0$. If $\mathrm{P}^=(s) = 1$, it is easy to get that $\sum_{s' \in S'} \mathrm{P}'(s, s') \cdot \mathcal{L}(s') \leq n = \mathcal{L}(s)$. If $\mathrm{P}^=(s) < 1$, we compute:

$$\begin{aligned}
\mathcal{L}(s) - \textstyle\sum_{s' \in S'} \mathrm{P}'(s, s') \cdot \mathcal{L}(s') \quad &= \quad \textstyle\sum_{s' \in S'} \mathrm{P}'(s, s') \cdot (\mathcal{L}(s) - \mathcal{L}(s')) \\
= \quad & \textstyle\sum_{k \geq n+1} \sum_{\substack{s' \in S' \text{s.t.} \\ \mathcal{L}(s') = k}} (n - k)\mathrm{P}'(s, s') + \sum_{\substack{s' \in S' \text{ s.t.} \\ \mathcal{L}(s') = n-1}} \mathrm{P}'(s, s') + n\mathrm{P}'(s, s_-) \\
= \quad & \textstyle\sum_{k \geq n+1} \sum_{\substack{s' \in S' \text{ s.t.} \\ \mathcal{L}(s') = k}} -\kappa^{k-n}(k - n)\mathrm{P}(s, s') + \sum_{\substack{s' \in S' \text{ s.t.} \\ \mathcal{L}(s') = n-1}} \frac{1}{\kappa}\mathrm{P}(s, s') + n(1 - h(s))
\end{aligned}$$

Since $\lim_{x \to +\infty} x\kappa^x = 0$, we let $B = \sup_{x \geq 0} x\kappa^x \geq \kappa$. We get after some calculation, and using Corollary 29:

$$\mathcal{L}(s) - \textstyle\sum_{s' \in S'} \mathrm{P}(s, s') \cdot \mathcal{L}(s') \quad \geq \quad \mathrm{P}^+(s)(-B + n\frac{2p-1}{(1-p)p} \cdot (p^+ - p))$$

It is then sufficient to define N_1 such that $-B + N_1 \frac{2p-1}{(1-p)p} \cdot (p^+ - p) \geq 0$. The condition of Theorem 21 holds for all states. By the theorem, $\mathcal{L}^{-1}([0, N_1]) = \alpha^{-1}([0, N_1]) \cup \{s_-\}$ is then a finite attractor of $\mathcal{C}'$, which concludes the proof. ◀

This theorem shows that the existence of a *Lyapunov* state function $\mathcal{L}$ for some set R ensures that R is an attractor and that the expected time to reach it with an explicit upper bound (given in the proof) depending on the value of $\mathcal{L}$ for the initial state.

Proof of Theorem 23. W.l.o.g. we assume that all states in R are absorbing. Pick some $s \in S$. Define $X_n(s)$ as the random state at time n when starting from s (that is, $X_n^{\mathcal{C},s}$) and $T_{s,R}$ the random time (in $\mathbb{N} \cup \{\infty\}$) to reach R from s. Observe that: $\mathbf{E}\left(T_{s,R}\right) = \sum_{n\in\mathbb{N}} \mathbf{Pr}\left(X_n(s) \notin R\right)$.

On the other hand, the inequality satisfied by $\mathcal{L}$ can be rewritten as follows. For Y random variable over $S \setminus R$, $\mathbf{E}(\mathcal{L}(X_1(Y)) - \mathcal{L}(Y)) \leq -\varepsilon$.

Let $n \in \mathbb{N}$. Since states of R are absorbing,

$$\begin{aligned} \mathbf{E}\left(\mathcal{L}(X_{n+1}(s)) - \mathcal{L}(X_0(s))\right) \quad &= \quad \textstyle\sum_{k\leq n} \mathbf{E}\left(\mathcal{L}(X_{k+1}(s)) - \mathcal{L}(X_k(s)) \cdot \mathbf{1}_{X_k(s)\notin R}\right) \\ = \quad \textstyle\sum_{k\leq n} \mathbf{E}\left(\mathcal{L}(X_{k+1}(s)) - \mathcal{L}(X_k(s)) \mid X_k(s) \notin R\right) &\cdot \mathbf{Pr}\left(X_k(s) \notin R\right) \\ \leq \quad -\varepsilon \textstyle\sum_{k\leq n} \mathbf{Pr}\left(X_k(s) \notin R\right) & \end{aligned}$$

Since $\mathcal{L}$ is nonnegative and $\mathbf{E}(\mathcal{L}(X_0(s))) = \mathcal{L}(s)$, one gets: $\sum_{k\leq n} \mathbf{Pr}(X_k(s) \notin R) \leq \frac{\mathcal{L}(s)}{\varepsilon}$. Letting n go to ∞, one gets $\mathbf{E}(T_{s,R}) \leq \frac{\mathcal{L}(s)}{\varepsilon}$, which concludes the proof. ◀

The next proposition shows that choosing $\mathcal{W}^p$ as an abstraction with $\frac{1}{2} < p < p^+$ ensures decisiveness of $\mathcal{C}'$ and finite expected time for statistical model checking due to the previous theorem.

Proof of Proposition 24. Let $\hat{p} = \inf_{s\in\lambda^{-1}(]N_0,\infty[)} \mathrm{P}^+(s)$. Due to Proposition 22, we already know that $\mathcal{C}'$ is decisive w.r.t. T. It remains to establish that the expected time to reach $T \cup \mathsf{Av}_{\mathcal{C}'}(T) = T \cup \{s_-\}$ is finite. For every $s \in S \setminus \mathsf{Av}(F)$ with $\alpha(s) > N_0$, $0 < \hat{p} \leq \mathrm{P}^+(s)$, hence $\mathrm{P}^=(s) < 1$. Therefore, using Corollary 29, for all $s \in S' \setminus \{s_-\} = S \setminus \mathsf{Av}_{\mathcal{C}}(F)$ with $\alpha(s) > N_0$,

$$\begin{aligned} \mathcal{L}(s) - \textstyle\sum_{s'\in S'} \mathrm{P}(s,s') \cdot \mathcal{L}(s') \quad &\geq \quad \mathrm{P}^+(s) \cdot \left(-B + n \cdot \tfrac{2p-1}{(1-p)p} \cdot (p^+ - p)\right) \\ &\geq \quad \hat{p} \cdot \left(-B + n \cdot \tfrac{2p-1}{(1-p)p} \cdot (p^+ - p)\right) \end{aligned}$$

Let $N_1 \geq N_0$ be such that $\hat{p} \cdot \left(-B + N_1 \cdot \frac{2p-1}{(1-p)p} \cdot (p^+ - p)\right) \geq 1$ and $R = \mathcal{L}^{-1}([0, N_1]) = \alpha^{-1}([0, N_1]) \cup \{s_-\}$. Then the condition of Theorem 23 holds with $\varepsilon = 1$. Applying it, the expected time to reach R from $s \in S \setminus \mathsf{Av}(F)$ with $\alpha(s) > N_1$ is finite and bounded by $\mathcal{L}(s) = \alpha(s)$.

It remains to establish that the expected time to reach T from every state is finite. We fix an initial state $s_0 \in S'$ and we consider the infinite random sequence $(\gamma(n))_{n\in\mathbb{N}}$, defined inductively as follows: $\gamma(0) = \min\{k \mid X_k(s_0) \in R\}$ and $\gamma(n+1) = \min\{k > \gamma(n) \mid X_k(s_0) \in R\}$; those are the successive times of visits in R. Since R is an attractor, this sequence is defined almost everywhere. Let $h_{\max} = \max\left\{\mathcal{L}(s') \mid \exists s \in R \text{ s.t. } \mathrm{P}'(s,s') > 0\right\}$ the maximal level that can be reached in one step from R. Due to the previous paragraph, for all n and $s \in R$, $\mathbf{E}\left(\gamma(n+1) - \gamma(n) \mid X_{\gamma(n)} = s\right) \leq 1 + h_{\max}$ and $\mathbf{E}\left(\gamma(0)\right) \leq \mathcal{L}(s_0)$.

Define $\tilde{T} = T \cup \mathsf{Av}_{\mathcal{C}'}(T) = T \cup \{s_-\}$, $\tau \overset{\text{def}}{=} \tau^{\mathcal{C}',s_0,\tilde{T}}$ the (random) time to reach $\tilde{T}$ from s_0 in $\mathcal{C}'$, $\ell_{\max}$ as the maximal length of a shortest path from $s \in R$ to $T \cup \mathsf{Av}_{\mathcal{C}'}(T)$ and $p_{\min}$ the minimal probability of these paths. Then:

$$\begin{aligned}
&\mathbf{E}\left(\tau\right)\\
&= \mathbf{E}(\gamma(0)) + \textstyle\sum_{n\in\mathbb{N}} \mathbf{E}\left(\gamma(n+1)-\gamma(n) \mid \bigwedge_{m\leq n} X_{\gamma(m)} \notin \tilde{T}\right) \cdot \mathbf{Pr}\left(\bigwedge_{m\leq n} X_{\gamma(m)} \notin \tilde{T}\right)\\
&= \mathbf{E}(\gamma(0)) + \textstyle\sum_{n\in\mathbb{N}} \mathbf{E}\left(\gamma(n+1)-\gamma(n) \mid X_{\gamma(n)} \notin \tilde{T}\right) \cdot \mathbf{Pr}\left(X_{\gamma(n)} \notin \tilde{T}\right)\\
&\leq \mathcal{L}(s_0) + (1+h_{\max}) \textstyle\sum_{n\in\mathbb{N}} \mathbf{Pr}\left(X_{\gamma(n)} \notin \tilde{T}\right)\\
&= \mathcal{L}(s_0) + (1+h_{\max}) \textstyle\sum_{n\in\mathbb{N}} \sum_{0\leq j<\ell_{\max}} \mathbf{Pr}\left(X_{\gamma(n\ell_{\max}+j)} \notin \tilde{T}\right)\\
&\leq \mathcal{L}(s_0) + (1+h_{\max})\ell_{\max} \textstyle\sum_{n\in\mathbb{N}} \mathbf{Pr}\left(X_{\gamma(n\ell_{\max})} \notin \tilde{T}\right)\\
&\leq \mathcal{L}(s_0) + (1+h_{\max})\ell_{\max} \textstyle\sum_{n\in\mathbb{N}} (1-p_{\min})^n < \infty
\end{aligned}$$

◀

B Details on the implementation presented in Section 5

B.1 Data-structure for exact summation

Algorithm 1 heavily relies on the capacity to accurately sum probabilities of very different magnitudes a large number of times. Indeed in early versions of the implementation, we have observed that without refined dedicated summation algorithms, the program does not terminate. Some methods exist to improve the accuracy of summation like Kahan summation algorithms [11] but are not sufficient in our setting. So we propose a data structure with better accuracy when summing up positive values, at the cost of increased memory consumption and time.

We present our data structure in the context of values in the interval $[0, 1]$, which is sufficient for probabilities. It encodes such a value r as a table of 512 integers (each encoded on 64 bits) such that the content each cell $c[i]$ represents a floating point value (float) with the content of the cell being the mantissa and the index of the cell i being the exponent. The value is encoded as the sum of the floats encoded by the cells, i.e. $r = \sum_{i\leq 512} c[i]2^{-i}$.

The addition of a float x to a table T encoding a number is done as follows (more details in [3]). The float is broken down into its exponent and mantissa i.e., $x = m2^{-u}$ with $m \leq 2^{52}$ and $1 \leq u \leq 512$. There are two cases. First if $T[u] = 0$ then $T[u]$ is set to m. Otherwise the float stored at index u is built ($y = T[u]2^{-u}$) and x and y are added. If this sum is at most 2^{52}, then it is stored in $T[u]$; otherwise there is an overflow, and $T[u]$ is assigned $x + y - 2^{52}$, while the procedure is applied recursively with exponent $u + 1$. In the worst case there are recursive calls over the whole range of T.

B.2 Heap with update

Algorithm 1 requires a data structure storing the set of states that have been visited with the probability and likelihood of the path reaching them. This data structure maps states to two real numbers representing the probability and the likelihood of the state. It requires to support three operations:

- insertion of a new mapping $s \mapsto (p, l)$ of a state to its probability and likelihood in the data structure;
- removing and retrieving the mapping with maximal probability;
- given a state s updating the probability and likelihood of this state.

Such a data structure can be implemented with a heap and a hash table, which points to the node in the heap allowing update. All operations are performed in logarithmic time w.r.t. to the number of elements in the data structure.

B.3 Implementation of the numerical algorithm for decisive Markov chains

Algorithm 1 can be specialized to likelihood functions that are "monoidal" functions of the path, in the sense that paths leading to the same state can be merged while only retaining the probability to reach it together with a weighted average (over the paths) of the likelihood. Furthermore, using a heap where states are ordered by their probability to be reached (and merging them) represents a large improvement. The data structure for the heap is described in section B.2. One can show that the heap management policy is fair, and that the corresponding algorithm terminates on decisive Markov chains (see [3]).

Plan Logic

Dylan Bellier
Univ Rennes, IRISA, CNRS, France

Massimo Benerecetti
Università degli Studi di Napoli Federico II, Italy

Fabio Mogavero
Università degli Studi di Napoli Federico II, Italy

Sophie Pinchinat
Univ Rennes, IRISA, CNRS, France

Abstract

When reasoning about games, one is often interested in verifying more intricate strategic properties than the mere existence of a winning strategy for a given coalition. Several languages, among which the very expressive *Strategy Logic* (SL), have been proposed that explicitly quantify over strategies in order to express and verify such properties. However, quantifying over strategies poses serious issues: not only does this lead to a non-elementary model-checking problem, but the classic Tarskian semantics is not fully adequate, both from a conceptual and practical viewpoint, since it does not guarantee the realisability of the strategies involved.

In this paper, we follow a different approach and introduce *Plan Logic* (PL), a logic that takes *plans*, *i.e.*, sequences of actions, as first-class citizens instead of strategies. Since plans are much simpler objects than strategies, it becomes easier to enforce realisability. In this setting, we can recover strategic reasoning by means of a compositional *hyperteams semantics*, inspired by the well-known *team semantics*. We show that the *Conjunctive-Goal* and *Disjunctive-Goal fragments* of SL are captured by PL, with an effective polynomial translation. This result relies on the definition of a suitable game-theoretic semantics for the two fragments. We also investigate the model-checking problem for PL. For the full prenex fragment, the problem is shown to be fixed-parameter-tractable: it is polynomial in the size of the model, when the formula is fixed, and 2-ExpTime-complete in the size of the formula. For the *Conjunctive-Goal* and *Disjunctive-Goal fragments* of PL this result can be improved to match the optimal polynomial complexity in the size of the model, regardless of the size of the formula.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Modal and temporal logics; Theory of computation → Complexity theory and logic

Keywords and phrases Logic for strategic reasoning, Strategy Logic, Realisable strategies, Strategies *vs.* plans, Hyperteam semantics

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.9

1 Introduction

When reasoning about games, one is often interested in verifying strategic properties involving the players participating in a game. The simplest such property asks whether one of the players is able to win the game, possibly under specific conditions, regardless of what the other players do. This corresponds to checking whether that player has a winning strategy, namely a set of rules, ideally in the form of a procedure or a function, stipulating how the player must choose its moves in each situation or position during the game in order to achieve the goal corresponding to its winning condition. This has led to the development of a number of logical languages specifically tailored to allow for expressing temporal properties that can predicate, more or less explicitly, over strategies. Notable examples are *Alternating Temporal Logic* (ATL*) [2, 3, 19] and *Strategy Logic* (SL) [7, 18, 8, 16, 17]. In its general form, a

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 9; pp. 9:1–9:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

strategy can be viewed as a function that maps histories, *i.e.*, finite sequences of observables encoding what players have seen up to the current situation in the game, to moves that the player following that strategy has to perform in the current position. In SL notation, for instance, one would express the property that player $\mathtt{a}$ can win a game against some other player, say $\mathtt{b}$, by means of the following first-order-like sentence $\exists \textsc{x}.\, \forall \textsc{y}.\, (\mathtt{a}, \textsc{x})(\mathtt{b}, \textsc{y})\psi$, where variables $\textsc{x}$ and $\textsc{y}$ range over strategies, $(\mathtt{a}, \textsc{x})$ and $(\mathtt{b}, \textsc{y})$ bind each player to a specific strategy, and ψ is an LTL formula encoding $\mathtt{a}$'s winning condition. The sentence can be read as follows: there exists a strategy $\textsc{x}$ such that, for all strategies $\textsc{y}$, if player $\mathtt{a}$ follows $\textsc{x}$ and $\mathtt{b}$ follows $\textsc{y}$, then the objective ψ is achieved. The separation of strategy quantifications and bindings is a distinctive feature of SL and allows for comparing different strategies for multiple objectives, called goals, each corresponding to a binding of agents and variables followed by an LTL formula. This is what provides the logic with the ability to directly express complex strategic properties, such as, for instance, the existence of Nash equilibria [7, 18].

Ideally, once we know that an objective in a game is achievable, we would like to be able to synthesise the existentially quantified strategies that witness the possibility of achieving that goal. This, in turn, would provide a concrete way to obtain a solution to the game by means of logical reasoning. However, quantification over strategies may lead to situations where a formula can be satisfied only if the witness strategies are built with full knowledge of the strategies of the opponents. What this means is that, in order to synthesise such strategies, one may need to know what the other strategies prescribe in the future or even in counterfactual situations, that is along histories different from the one actually followed. Satisfaction of the sentence $\forall \textsc{y}.\, \exists \textsc{z}.\, (\mathtt{a}, \textsc{z})(\mathtt{b}, \textsc{y})\psi$, for instance, boils down to the existence of a Skolem function f such that the purely universal sentence $\forall \textsc{y}.\, (\mathtt{a}, \mathsf{f}(\textsc{y}))(\mathtt{b}, \textsc{y})\psi$ is satisfied. Function f essentially encodes the mechanism that allows one to build the required witness strategy. However, the input to f is a full-fledged strategy, namely a tree-like object that dictates a response to every history of the game. As a consequence, the response $\mathsf{f}(\sigma)(\varpi)$ of the strategy $\mathsf{f}(\sigma)$ on a given history ϖ may well depend on what the input strategy σ dictates on histories different from ϖ. Such information is, however, usually not available while playing the game, as neither future nor counterfactual situations have been encountered by the players. In many cases, the dependency of the Skolem function on these situations is actually not necessary to satisfy the formula and, in those cases, one can prove that a Skolem function exists that indeed does not. When the only Skolem functions that provide satisfaction of a formula do depend on future or counterfactual situations, we say that the corresponding witness existential strategies are unrealisable. For instance, this is the case for the multi-goal sentence $\forall \textsc{y}.\, \exists \textsc{z}.\, ((\mathtt{a}, \textsc{y})\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (\mathtt{a}, \textsc{z})\mathtt{X}\, p)$, requiring that, for any strategy $\textsc{y}$, there exists a response strategy $\textsc{z}$ which, when followed by the same agent a, ensures in the next step the same literal granted by $\textsc{y}$ two steps ahead. On structures where all positions have successors for each literal, that sentence can clearly only be satisfied by non-realisable strategies. The phenomenon where a sentence turns out to be satisfied only with unrealisable witnesses is referred to as *non-behavioural satisfaction* in the literature [16, 11]. Not only does this allow for satisfiable sentences for which no concrete and effective mechanisms can be implemented that synthesise the corresponding winning strategies for the game, it has also been shown to be the main source of complexity in strategic reasoning [16, 5], often leading to non-elementary procedures for the decision problems.

Interestingly enough, there are fragments of SL that are not intrinsically affected by the problem. More specifically, it was shown in [16] that every formula in the One-Goal fragment of SL is *behaviourally satisfiable*, meaning that if it is satisfiable then there exists a realisable Skolem function that, along each history, only needs to look at that history to choose the next move. This result was later generalised in [11, 12], where a new semantics for SL, called *timeline semantics*, was proposed and the maximal fragment SL[EG], based on the semantic notion of semi-stability, was identified that enjoys the realisability property. These results try to overcome the problem of non-behavioural satisfaction by identifying well-behaved sets of formulae, for which focusing only on the observations of the current history is enough to decide satisfaction. One may argue that, while reasoning about games, those are the only formulae we are really interested in when we want to figure out how to enforce the objectives.

Based on the above observations, we propose here a logic, called *Plan Logic* (PL), for which, by design, no such problem can arise and which can nonetheless express most of the relevant game-theoretic strategic properties. In addition, the truth of all such properties can be checked in doubly-exponential time at most, with the additional guarantee that any satisfied sentence is realisable in the sense discussed above. This is achieved by taking *plans*, namely infinite sequence of moves, as primary objects for the domain of quantification instead of strategies. Plans are much simpler objects compared to strategies, as they have an intrinsically linear nature. Strategies, by contrast, exhibit a branching nature, as they must take care of a player's behaviours along all possible histories, which, taken all together, form a tree-like structure. In a sense, strategies can be viewed as adaptive plans that may react differently depending on the context and the same strategy can also be used for different goals. Such a strategy would prescribe the same choices for two goals as long as they are indistinguishable to the players, that is as long as the histories along the corresponding plays for the two goals coincide, still allowing for different behaviours when the two goals can be distinguished. This feature is not a native one for plans though. Hence, in order to enforce the same behaviour in indistinguishable contexts, we allow for plans to be tied together by means of specific operators. Essentially, as long as two goals are indistinguishable for the players, two tied plans are required to prescribe the same actions, exactly like a single strategy would do.

The linear nature of plans, on the other hand, makes it much easier to enforce realisability. For one, dependence on counterfactual futures becomes a non-issue, since each plan dictates the moves an agent has to take along a single history and different goals would use distinct, though possibly tied, plans. In order to ensure that the choices of a plan do not depend on the future choices of other plans along the same history, we simply need to impose suitable restrictions on the possible dependencies between the quantified variables at the semantic level. Capturing such constraints requires a semantics able to meaningfully express functional dependencies among quantified variables and, at the same time, retain the determinacy of the logic, meaning that each sentence is either true or false in a given structure. To this end, we employ the *Alternating Hodges' semantics*, a semantics based on *hyperteams*, namely sets of sets of variable assignments, in a similar vein to what has been done for QPTL in [5].

Besides the design of PL and the corresponding compositional hyperteam semantics, our contribution is manifold. We provide a polynomial translation of the *Conjunctive-Goal* and *Disjunctive-Goal fragments* of SL under timeline semantics into PL, whose spirit consists in simulating strategy variables by means of several suitably-tied plan variables. The soundness of this translation (Theorem 7) deeply relies upon the introduction of a game-theoretic semantics for these fragments (Theorem 8), which, to our knowledge, has never been proposed in the literature. The result also shows that each Boolean connective, taken in isolation, exhibits a game-theoretic behaviour. In addition, we study the model-checking problem for PL, taking inspiration from the introduced game-theoretic approach. We prove that, for the *Boolean-Goal fragment* of PL, the problem is 2-EXPTIME-COMPLETE in the length of the formula and fixed-parameter-tractable in the size of the model, once the maximum number of bindings is fixed in the formula (Theorem 14). This is the first result with an elementary complexity of the entire Boolean-Goal fragment of a logic for strategic reasoning, in stark contrast with the tower-complete complexity of Boolean-Goal SL under standard semantics [6]. Incidentally, note that no model-checking procedure exists for Boolean-Goal SL under timeline semantics [11, 12]. By leveraging the similarity between the game-theoretic semantics of the Conjunctive and Disjunctive-Goal fragments of both PL and SL, we improve the model-checking complexity of those fragments to PTIME-COMPLETE in the size of the model (Theorem 16).

In light of all these results, we argue that plans not only appear to be a powerful alternative to strategies, but they may also be preferable in terms of adequacy, as most of the difficulties and annoyances that come into play when dealing with strategies do not affect plans.

2 Preliminaries

We denote by Σ^∞ (*resp.*, Σ^*, Σ^+, Σ^ω) the set of (*resp.*, *finite*, *non-empty finite*, *infinite*) *sequences* w over the alphabet Σ, with *length* $|w| \in \mathbb{N} \cup \{\infty\}$. Given $n < |w|$, the *element at* n of w is denoted by $(w)_n$, while its *prefix up to* n by $(w)_{\leq n}$. Two sequences $w, u \in \Sigma^\infty$ are *equal up to* $n \in \mathbb{N}$, in symbols $w =_{\leq n} u$, if $w = u$ or $n < \mathsf{min}\{|w|, |u|\}$ and $(w)_{\leq n} = (u)_{\leq n}$. This equivalence relation lifts to partial functions $\mathsf{f}, \mathsf{g} \colon \mathrm{Z} \rightharpoonup \mathrm{X}^\infty$ on an arbitrary domain Z as follows: $\mathsf{f} =_{\leq n} \mathsf{g}$ if $\mathsf{dom}(\mathsf{f}) = \mathsf{dom}(\mathsf{g})$ and $\mathsf{f}(z) =_{\leq n} \mathsf{g}(z)$, for all $z \in \mathsf{dom}(\mathsf{f})$.

A *concurrent game structure* (CGS, for short) *w.r.t.* an *a priori* fixed countably-infinite set of *atomic propositions* AP is a structure $\mathfrak{G} \triangleq \langle \mathrm{Ag}, \mathrm{Ac}, \mathrm{Ps}, v_I, \delta, \lambda \rangle$, where Ag is a finite non-empty set of *agents*, Ac and Ps are countable non-empty sets of *actions* and *positions*, $v_I \in \mathrm{Ps}$ is an *initial position*, $\delta \colon \mathrm{Ps} \times \mathrm{Ac}^{\mathrm{Ag}} \to \mathrm{Ps}$ is a *transition function* mapping every position $v \in \mathrm{Ps}$ and *action profile* $\vec{c} \in \mathrm{Ac}^{\mathrm{Ag}}$ to a position $\delta(v, \vec{c}) \in \mathrm{Ps}$, and, finally, $\lambda \colon \mathrm{Ps} \to 2^{\mathrm{AP}}$ is a *labelling function* mapping every position $v \in \mathrm{Ps}$ to the finite set of atomic propositions $\lambda(v) \subset_{\texttt{fin}} \mathrm{AP}$ true at that position. The size of $\mathfrak{G}$ is the number of its positions, *i.e.*, $|\mathfrak{G}| \triangleq |\mathrm{Ps}|$. By abuse of notation, $\delta \subseteq \mathrm{Ps} \times \mathrm{Ps}$ also denotes the *transition relation* between positions such that $(v, u) \in \delta$ *iff* $\delta(v, \vec{c}) = u$, for some $\vec{c} \in \mathrm{Ac}^{\mathrm{Ag}}$. A *path* $\pi \in \mathrm{Pth} \subseteq \mathrm{Ps}^\infty \setminus \{\varepsilon\}$ is a sequence of positions compatible with the transition function and beginning with the initial position, *i.e.*, $(\pi)_0 = v_I$ and $((\pi)_i, (\pi)_{i+1}) \in \delta$, for each $0 \leq i < |\pi| - 1$. The labelling function lifts from positions to paths in the usual way: $\lambda \colon \mathrm{Pth} \to (2^{\mathrm{AP}})^+$. A *history* is a finite path $\varpi \in \mathrm{Hst} \triangleq \mathrm{Pth} \cap \mathrm{Ps}^+$, while a *play* $\pi \in \mathrm{Play} \triangleq \mathrm{Pth} \cap \mathrm{Ps}^\omega$ is an infinite one. A *strategy* is a function $\sigma \in \mathrm{Str} \triangleq \mathrm{Hst} \to \mathrm{Ac}$ mapping every history $\varpi \in \mathrm{Hst}$ to an action $\sigma(\varpi) \in \mathrm{Ac}$. A play $\pi \in \mathrm{Play}$ is *compatible* with a *strategy profile* $\vec{\sigma} \in \mathrm{Str}^{\mathrm{Ag}}$ if, for all $i \in \mathbb{N}$, it holds that $(\pi)_{i+1} = \delta((\pi)_i, \vec{c}_i)$, where $\vec{c}_i \in \mathrm{Ac}^{\mathrm{Ag}}$ is the action profile with $\vec{c}_i(a) = \vec{\sigma}(a)((\pi)_{\leq i})$, for all agents $a \in \mathrm{Ag}$. The function $\mathsf{play} \colon \mathrm{Str}^{\mathrm{Ag}} \to \mathrm{Play}$ assigns to each profile $\vec{\sigma} \in \mathrm{Str}^{\mathrm{Ag}}$ the unique play $\mathsf{play}(\vec{\sigma}) \in \mathrm{Play}$ compatible with $\vec{\sigma}$; we also say that $\vec{\sigma}$ *induces* $\mathsf{play}(\vec{\sigma})$.

3 A Logic of Plans

As opposed to existing logics for strategic reasoning, such as ATL* [2] and SL [7, 18], where the (implicit or explicit) domain of quantification is composed of strategies, quite complex objects, we introduce *Plan Logic* that relies on the much simpler notion of *plan*. Plans are infinite sequences $\rho \in \mathrm{Pln} \triangleq \mathrm{Ac}^\omega$ that describes the course of actions an agent chooses to execute in response to what the other agents already decided to do.

From a syntactic standpoint, Plan Logic bears a strong similarity with SL. In particular, PL extends LTL by allowing (i) to quantify explicitly over plans, (ii) to assign plans to agents by means of a binding mechanism similar to the one of SL that connects agents and plan variables, and (iii) to form bundles of plan variables via *tying operations* that are crucial to correlate different plans as parts of essentially the same strategy in the game model.

Syntax. Throughout this work, we implicitly assume an *a priori* fixed countably-infinite set of *variables* Vr. A *binding* $\flat \in \mathrm{Bn} \triangleq \mathrm{Vr}^{\mathrm{Ag}}$ is a function mapping every agent $a \in \mathrm{Ag}$ to a variable $\flat(a) \in \mathrm{Vr}$, commonly represented as a finite sequence of binding pairs $(a_1, x_1), \ldots, (a_k, x_k)$, where each agent occurs exactly once. By $\mathsf{vr}(\flat) \subset \mathrm{Vr}$ we denote the set of variables occurring in $\flat$ and lift the notation to sets of bindings as the union of the corresponding sets element-wise.

For simplicity, the syntax of the full logic forces formulae to be *flat*, as in the flat fragments of CTL* [9] and ATL* [13], where sentences can be combined in a Boolean way, but cannot be nested. Notice that this flatness constraint comes *w.l.o.g.*, when the model-checking problem is considered, as the latter can always be reduced to reasoning about flat formulae via a relabelling of the underlying structure (see [15, 3], for details).

▶ **Definition 1.** *Plan Logic (*PL*, for short) is the set of formulae built according to the following context-free grammar, where* $\flat \in \mathrm{Bn}$*,* $\psi \in \mathrm{LTL}$*,* $\mathrm{V} \subset_{\mathtt{fin}} \mathrm{Vr}$*, and* $x \in \mathrm{Vr}$*:*

$$\varphi \coloneqq \flat\psi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle \mathrm{V} \rangle\, \varphi \mid [\mathrm{V}]\, \varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi.$$

We shall denote by $\mathsf{free}(\varphi) \subseteq \mathsf{vr}(\varphi) \subset \mathrm{Vr}$ the sets of free variables and variables occurring in φ. Specifically, $\mathsf{free}(\flat\psi) \triangleq \mathsf{vr}(\flat)$ and $\mathsf{free}(\langle \mathrm{V} \rangle\, \varphi) = \mathsf{free}([\mathrm{V}]\, \varphi) \triangleq \mathrm{V} \cup \mathsf{free}(\varphi)$; all other cases are as usual. A *sentence* φ is a formula without free variables, *i.e.*, $\mathsf{free}(\varphi) = \emptyset$. Similarly, $\mathsf{bnd}(\varphi) \subset \mathrm{Bn}$ denotes the set of bindings occurring in φ.

The binding $\flat$ in a PL *goal* $\flat\psi$ have basically the same interpretation as in SL, namely as the mechanism that associates agents with the content of variables, plans in our case, against which LTL formulae can be evaluated, once the corresponding play is determined. Quantifiers and tying operators, on the other hand, need some explaining in game-theoretic terms. Since we are interested in realisability, we require that the plans we quantify over must be effectively computable, namely that each action chosen at some instant can only depend on the past choices of all the quantified plans. This allows us to view plans as branches of the tree representations of strategies. With this view in mind, the quantifier $\exists x$ (*resp.*, $\forall x$) can be read as "*there exists a realisable plan ...*" (*resp.*, "*for all realisable plans ...*"). Tying operators, instead, are precisely the mechanism that connects plans to strategies in the following sense. Different plan variables denote branches of the same strategy, as long as they provide the same choices for any two bindings that share the same history. The operator $\langle \mathrm{V} \rangle$ (*resp.*, $[\mathrm{V}]$) can then be read as "*the plans in* V *are part of a strategy and ...*" (*resp.*, "*if the plans associated with* V *are part of a strategy then ...*"). Essentially, the two operators filter out sets of plans that cannot be part of the same strategy, because they prescribe different actions for the same history. In a sense, these operators play the role of strategic constructs, implicitly quantifying existentially and universally over strategies via their component plans.

To better understand these intuitions, let us discuss some examples of SL formulae and their corresponding PL equivalents. The simple SL sentence $\Phi_{\mathtt{W}} = \exists \mathrm{X}.\, \forall \mathrm{Y}.\, (a, \mathrm{X})(b, \mathrm{Y})\psi$ states that an agent a can win a two-player game with LTL objective ψ. Specifically, it requires the existence of a strategy X whose induced plays, each one induced by some strategy Y of the adversary b, satisfy ψ. This same property would be expressed in PL by the sentence $\varphi_{\mathtt{W}} = \exists x.\, \forall y.\, \langle x \rangle\, [y](a, x)(b, y)\psi$, which states that there exists a realisable plan followed by a that is part of some strategy, *e.g.*, the witness strategy for X of the SL sentence, and ensures the objective, regardless of the realisable plans y that are part of possible strategies $\textsc{y}$ followed by the adversary. Note that the realisability requirement for plans is crucial here, since it means that their actions must be chosen on-the-fly only with knowledge of the past history, in order to mimic the behaviour of strategies.

For a second example, let us consider the property claiming the existence of a strategy for some objective ψ that is not strictly dominated by any other strategy. This is expressed by the SL sentence $\Phi_{\texttt{NSD}} = \exists \textsc{x}.\, \forall \textsc{x}'.\, \exists \textsc{y}.\, ((a, \textsc{x}')(b, \textsc{y})\psi \rightarrow (a, \textsc{x})(b, \textsc{y})\psi)$. The formula asserts that, for some strategy $\textsc{x}$ and any other strategy $\textsc{x}'$, both for the same agent a, there is at least one strategy $\textsc{y}$ for the opponent such that following $\textsc{x}'$ instead of $\textsc{x}$ would not give a a better outcome. In PL terms, that property is captured by the sentence $\varphi_{\texttt{NSD}} = \exists x.\, \forall x'.\, \exists y_1, y_2.\, \langle x \rangle$ $[x'] \, \langle y_1, y_2 \rangle \, ((a, x')(b, y_1)\psi \rightarrow (a, x)(b, y_2)\psi)$, where we ensure that the two plans y_1 and y_2 are part of the same existentially quantified strategy $\textsc{y}$ for b.

As a final example, consider the existence of a Nash equilibrium for the two agents, a and b, whose objectives are ψ_a and ψ_b, respectively. An SL sentence for this property is $\Phi_{\texttt{NE}} = \exists \textsc{x}.\, \exists \textsc{y}.\, \forall \textsc{z}.\, (((a, \textsc{z})(b, \textsc{y})\psi_a \rightarrow (a, \textsc{x})(b, \textsc{y})\psi_a) \wedge ((a, \textsc{x})(b, \textsc{z})\psi_b \rightarrow (a, \textsc{x})(b, \textsc{y})\psi_b))$, where $\textsc{x}$ and $\textsc{y}$ represent the equilibrium strategies. The sentence asserts that neither agent can improve by unilaterally deviating from the profile, *i.e.*, by deciding to follow any other strategy $\textsc{z}$ instead of $\textsc{x}$ and $\textsc{y}$. The corresponding PL sentence is

$$\varphi_{\texttt{NE}} = \exists x_1, x_2.\, \exists y_1, y_2.\, \forall z_1, z_2.\, \langle x_1, x_2 \rangle \, \langle y_1, y_2 \rangle \, [z_1, z_2] \begin{pmatrix} ((a, z_1)(b, y_1)\psi_a \rightarrow (a, x_2)(b, y_2)\psi_a) \\ \wedge \\ ((a, x_1)(b, z_2)\psi_b \rightarrow (a, x_2)(b, y_2)\psi_b) \end{pmatrix},$$

where the existential strategies $\textsc{x}$ and $\textsc{y}$ are simulated via the operators $\langle x_1, x_2 \rangle$ and $\langle y_1, y_2 \rangle$ on the pairs of plans x_1, x_2 and y_1, y_2, while the universal strategy $\textsc{z}$ via $[z_1, z_2]$ on z_1, z_2.

The overall intuition underlying the correspondence between SL and PL is that, in order to express a strategic property comprising a given set of different bindings, one really only needs to be able to predicate on a small portion of the strategies involved, namely on a single plan for each binding occurring in the property. This intuition is formally substantiated in Section 4, where a formal translation of some behavioural fragments of SL is provided.

Semantics. The semantics of PL formulae relies on the basic notion of *assignment*, a partial function $\chi \in \mathrm{Asg} \triangleq \mathrm{Vr} \rightharpoonup \mathrm{Pln}$ interpreting variables as plans. We may distinguish assignments defined exactly over some set $\mathrm{V} \subseteq \mathrm{Vr}$, *i.e.*, elements of $\mathrm{Asg}(\mathrm{V}) \triangleq \{\chi \in \mathrm{Asg} \mid \mathsf{dom}(\chi) = \mathrm{V}\}$, and those defined on a superset of V, *i.e.*, elements of $\mathrm{Asg}_{\supseteq}(\mathrm{V}) \triangleq \{\chi \in \mathrm{Asg} \mid \mathrm{V} \subseteq \mathsf{dom}(\chi)\}$. The assignment $\chi[x \mapsto \rho]$ is derived from χ by assigning plan $\rho \in \mathrm{Pln}$ to variable $x \in \mathrm{Vr}$.

To interpret a goal $\flat\psi$ *w.r.t.* an assignment $\chi \in \mathrm{Asg}_{\supseteq}(\mathsf{vr}(\flat))$, one needs to consider the play $\mathsf{play}_{\flat}(\chi)$ that is induced by the *plan profile* $\vec{\rho} \triangleq \chi \circ \flat \in \mathrm{Pln}^{\mathrm{Ag}}$ obtained as the functional composition of χ and $\flat$ and associating a plan in χ with every agent, in accordance with the binding $\flat$. Formally, $\mathsf{play}_{\flat}(\chi)$ is the unique play $\pi \in \mathrm{Play}$ such that $(\pi)_{i+1} = \delta((\pi)_i, \vec{c}_i)$, for all $i \in \mathbb{N}$, where $\vec{c}_i \in \mathrm{Ac}^{\mathrm{Ag}}$ is the action profile associating with each agent $a \in \mathrm{Ag}$ the action stipulated at time i by the plan assigned to a in the plan profile $\vec{\rho}$, *i.e.*, $\vec{c}_i(a) = (\vec{\rho}(a))_i$.

The semantics of the tying operators $\langle \mathrm{V} \rangle$ and $[\mathrm{V}]$ requires some intermediate notions. Two bindings $\flat_1, \flat_2 \in \mathrm{Bn}$ *agree up to* $n \in \mathbb{N}$ *on* an assignment $\chi \in \mathrm{Asg}$ if $\mathsf{play}_{\flat_1}(\widehat{\chi}) =_{\leq n} \mathsf{play}_{\flat_2}(\widehat{\chi})$, for some extension $\chi \subseteq \widehat{\chi} \in \mathrm{Asg}_{\supseteq}(\mathsf{vr}(\{\flat_1, \flat_2\}))$. Intuitively, $\flat_1$ and $\flat_2$ agree up to n on χ if two corresponding plan profiles induce the same history ϖ of length $n + 1$, where n evolution steps have occurred since the initial position. Note that $\flat_1$ and $\flat_2$ agree up to 0 on every assignment, since the initial position is always a common history of length 1. For an assignment $\chi \in \mathrm{Asg}$, two variables $x_1, x_2 \in \mathsf{dom}(\chi)$, and two bindings $\flat_1, \flat_2 \in \mathrm{B}$, with $x_1 \in \mathsf{vr}(\flat_1)$ and $x_2 \in \mathsf{vr}(\flat_2)$, we say that the pair (x_1, x_2) is $(\flat_1, \flat_2)$-*tied in* χ when, for every $n \in \mathbb{N}$, if $\flat_1, \flat_2$ agree up to n on χ then $\chi(x_1) =_{\leq n} \chi(x_2)$. This condition ensures the existence of a strategy σ such that the actions $(\chi(x_1))_n$ and $(\chi(x_2))_n$, at every instant of time n, coincide with the action $\sigma(\varpi)$, for some n-evolution-step history ϖ. We lift the notion to sets of variables $\mathrm{V} \subseteq \mathsf{dom}(\chi)$ and bindings $\mathrm{B} \subseteq \mathrm{Bn}$ as follows: V is B-*tied in* χ if (x_1, x_2) is $(\flat_1, \flat_2)$-tied in χ, for all $x_1, x_2 \in \mathrm{V}$ and $\flat_1, \flat_2 \in \mathrm{B}$, with $x_1 \in \mathsf{vr}(\flat_1)$ and $x_2 \in \mathsf{vr}(\flat_2)$.

A Tarskian semantics for PL, *à la* SL, would be formalised as follows.

▶ **Definition 2.** *For an implicitly given* CGS $\mathfrak{G}$*,* Tarski's semantic relation $\chi \models \varphi$ *for* PL *is inductively defined as follows, for all* PL *formulae* φ *and assignments* $\chi \in \mathrm{Asg}_{\supseteq}(\mathsf{free}(\varphi))$*.*

1. $\chi \models \flat\psi$*, if* $\lambda(\mathsf{play}_\flat(\chi)) \models_{\mathrm{LTL}} \psi$*;*
2. *the semantics of Boolean connectives is defined as usual;*
3. $\chi \models \langle \mathrm{V} \rangle\, \varphi$*, if* $\chi \models \varphi$ *and* V *is* $\mathsf{bnd}(\varphi)$*-tied in* χ*;*
4. $\chi \models [\mathrm{V}]\, \varphi$*, if* $\chi \models \varphi$ *when* V *is* $\mathsf{bnd}(\varphi)$*-tied in* χ*;*
5. $\chi \models \exists x.\, \phi$*, if* $\chi[x \mapsto \rho] \models \phi$*, for some plan* $\rho \in \mathrm{Pln}$*;*
6. $\chi \models \forall x.\, \phi$*, if* $\chi[x \mapsto \rho] \models \phi$*, for all plans* $\rho \in \mathrm{Pln}$.

The meaning of all conditions above should be self-evident. In particular, Item 3 requires, besides the satisfaction of the formula φ, that the set of variables V be tied in the assignment *w.r.t.* the entire set of bindings $\mathsf{bnd}(\varphi)$ occurring in φ, thus ensuring the existence of a strategy containing the plans associated with V. Item 4 just expresses the dual condition, witnessing the equivalence between $\neg \langle \mathrm{V} \rangle\, \varphi$ and $[\mathrm{V}]\, \neg\varphi$.

Despite its simplicity, the treatment of plan quantifiers in this semantics does not correctly capture the effective computability requirement for the plans discussed above. To see why, consider again the non-behaviourally satisfiable SL sentence given in the introduction: $\Phi_{\mathtt{NB}} \triangleq \forall \mathrm{Y}.\, \exists \mathrm{Z}.\, ((\mathsf{a}, \mathrm{Y}) \mathtt{X}\, \mathtt{X}\, p \leftrightarrow (\mathsf{a}, \mathrm{Z}) \mathtt{X}\, p)$. The corresponding PL translation, obtained similarly to the previous examples, can be, indeed, shown satisfiable under the Tarskian semantics as follows.

▶ **Example 3.** Consider the sentence $\varphi_{\mathtt{NB}} = \forall y.\, \exists z.\, [y]\, \langle z \rangle ((a, y) \mathtt{X}\mathtt{X}\, p \leftrightarrow (a, z) \mathtt{X}\, p)$ and the single-agent two-action two-position CGS $\mathfrak{G} = \langle \{a\}, \{0, 1\}, \{v_0, v_1\}, v_0, \delta, \lambda \rangle$, where (i) action 0 always leads to v_0 and action 1 always to v_1, regardless of the current position, *i.e.*, $\delta(v_i, \{a \mapsto j\}) = v_j$, and (ii) position v_1 is the only one labelled by p, *i.e.*, $\lambda = \{v_0 \mapsto \emptyset, v_1 \mapsto \{p\}\}$. Being a sentence, we evaluate $\varphi_{\mathtt{NB}}$ against the empty assignment $\varnothing$. By applying Items 6 and 5 of Definition 2, we obtain $\mathfrak{G}, \varnothing \models \varphi_{\mathtt{NB}}$ *iff*, for every plan ρ_y, there exists a plan ρ_z such that $\mathfrak{G}, \{y \mapsto \rho_y, z \mapsto \rho_z\} \models [y]\, \langle z \rangle\, ((a, y) \mathtt{X}\, \mathtt{X}\, p \leftrightarrow (a, z) \mathtt{X}\, p)$. Now, by Items 4 and 3, it is immediate to see that the two tying operators $[y]$ and $\langle z \rangle$ do not affect the reasoning, since a singleton set of variables is always trivially tied, no matter which assignment or set of bindings is taken into account. Thus, $\mathfrak{G}, \{y \mapsto \rho_y, z \mapsto \rho_z\} \models [y]\, \langle z \rangle\, ((a, y) \mathtt{X}\, \mathtt{X}\, p \leftrightarrow (a, z) \mathtt{X}\, p)$ *iff* $\mathfrak{G}, \{y \mapsto \rho_y, z \mapsto \rho_z\} \models (a, y) \mathtt{X}\, \mathtt{X}\, p \leftrightarrow (a, z) \mathtt{X}\, p$. At this point, one can simply choose $\rho_z \triangleq (\rho_y)_1 \cdot 0^\omega$ to satisfy the formula. Hence, following the naive interpretation of $\varphi_{\mathtt{NB}}$ via Tarski's semantics, it holds that $\mathfrak{G}$ satisfies $\varphi_{\mathtt{NB}}$ in a non-realisable way, since ρ_z requires knowledge of ρ_y one step ahead.

This example clearly shows that a precise formalisation of game-theoretic plan quantifications cannot be achieved by following a first-order Tarskian approach, due to treatment of plans as monolithic entities. To adequately model plans both as realisable objects and linear components of strategies, we are, indeed, faced with a challenge. We need to ensure that, when a plan is chosen by a quantifier, the selection of the action provided by that plan at each time instant can only depend on the choices made by the other plans so far during the play. This means that the choice must be made with knowledge of the past, but no knowledge of the future. Not only does this requirement guarantee the realisability of the plans, which is one of the main concerns of this work, but it also makes plans compatible with strategies, where the choices of actions are functionally dependent only on the histories. To overcome this challenge, we resort to a semantic framework recently proposed in [5] precisely to handle behavioural functional dependencies among quantified variables. *Alternating Hodges' semantics* is a compositional formulation of the interpretation of formulae with a distinctive game-theoretic flavour involving two players: Eloise, who wishes to prove the formula true, and Abelard, who tries to disprove it. The underlying idea is that the interpretations of the free variables of a formula φ correspond to the choices that the two players have made prior to the current stage of evaluation of φ. These possible choices are recorded in a two-level structure, called *hyperteam*, which is a set of sets of assignments or, in team-semantics terminology [20], a set of *teams*. Each level summarises the information about the choices a given player can make in its turns. To evaluate φ, then, one player chooses a team, while the opponent chooses one assignment in that team. We shall use a flag $\alpha \in \{\exists\forall, \forall\exists\}$, called *alternation flag*, to keep track of which player is assigned to which level of choice, together with two corresponding satisfaction relations, $\models^{\exists\forall}$ and $\models^{\forall\exists}$, for the evaluation. If $\alpha = \exists\forall$, Eloise chooses the team, while Abelard chooses one of the contained assignments, which must satisfy φ; if $\alpha = \forall\exists$, the dual reasoning applies. Given a flag $\alpha \in \{\exists\forall, \forall\exists\}$, we denote by $\overline{\alpha}$ the dual flag, *i.e.*, $\overline{\alpha} \in \{\exists\forall, \forall\exists\}$ with $\overline{\alpha} \neq \alpha$. For the sake of space, we refer to [5, 4] for a detailed analysis of this semantic framework, for the proofs of classic model-theoretic properties, *e.g.*, De Morgan laws, the closure under *positive normal form*, and for further discussions and explanations.

First-order quantifiers $\mathsf{Q}x$ are dealt with by means of the notion of *response function*, a refined version of Skolem function, namely a map $\mathsf{F} \in \mathrm{Rsp} \subseteq \mathrm{Asg} \to \mathrm{Pln}$ from assignments to plans such that if $\chi_1 =_{\leq n} \chi_2$ then $\mathsf{F}(\chi_1) =_{\leq n} \mathsf{F}(\chi_2)$, for all $\chi_1, \chi_2 \in \mathrm{Asg}$ and $n \in \mathbb{N}$. Intuitively, at each time instant n, the action $(\mathsf{F}(\chi))_n$ of the chosen plan $\mathsf{F}(\chi)$ only depends on the actions $(\chi(x))_t$ of the plan $\chi(x)$ at the time instants $t \leq n$, for each variable $x \in \mathsf{dom}(\chi)$. This obviously means that $(\mathsf{F}(\chi))_n$ is independent of $(\chi(x))_t$ at any future instant $t > n$. This corresponds precisely to the notion of behavioural functor in [5] and captures the realisability constraint on plans discussed earlier. For an assignment $\chi \in \mathrm{Asg}$ and a variable $x \in \mathrm{Vr}$, the F-*extension with* x of χ is the assignment $\mathsf{ext}(\chi, \mathsf{F}, x) \triangleq \chi[x \mapsto \mathsf{F}(\chi)]$.

Similar to [14, 20], a *team* $\mathrm{X} \in \mathrm{Tm} \triangleq \{\mathrm{X} \subseteq \mathrm{Asg}(\mathrm{V}) \mid \mathrm{V} \subseteq \mathrm{Vr}\}$ is a set of assignments on the same domain. Teams defined over some prescribed $\mathrm{V} \subseteq \mathrm{Vr}$ and those defined at least over V are grouped in $\mathrm{Tm}(\mathrm{V}) \triangleq \{\mathrm{X} \in \mathrm{Tm} \mid \mathrm{X} \subseteq \mathrm{Asg}(\mathrm{V})\}$ and $\mathrm{Tm}_{\supseteq}(\mathrm{V}) \triangleq \left\{\mathrm{X} \in \mathrm{Tm} \,\middle|\, \mathrm{X} \subseteq \mathrm{Asg}_{\supseteq}(\mathrm{V})\right\}$. The set of variables on which all the assignments inside a team $\mathrm{X} \in \mathrm{Tm}$ are defined is denoted by $\mathsf{vr}(\mathrm{X})$. The team $\{\varnothing\}$ only containing the empty assignment $\varnothing$ is called the *trivial team*. A set of variables $\mathrm{V} \subseteq \mathsf{vr}(\mathrm{X})$ is B-*tied in* X, for a set of bindings $\mathrm{B} \subseteq \mathrm{Bn}$, if V is B-tied in every assignment $\chi \in \mathrm{X}$. For a response function $\mathsf{F} \in \mathrm{Rsp}$ and a variable $x \in \mathrm{Vr}$, the notion of F-*extension with* x lifts from assignments to teams as follows: $\mathsf{ext}(\mathrm{X}, \mathsf{F}, x) \triangleq \{\mathsf{ext}(\chi, \mathsf{F}, x) \mid \chi \in \mathrm{X}\}$.

As defined in [5, 4], a *hyperteam* $\mathfrak{X} \in \mathrm{HTm} \triangleq \{\mathfrak{X} \subseteq \mathrm{Tm}(\mathrm{V}) \mid \mathrm{V} \subseteq \mathrm{Vr}\}$ is a set of teams. Hyperteams defined over some given $\mathrm{V} \subseteq \mathrm{Vr}$ and those defined at least over V are grouped in $\mathrm{HTm}(\mathrm{V}) \triangleq \{\mathfrak{X} \in \mathrm{HTm} \mid \mathfrak{X} \subseteq \mathrm{Tm}(\mathrm{V})\}$ and $\mathrm{HTm}_{\supseteq}(\mathrm{V}) \triangleq \{\mathfrak{X} \in \mathrm{HTm} \mid \mathfrak{X} \subseteq \mathrm{Tm}_{\supseteq}(\mathrm{V})\}$. The set of variables shared by all teams inside a hyperteam $\mathfrak{X} \in \mathrm{HTm}$ is denoted by $\mathsf{vr}(\mathfrak{X})$. The hyperteam $\{\{\varnothing\}\}$ comprised only of the trivial team is called the *trivial hyperteam*.

The semantics of PL is based on four operations on hyperteams, that take care of the various logical operators. The *partitioning* $\mathsf{par}(\mathfrak{X}) \triangleq \{(\mathfrak{X}_1, \mathfrak{X}_2) \in 2^{\mathfrak{X}} \times 2^{\mathfrak{X}} \mid \mathfrak{X}_1 \uplus \mathfrak{X}_2 = \mathfrak{X}\}$ handles the Boolean connectives, by reducing the evaluation of the entire formula *w.r.t.* $\mathfrak{X}$ to the evaluation of its Boolean components *w.r.t.* disjoint parts $\mathfrak{X}_1$ and $\mathfrak{X}_2$ of $\mathfrak{X}$. The *filtering* $\mathsf{flt}(\mathfrak{X}, \mathrm{V}, \mathrm{B}) \triangleq \{\mathrm{X} \in \mathfrak{X} \mid \mathrm{V} \text{ is B-tied in } \mathrm{X}\}$ *w.r.t.* the sets of variables $\mathrm{V} \subseteq \mathsf{vr}(\mathrm{X})$ and bindings $\mathrm{B} \subseteq \mathrm{Bn}$ deals with the tying operators, by filtering out of $\mathfrak{X}$ all teams X in which V is not B-tied. The *extension* $\mathsf{ext}(\mathfrak{X}, x) \triangleq \{\mathsf{ext}(\mathrm{X}, \mathsf{F}, x) \mid \mathrm{X} \in \mathfrak{X} \text{ and } \mathsf{F} \in \mathrm{Rsp}\}$ *w.r.t.* the variable $x \in \mathrm{Vr}$ takes care of the first-order quantifiers, by F-extending with x every team X in $\mathfrak{X}$, for all possible response functions F. Finally, the *dualisation* $\overline{\mathfrak{X}}$ swaps the role of the two players in a hyperteam, allowing for connecting the two satisfaction relations and for a symmetric treatment of all PL constructs. The swap is accomplished via the notion of choice function $\mathsf{\Gamma} \colon \mathfrak{X} \to \mathrm{Asg}$ over a hyperteam $\mathfrak{X}$, which picks a single assignment from each team: $\mathrm{Chc}(\mathfrak{X}) \triangleq \{\mathsf{\Gamma} \colon \mathfrak{X} \to \mathrm{Asg} \mid \mathsf{\Gamma}(\mathrm{X}) \in \mathrm{X}, \text{for each } \mathrm{X} \in \mathfrak{X}\}$. Then, the dualisation builds a new hyperteam, whose teams are obtained by gathering all the assignments chosen by one of the choice functions: $\overline{\mathfrak{X}} \triangleq \{\mathsf{img}(\mathsf{\Gamma}) \mid \mathsf{\Gamma} \in \mathrm{Chc}(\mathfrak{X})\}$. Observe that the trivial hyperteam is self-dual, *i.e.*, $\overline{\{\{\varnothing\}\}} = \{\{\varnothing\}\}$. This approach bears strong similarity with the transformations between DNF and CNF formulae, where a hyperteam can be viewed as a disjunction of conjunctive clauses over assignments, if $\alpha = \forall\exists$, and as a conjunction of disjunctive clauses, if $\alpha = \exists\forall$.

The compositional semantics of PL based on hyperteams can then be defined as follows.

▶ **Definition 4.** *For an implicitly given* CGS $\mathfrak{G}$, Hodges' alternating semantic relation $\mathfrak{X} \models^{\alpha} \varphi$ *for* PL *is inductively defined as follows, for all* PL *formulae* φ, *alternation flags* $\alpha \in \{\exists\forall, \forall\exists\}$, *and hyperteams* $\mathfrak{X} \in \mathrm{HTm}_{\supseteq}(\mathsf{free}(\varphi))$*:*

1. $\mathfrak{X} \models^{\exists\forall} \flat\psi$, *if there exists* $\mathrm{X} \in \mathfrak{X}$ *such that* $\lambda(\mathsf{play}_{\flat}(\chi)) \models_{\mathrm{LTL}} \psi$, *for all* $\chi \in \mathrm{X}$*;*
2. $\mathfrak{X} \models^{\alpha} \neg\varphi$, *if* $\mathfrak{X} \not\models^{\overline{\alpha}} \varphi$*;*
3. $\mathfrak{X} \models^{\exists\forall} \varphi_1 \wedge \varphi_2$, *if* $\mathfrak{X}_1 \models^{\exists\forall} \varphi_1$ *or* $\mathfrak{X}_2 \models^{\exists\forall} \varphi_2$, *for all* $(\mathfrak{X}_1, \mathfrak{X}_2) \in \mathsf{par}(\mathfrak{X})$*;*
4. $\mathfrak{X} \models^{\forall\exists} \varphi_1 \vee \varphi_2$, *if* $\mathfrak{X}_1 \models^{\forall\exists} \varphi_1$ *and* $\mathfrak{X}_2 \models^{\forall\exists} \varphi_2$, *for some* $(\mathfrak{X}_1, \mathfrak{X}_2) \in \mathsf{par}(\mathfrak{X})$*;*
5. $\mathfrak{X} \models^{\exists\forall} \langle \mathrm{V} \rangle \varphi$, *if* $\mathsf{flt}(\mathfrak{X}, \mathrm{V}, \mathsf{bnd}(\varphi)) \models^{\exists\forall} \varphi$*;*
6. $\mathfrak{X} \models^{\forall\exists} [\mathrm{V}] \varphi$, *if* $\mathsf{flt}(\mathfrak{X}, \mathrm{V}, \mathsf{bnd}(\varphi)) \models^{\forall\exists} \varphi$*;*
7. $\mathfrak{X} \models^{\exists\forall} \exists x. \varphi$, *if* $\mathsf{ext}(\mathfrak{X}, x) \models^{\exists\forall} \varphi$*;*
8. $\mathfrak{X} \models^{\forall\exists} \forall x. \varphi$, *if* $\mathsf{ext}(\mathfrak{X}, x) \models^{\forall\exists} \varphi$*;*
9. $\mathfrak{X} \models^{\alpha} \varphi$, *if* $\overline{\mathfrak{X}} \models^{\overline{\alpha}} \varphi$, *for all other cases.*

The base case (Item 1) for the goals $\flat\psi$ formalises the intuition for satisfaction relative to the flag $\exists\forall$: there exists a team X in $\mathfrak{X}$, all assignments χ of which induce a play $\mathsf{play}_{\flat}(\chi)$ with a labelling that satisfy the LTL formula ψ. One could equivalently define the semantics for the dual flag $\forall\exists$: for all $\mathrm{X} \in \mathfrak{X}$, it holds that $\lambda(\mathsf{play}_{\flat}(\chi)) \models_{\mathrm{LTL}} \psi$, for some $\chi \in \mathrm{X}$. The choice here is immaterial, thanks to the dualisation rule of Item 9. Negation, in accordance with the game-theoretic interpretation, is dealt with in Item 2 by swapping the players associated with the two internal levels of the hyperteam. The semantics of the Boolean connectives (Items 3 and 4), tying operators (Items 5 and 6), and first-order quantifiers (Items 7 and 8) relies on the first three hyperteam operations discussed above. Finally, the semantics for all the remaining cases reduce, thanks to Item 9, to one of the cases presented, after dualising both the hyperteam and the alternation flag. It is immediate to observe that, for a fixed CGS $\mathfrak{G}$, the truth value of a PL sentence φ, when evaluated *w.r.t.* the trivial hyperteam, does not depend on the specific flag, *i.e.*, $\{\{\varnothing\}\} \models^{\exists\forall} \varphi$ *iff* $\{\{\varnothing\}\} \models^{\forall\exists} \varphi$, due to the self duality of $\{\{\varnothing\}\}$. We shall thus write $\mathfrak{G} \models_{\mathrm{PL}} \varphi$ to assert both $\{\{\varnothing\}\} \models^{\exists\forall} \varphi$ and $\{\{\varnothing\}\} \models^{\forall\exists} \varphi$.

The following result, whose proof is a trivial adaptation of the corresponding one in [5], shows that, when no quantifiers are present, the hyperteam semantics bears a natural correspondence with the Tarskian one.

▶ **Theorem 5.** *For all* PL *quantifier-free formulae* φ *and hyperteams* $\mathfrak{X} \in \mathrm{HTm}_{\supseteq}(\mathsf{free}(\varphi))$*:*

1. $\mathfrak{X} \models^{\exists\forall} \varphi$ iff *there exists* $\mathrm{X} \in \mathfrak{X}$ *such that* $\chi \models \varphi$, *for all* $\chi \in \mathrm{X}$*;*
2. $\mathfrak{X} \models^{\forall\exists} \varphi$ iff, *for all* $\mathrm{X} \in \mathfrak{X}$, *it holds that* $\chi \models \varphi$, *for some* $\chi \in \mathrm{X}$.

We can now show that, under the hyperteam semantics, the non-behavioural property reported in the introduction is, as expected, no more satisfiable.

▶ **Example 6.** Consider again the sentence $\varphi_{\mathtt{NB}} = \forall y.\, \exists z.\, [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p)$ and the CGS $\mathfrak{G}$ of Example 3. We want to show that $\mathfrak{G} \not\models_{\mathrm{PL}} \varphi_{\mathtt{NB}}$, meaning that $\varphi_{\mathtt{NB}}$ is not behaviourally satisfiable on $\mathfrak{G}$, *i.e.*, there is no realisable plan for z ensuring a match of the truth values of p at time instants 1 and 2. Since $\mathsf{free}(\varphi_{\mathtt{NB}}) = \emptyset$, we evaluate $\varphi_{\mathtt{NB}}$ against the trivial hyperteam $\{\{\varnothing\}\}$, which, as observed before, implies that the alternation flag is of no consequence. *W.l.o.g.*, we choose $\alpha = \forall\exists$, thus focusing on proving $\{\{\varnothing\}\} \not\models^{\forall\exists} \varphi_{\mathtt{NB}}$.

The rule for the universal quantifier $\forall y$ (Item 8) requires to compute the extension $\mathfrak{X} \triangleq \mathsf{ext}(\{\{\varnothing\}\}, y) = \{\{y\,{:}\,000^\omega\}, \{y\,{:}\,010^\omega\}, \{y\,{:}\,100^\omega\}, \{y\,{:}\,110^\omega\}, \ldots\}$ of $\{\{\varnothing\}\}$, containing a singleton team for each one of the uncountably many plans to assign to y. This results in

$$\{\{\varnothing\}\} \models^{\forall\exists} \forall y.\, \exists z.\, [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p) \;\; \textit{iff} \;\; \mathfrak{X} \models^{\forall\exists} \exists z.\, [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p).$$

To apply the rule for the existential quantifier $\exists z$ (Item 7), we first need to dualise the hyperteam and switch to the $\exists\forall$ flag (Item 9). Since every team of $\mathfrak{X}$ is a singleton set, there is only one possible choice function for it, thus, the result is

$$\mathfrak{X} \models^{\forall\exists} \exists z.\, [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p) \;\; \textit{iff} \;\; \overline{\mathfrak{X}} \models^{\exists\forall} \exists z.\, [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p),$$

where $\overline{\mathfrak{X}} = \{\{y\,{:}\,000^\omega, y\,{:}\,010^\omega, y\,{:}\,100^\omega, y\,{:}\,110^\omega, \ldots\}\}$ is the singleton hyperteam composed of the unique team containing all plans for y. The quantifier $\exists z$ and the alternation flag $\exists\forall$ are coherent, so we can proceed extending the hyperteam to obtain $\mathfrak{X}' \triangleq \mathsf{ext}\big(\overline{\mathfrak{X}}, z\big)$. The result is

$$\overline{\mathfrak{X}} \models^{\exists\forall} \exists z.\, [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p) \;\; \textit{iff} \;\; \mathfrak{X}' \models^{\exists\forall} [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p),$$

where $\left\{\left\{\begin{matrix} y\,{:}\,000^\omega \\ z\,{:}\,00^\omega \end{matrix}, \begin{matrix} y\,{:}\,010^\omega \\ z\,{:}\,00^\omega \end{matrix}, \begin{matrix} y\,{:}\,100^\omega \\ z\,{:}\,00^\omega \end{matrix}, \begin{matrix} y\,{:}\,110^\omega \\ z\,{:}\,00^\omega \end{matrix}, \ldots\right\}, \left\{\begin{matrix} y\,{:}\,000^\omega \\ z\,{:}\,10^\omega \end{matrix}, \begin{matrix} y\,{:}\,010^\omega \\ z\,{:}\,10^\omega \end{matrix}, \begin{matrix} y\,{:}\,100^\omega \\ z\,{:}\,00^\omega \end{matrix}, \begin{matrix} y\,{:}\,110^\omega \\ z\,{:}\,00^\omega \end{matrix}, \ldots\right\}, \ldots\right\}$ is the hyperteam $\mathfrak{X}'$ containing one team $\mathsf{ext}(\mathrm{X}, \mathsf{F}, z)$ for every response function $\mathsf{F} \in \mathrm{Rsp}$, where $\mathrm{X} = \{y\,{:}\,000^\omega, y\,{:}\,010^\omega, y\,{:}\,100^\omega, y\,{:}\,110^\omega, \ldots\}$ is the unique team in $\overline{\mathfrak{X}}$. For instance, the first team in $\mathfrak{X}'$ is obtained by applying the constant function $\mathsf{F}(\chi) = 0^\omega$, while, for the second one, we use the time-0-flip function $\mathsf{F}(\chi) = (1 - (\chi(y))_0) \cdot 0^\omega$. In general, by the behavioural restriction, the action $(\mathsf{F}(\chi))_0$ may only depend on the action $(\chi(y))_0$. Hence, every team X' of $\mathfrak{X}'$ contains at least one assignment χ such that $(\chi(y))_1 \neq (\chi(z))_0$, which implies that $(\mathsf{play}_{(a,y)}(\chi))_2 \neq (\mathsf{play}_{(a,z)}(\chi))_1$. Therefore, for all $\mathrm{X}' \in \mathfrak{X}'$, it holds that $\chi \not\models [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p)$, for some $\chi \in \mathrm{X}'$. As a consequence of Item 1 of Theorem 5, it holds that $\mathfrak{X}' \not\models^{\exists\forall} [y]\, \langle z \rangle\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p)$, which, in turn, means that $\{\{\varnothing\}\} \not\models^{\forall\exists} \varphi_{\mathtt{NB}}$ and, so, $\mathfrak{G} \not\models_{\mathrm{PL}} \varphi_{\mathtt{NB}}$, as expected.

4 Adequacy with Strategy Logic under Timeline Semantics

While the PL semantics – thanks to the tying operators – ensures that the strategies involved are also realisable, we have shown for instance that the SL formula $\forall y.\, \exists z.\, ((a, y)\mathtt{X}\,\mathtt{X}\, p \leftrightarrow (a, z)\mathtt{X}\, p)$ involves strategies that are not. As an immediate consequence, the two logics are not directly comparable. Still, as shown in [16], for the *one-goal fragment* of SL, written SL[1G] here, a formula is satisfiable *iff* it is satisfiable when quantifying only over realisable strategies [16]. Moreover, prenex formulae of SL can be given the so-called *timeline semantics* [11, 12] that enforces realisability of the strategies. This semantics relies on the important notion of *maps* – which are objects very close to Skolem functions.

We relate SL with timeline semantics and PL, by showing that the SL *conjunctive goal* and the *disjunctive goal* fragments, respectively denoted by SL[CG] and SL[DG] (their union is written SL[CG/DG]) can be translated into PL. Due to lack of space, we do not recall here the original timeline semantics of SL, instead we introduce a game-theoretic version (whose correctness is established in Theorem 8), that we use to prove the soundness of this translation. It is worth noting that the SL[CG] fragment encompasses the ATL* extension studied in [10].

4.1 Strategy Logic under Timeline Semantics and Plan Logic

Syntax. The timeline semantics of SL is given for the prenex fragment of the language, in which each formula starts with a *quantifier prefix*, namely a finite sequence $\wp$ of existential $\exists \mathrm{x}$ and universal $\forall \mathrm{x}$ quantifiers, where each variable occurs at most once. The set of variables occurring in a quantifier prefix $\wp$ is $\mathsf{vr}(\wp)$, and we let $\mathsf{vr}_{\exists}(\wp)$ (*resp.* $\mathsf{vr}_{\forall}(\wp)$) be the set of existentially (*resp.* universally) quantified variables. In the rest of this section, we implicitly consider SL under the timeline semantics, and thus every SL formula is in prenex form.

Formulae of the fragments SL[CG] and SL[DG] are, respectively, of the form $\wp \bigwedge_{\flat \in B} \flat \psi_{\flat}$ and $\wp \bigvee_{\flat \in B} \flat \psi_{\flat}$, where $\wp$ is a quantifier prefix, B is a set of bindings, and each $\psi_{\flat}$ is an LTL formula. Observe that the one-goal fragment SL[1G] of SL is contained in the intersection of SL[CG] and SL[DG], which amounts to requiring B to be a singleton set. We may use notation SL[BG] to refer to the SL fragment allowing for arbitrary Boolean combinations of goals.

Translation from SL to PL. The translation for the full SL[BG] fragment involves three steps. First we encode each strategy variable with as many plan variables as there are goals in the formula. These plan variables inherit the same quantifier as the original SL variable in the resulting quantifier prefix. Second, to account for the fact that the corresponding plans must be part of the same strategy, we tie such plan variables together by means of a *tying prefix* of suitable tying operators. Third, we replace the strategy variables occurring in the goals of the matrix, *i.e.* the quantifier free subformula following the prefix, with the corresponding plan variable for that goal. More in detail, let $\Phi = \wp\phi$ be a SL[BG] formula. Each quantifier $\mathtt{Q}\mathrm{x}$ in $\wp$ is transformed into a sequence of quantifiers of the form $\mathtt{Q}x_{\flat}$, one for every $\flat \in \mathsf{bnd}(\Phi)$ with $\mathrm{x} \in \mathsf{vr}(\flat)$. Formally, the quantifier prefix of the translation is $\wp_{\mathsf{SL2PL}}(\Phi) \triangleq ((\mathtt{Q}_{\mathrm{x}} x_{\flat})_{\flat \in B_{\mathrm{x}}})_{\mathrm{x} \in \mathsf{vr}(\wp)}$ with $B_{\mathrm{x}} = \{\flat \in \ \mathsf{bnd}(\varphi) \,|\, \mathrm{x} \in \mathsf{vr}(\flat)\}$ and $\mathtt{Q}_{\mathrm{x}} = \exists$ if $\mathrm{x} \in \mathsf{vr}_{\exists}(\wp)$ and $\mathtt{Q}_{\mathrm{x}} = \forall$ if $\mathrm{x} \in \mathsf{vr}_{\forall}(\wp)$.

We now keep track of the fact that the various obtained variables $x_{\flat}$ stem from a single variable x by tying them in a coherent manner via a tying prefix $\tau_{\mathsf{SL2PL}}(\Phi)$: when variable x was existentially (*resp.* universally) quantified, the tying of the $x_{\flat}$'s is existential (*resp.* universal) as follows. Writing $\mathrm{V}_{\mathrm{x}} \triangleq \left\{ x_{\flat} \,\middle|\, \flat \in B_{\mathrm{x}} \right\}$ for the set of plan variables associated with the strategy variable x, we let $\tau_{\mathsf{SL2PL}}(\Phi) \triangleq (\langle \mathrm{V}_{\mathrm{x}} \rangle)_{\mathrm{x} \in \mathsf{vr}_{\exists}(\wp)} ([\mathrm{V}_{\mathrm{x}}])_{\mathrm{x} \in \mathsf{vr}_{\forall}(\wp)}$. Finally, in each original goal subformula $\flat\psi$, we replace every occurrence of variable x with the new variable $x_{\flat}$. The complete translation of the matrix ϕ (a Boolean combination of goals) is denoted by $\phi_{\mathsf{SL2PL}}(\Phi)$. Gathering all the translation components we have defined, we obtain $\mathsf{SL2PL}(\Phi) \triangleq \wp_{\mathsf{SL2PL}}(\Phi)\, \tau_{\mathsf{SL2PL}}(\Phi)\, \phi_{\mathsf{SL2PL}}(\Phi)$, whose size is polynomial in that of Φ. In the next subsection we show that this translation is sound for both the conjunctive and disjunctive goal fragments of SL. We refer the reader to the examples in Section 3 for instances of this translation.

▶ **Theorem 7.** $\mathfrak{G} \models_{\mathrm{SL}} \Phi$ iff $\mathfrak{G} \models \mathsf{SL2PL}(\Phi)$, *for every* SL[CG/DG] *sentence* Φ *and* CGS $\mathfrak{G}$.

The proof of Theorem 7 is reported in Section 4.2. A crucial step in the proof is the introduction of a game-theoretic semantics for SL, which reduces the evaluation of an SL[CG/DG] sentence Φ in a given CGS $\mathfrak{G}$ to the evaluation of a corresponding PL formula $\mathsf{GTS}_{\mathrm{SL}}(\Phi)$ in a modified CGS $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$. This construction turns out to be a game-theoretic semantics for the conjunctive and disjunctive goal fragments of SL.

4.2 Game-theoretic Semantics of SL[CG/DG]

The game-theoretic semantics of SL[CG/DG] employs an additional *operator agent*, who plays the role of the single Boolean operator involved in the quantifier-free matrix of the sentence (either $\wedge$ or $\vee$) and can choose the specific goal formula to be falsified/verified. Essentially, the key idea behind the proposed semantics is that, as long as two bindings follow the same play, the operator agent can postpone the decision of which of the corresponding goal formula to falsify/verify.

Given a CGS $\mathfrak{G} = \langle \mathrm{Ag}, \mathrm{Ac}, \mathrm{Ps}, v_I, \delta, \lambda \rangle$ and an SL[CG/DG] sentence $\Phi = \wp\phi$, we build the new CGS $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$ and the new formula $\mathsf{GTS}_{\mathrm{SL}}(\Phi)$ as follows, where $B_\Phi = \mathsf{bnd}(\Phi)$.

▶ **Construction 1.** *In* CGS $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$*, a position* $\widehat{v} = (v, B)$ *stems from a position* v *in* $\mathfrak{G}$ *equipped with a set* B *of bindings, precisely those that agree so far along the history that led to* $\widehat{v}$*. We set* $\widehat{\mathrm{Ps}} \triangleq \{\widehat{v}_\exists, \widehat{v}_\forall, \widehat{v}_\circledast\} \cup \mathrm{Ps} \times 2^{B_\Phi}$*, where three special sink positions* $\widehat{v}_\exists, \widehat{v}_\forall$ *and* $\widehat{v}_\circledast$ *are explained later. The initial position of* $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$ *is* $\widehat{v_I} = (v_I, B_\Phi)$*, since at the beginning all the bindings agree on the empty history. The set of agents in* $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$ *gathers the* variable agents*, one for each variable quantified in* $\wp$*, and the extra operator agent, written* $x_\circledast$, i.e. $\widehat{\mathrm{Ag}} \triangleq \mathsf{vr}(\Phi) \cup \{x_\circledast\}$*. The actions of* $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$ *include all the actions of the original* CGS $\mathfrak{G}$ *and a new* binding action *for each binding occurring in the original formula* Φ, i.e. $\widehat{\mathrm{Ac}} \triangleq \mathrm{Ac} \cup B_\Phi$*. The variable agents are only allowed to choose an action from the original* CGS *, while binding actions are reserved to agent* $x_\circledast$*, who can only choose a binding belonging to the decoration of the current position. To force each agent to always pick the right type of action, we use the three sink positions* $\widehat{v}_\exists$*,* $\widehat{v}_\forall$ *and* $\widehat{v}_\circledast$*. Specifically, position* $\widehat{v}_\exists$ *(*resp. $\widehat{v}_\forall$*) is reached every time the agent for a universally (*resp. *existentially) quantified variable mischooses a binding action instead of a proper one. Conversely,* $\widehat{v}_\circledast$ *is reached any time agent* $x_\circledast$ *either mischooses a proper action or takes a binding action outside of the decoration of the current position. Formally, we say that an action profile* $\vec{c} \in (\mathrm{Ac} \cup \mathsf{bnd}(\varphi))^{\mathsf{vr}(\wp) \cup \{x_\circledast\}}$ *is* Q-ill-typed*, for* $\mathsf{Q} \in \{\forall, \exists\}$*, if the leftmost variable* x *in the quantifier prefix* $\wp$ *such that* $\vec{c}(x) \notin \mathrm{Ac}$ *is* Q*-quantified, and that* $\vec{c}$ *is* $\circledast$-ill-typed *in position* $\widehat{v} = (v, B)$ *if* $\vec{c}(x) \notin B$*. An action profile is* well-typed *in position* $\widehat{v}$ *if it is neither* Q*-ill-typed nor* $\circledast$*-ill-typed in position* $\widehat{v}$*. The notion of bindings that agree with the choice of* $x_\circledast$ *is formalized as follows. We say that two bindings* $\flat_1, \flat_2 \in \mathrm{Bn}$ *(whose variables are in* $\mathsf{vr}(\wp)$*) are* indistinguishable at position $v \in \mathrm{Ps}$ w.r.t. action assignment $\vec{c} \in \mathrm{Ac}^{\mathsf{vr}(\wp)}$ *of variable agents, in symbols* $\flat_1 \equiv_v^{\vec{c}} \flat_2$*, whenever* $\delta(v, \vec{c} \circ \flat_1) = \delta(v, \vec{c} \circ \flat_2)$, i.e.*, the same position is reached by playing either* $\vec{c} \circ \flat_1$ *or* $\vec{c} \circ \flat_2$*. A move at position* $\widehat{v} = (v, B)$ *with well-typed action profile* $\vec{c}$ *in* $\widehat{v}$ *leads to position* $\widehat{u} = (u, C)$ *where* $u = \delta(v, \vec{c} \circ \flat)$ *for the choice* $\flat = \vec{c}(x_\circledast)$ *of agent* $x_\circledast$*, and* $C \subseteq B$ *retains only the bindings that are indistinguishable from* $\flat$ *(at* v w.r.t. $\vec{c}$*). Formally,*

$$\widehat{\delta}(\widehat{v}, \vec{c}) \triangleq \begin{cases} \widehat{v}_{\mathsf{Q}} & \textit{if } \widehat{v} = \widehat{v}_{\mathsf{Q}}, \textit{ or } \widehat{v} \neq \widehat{v}_\circledast \textit{ and } \vec{c} \textit{ is } \overline{\mathsf{Q}}\textit{-ill-typed, with } \mathsf{Q} \in \{\exists, \forall\}; \\ \widehat{v}_\circledast & \textit{if } \widehat{v} = \widehat{v}_\circledast \textit{ or } \vec{c} \textit{ is } \circledast\textit{-ill-typed in } \widehat{v}; \\ (\delta(v, \vec{c} \circ \flat), \{\flat' \in B \mid \flat' \equiv_v^{\vec{c}} \flat\}) & \textit{with } (v, B) = \widehat{v} \textit{ and } \flat = \vec{c}(x_\circledast), \textit{ otherwise.} \end{cases}$$

Finally, the label of $\widehat{v} = (v, B)$ *inherits from the label of* v *in* $\mathfrak{G}$ *with the extra propositions* $q_\flat$*, one for each binding* $\flat \in B$*. Formally,* $\widehat{\lambda}(\widehat{v}_\exists) \triangleq \{p_\exists\}$*,* $\widehat{\lambda}(\widehat{v}_\forall) \triangleq \{p_\forall\}$*,* $\widehat{\lambda}(\widehat{v}_\circledast) \triangleq \emptyset$*, and* $\widehat{\lambda}((v, B)) \triangleq \lambda(v) \cup \{q_\flat \in \mathrm{AP} \mid \flat \in B\}$*.*

We now turn to the definition of $\mathsf{GTS}_{\mathrm{SL}}(\Phi)$ that is to be evaluated on $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi)$. Since in the construction above variables turned into agents, the involved bindings all collapse to the single identity binding $\flat_{\mathrm{id}}$, *i.e.* $\flat_{\mathrm{id}}(x) = x$ for every $x \in \mathsf{vr}(\wp) \cup \{x_{\circledast}\}$. As a consequence, formula $\mathsf{GTS}_{\mathrm{SL}}(\Phi)$ contains only one goal of the form $\flat_{\mathrm{id}}\psi$, where the definition of ψ depends on whether Φ belongs to SL[CG] or to SL[DG]. Here we illustrate the case $\Phi = \wp \bigwedge_{\flat \in \mathrm{B}} \flat \psi_{\flat} \in$ SL[CG], for which we set:

$$\mathsf{GTS}_{\mathrm{SL}}(\wp \bigwedge_{\flat \in \mathrm{B}} \flat \psi_{\flat}) \triangleq \wp \forall x_{\circledast} \flat_{\mathrm{id}}((\mathtt{F}\, p_{\exists}) \vee ((\mathtt{G}\, \neg p_{\forall}) \wedge \bigwedge_{\flat \in \mathrm{B}} ((\mathtt{G}\, q_{\flat}) \rightarrow \psi_{\flat}))).$$

Intuitively, formula $(\mathtt{F}\, p_{\exists}) \vee ((\mathtt{G}\, \neg p_{\forall}) \wedge \bigwedge_{\flat \in \mathrm{B}} ((\mathtt{G}\, q_{\flat}) \rightarrow \psi_{\flat}))$ gives the win to the existential agents as soon as a universal variable agent makes an ill-typed decision (this is the disjunct $\mathtt{F}\, p_{\exists}$). Otherwise, for the existential variable agents to win, they should never make an ill-typed decision (see the $\mathtt{G}\, \neg p_{\forall}$ subformula) and should guarantee each $\flat$-objective $\psi_{\flat}$ if the obtained play coincides with the original play, namely the one induced by $\flat$ in the original arena; note that in case operator agent chooses an ill-typed action, no such original play exits.

A dual approach holds for the disjunctive case, that results in setting:

$$\mathsf{GTS}_{\mathrm{SL}}(\wp \bigvee_{\flat \in \mathrm{B}} \flat \psi_{\flat}) \triangleq \wp \exists x_{\circledast} \flat_{\mathrm{id}}((\mathtt{G}\, \neg p_{\forall}) \wedge ((\mathtt{F}\, p_{\exists}) \vee \bigvee_{\flat \in \mathrm{B}} ((\mathtt{G}\, q_{\flat}) \wedge \psi_{\flat}))).$$

The following theorem states that the above constructions provide a proper game-theoretic semantics for SL[CG/DG].

▶ **Theorem 8.** $\mathfrak{G} \models_{\mathrm{SL}} \Phi$ iff $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi) \models \mathsf{GTS}_{\mathrm{SL}}(\Phi)$, *for all* SL[CG/DG] *sentences* Φ.

We sketch the proof road-map of Theorem 8 that consists in showing both (a) that the truth of an SL[CG] formula entails the truth of its $\mathsf{GTS}_{\mathrm{SL}}$ translation, and (b) that the truth of an SL[DG] formula entails the truth of its $\mathsf{GTS}_{\mathrm{SL}}$ translation. Observe that the if direction of Theorem 8 follows from (the contrapositions of) items (a) and (b), the determinacy of SL, and the duality of the $\mathsf{GTS}_{\mathrm{SL}}$ constructions for SL[CG] and SL[DG]. Recall that one quantifies over strategies in SL and over plans in PL, the target setting of the game-theoretic semantics. According to the hyperteam semantics of PL, quantifications of plan variables is dealt with by means of responses to variable assignments. What one needs to do, then, is design a correspondence between the strategies of the SL sentence and those responses.

▶ **Theorem 9.** $\mathfrak{G} \models \mathsf{SL_2PL}(\Phi)$ iff $\mathsf{GTS}_{\mathrm{SL}}(\mathfrak{G}, \Phi) \models \mathsf{GTS}_{\mathrm{SL}}(\Phi)$, *for all* SL[CG/DG] *sentences* Φ.

Similarly to the preceding proof approach, we show that (a) the truth of $\mathsf{SL_2PL}(\Phi)$, where $\Phi \in$ SL[CG], entails the truth of its $\mathsf{GTS}_{\mathrm{SL}}$ translation, and (b) the truth of $\mathsf{SL_2PL}(\Phi)$, where $\Phi \in$ SL[DG], entails the truth of its $\mathsf{GTS}_{\mathrm{SL}}$ translation. Notice that both formulae are in PL, but that formula $\mathsf{SL_2PL}(\Phi)$ is based on duplicates $x_{\flat}$'s of the original variables x in Φ, while in $\mathsf{GTS}_{\mathrm{SL}}(\Phi)$ the original variables of Φ are kept as is, with an extra operator agent variable. Reconstructing a response for x from those of the $x_{\flat}$'s is made possible thanks to the tying operators introduced in the formula $\mathsf{SL_2PL}(\Phi)$.

Theorems 8 and 9 entail Theorem 7.

5 Model Checking of Plan Logic

We finally consider the model-checking problem of four fragments of PL, namely PL[BG], PL[CG], PL[DG], and PL[1G], similar to the ones exhibited for SL. Recall that the model-checking problem of PL (and its fragments) is a decision problem that asks whether an input CGS is a model of an input PL sentence. In [6], it has been shown that, due to the non-behaviouralness, *i.e.*, unrealisability, of the Tarskian semantics of the Boolean-Goal fragment of SL (SL[BG]) [16], its model-checking problem is tower complete in the alternation of quantifiers. We prove instead that, despite its high expressive power, PL[BG] enjoys a problem with a 2ExpTime-complete *formula complexity*, which is not harder than the one for the much simpler logic ATL*. This result is obtained by reducing the evaluation of a PL[BG] sentence φ in a given CGS $\mathfrak{G}$ to the evaluation of a PL[1G] sentence $\widehat{\varphi}$ in a modified structure $\widehat{\mathfrak{G}}$. Also, by tuning the reduction for PL[CG] and PL[DG], we obtain a model-checking procedure with an optimal PTime-complete *model complexity.*

Goal Fragments of PL. The *Boolean-Goal fragment* of PL (PL[BG]) comprises all positive Boolean combinations of formulae (in prenex form) $\wp\,\tau\,\phi$, where $\wp$ is a quantifier prefix, τ a tying prefix, and ϕ a positive Boolean combination of goals $\flat\psi$. The *Conjunctive-Goal fragment* of PL (PL[CG]) (*resp.*, *Disjunctive-Goal fragment* of PL (PL[DG])) further restricts PL[BG] by requiring ϕ to be a conjunction (resp, disjunction) of goals. Finally, in the *One-Goal fragment* of PL (PL[1G]), ϕ is assumed to be a single goal $\flat\psi$.

The encoding $\varphi_{\mathtt{NE}}$ of the existence of a Nash equilibrium discussed in Section 3 is an example of PL[BG] formula, as well as the sentence $\varphi_{\mathtt{NB}}$ of Example 3. The sentence $\varphi_{\mathtt{W}}$ stating the existence of a winning strategy in a two player game clearly belongs to PL[1G], while the existence of a non strictly-dominated strategy can be expressed in PL[DG], as witnessed by the encoding $\varphi_{\mathtt{NSD}}$. By turning $\neg\varphi_{\mathtt{NSD}}$ into positive normal form, we obtain the following PL[CG] sentence:

$$\forall x.\,\exists x'.\,\forall y_1, y_2.\,[x]\,\langle x'\rangle\,[y_1, y_2]\,((a, x')(b, y_1)\neg\psi \wedge (a, x)(b, y_2)\neg\psi)\,.$$

In [1], it has been shown that Nash equilibria can actually be expressed in SL[CG]. Thus, the corresponding translations into PL would result in sentences of the PL[CG] fragment. Indeed, the conversion function $\mathsf{SL_2PL}\colon \mathrm{SL} \to \mathrm{PL}$, when applied to an SL[CG/DG] sentence, necessarily returns a PL[CG/DG] one. Finally, $\mathsf{GTS}_{\mathrm{SL}}\colon \mathrm{SL} \to \mathrm{PL}$ always produces a PL[1G] sentence.

The One-Goal Fragment. A simple inspection of the syntactic translation $\mathsf{SL_2PL}\colon \mathrm{SL} \to \mathrm{PL}$ of the previous section shows that its application to an SL[1G] sentence results in a PL[1G] one with the same quantifier prefix, the same goal, and an eliminable prefix of tying operators on a single variable. Actually, a more general elimination property can be proven for arbitrary tying operators in a PL[1G] formula $\wp\tau\flat\psi$: (a) if τ contains $\langle \mathrm{V}\rangle$, with two variables $x, y \in \mathrm{V}$, where y is universally quantified after x in $\wp$, then the subformula originating in $\langle \mathrm{V}\rangle$ is equivalent to $\bot$; (b) dually, if τ contains $[\mathrm{V}]$, with two variables $x, y \in \mathrm{V}$, where y is existentially quantified after x in $\wp$, then the subformula originating in $[\mathrm{V}]$ is equivalent to $\top$; (c) in all other cases, the tying operator can be eliminated, by replacing all the variables in V with the first one of V quantified in $\wp$. *E.g.*, assuming $\wp = \forall x\exists y\forall z$ and $\flat = (a, x)(b, y)(c, y)$, we have that (i) $\wp\,[x, y]\langle y, z\rangle\,\flat\psi \equiv \top$, (ii) $\wp\,\langle y, z\rangle[x, y]\,\flat\psi \equiv \bot$, and (iii) $\wp\,\langle x, y\rangle[y, z]\,\flat\psi \equiv \forall x.\,(a, x)(b, x)(c, x)\psi$. Thus, the following tying-elimination property holds true.

▶ **Proposition 10.** *Every sentence* $\wp\tau\flat\psi$ *in* PL[1G] *has an equivalent sentence of the form* $\wp'\flat'\psi$.

By combining this proposition with Theorem 7, we obtain that the One-Goal fragments of SL and PL semantically coincide.

▶ **Theorem 11.** *For every* $\mathrm{SL}[1\textsc{g}]$ *sentence* Φ*, there is a* $\mathrm{PL}[1\textsc{g}]$ *sentence* φ *and,* vice versa*, for every* $\mathrm{PL}[1\textsc{g}]$ *sentence* φ*, there is an* $\mathrm{SL}[1\textsc{g}]$ *sentence* Φ *such that:* $|\Phi| = \Theta(|\varphi|)$ *and* $\mathfrak{G} \models_{\mathrm{SL}} \Phi$ iff $\mathfrak{G} \models_{\mathrm{PL}} \varphi$*.*

Due to the known $2^{2^{\mathrm{O}(|\varphi|)}} \cdot |\mathfrak{G}|^{\mathrm{O}(1)}$ complexity of the model-checking problem of $\mathrm{SL}[1\textsc{g}]$ [16, Theorem 5.14], we can immediately derive the following theorem.

▶ **Theorem 12.** $\mathrm{PL}[1\textsc{g}]$ *model-checking problem is* 2-ExpTime-complete$(|\varphi|)$ *in the length of the specification* φ *and* PTime-complete$(|\mathfrak{G}|)$ *in the size of the model* $\mathfrak{G}$*.*

The Boolean-Goal Fragment. The encoding underlying Theorem 8 of the game-theoretic semantics for $\mathrm{SL}[\textsc{cg}/\textsc{dg}]$ into $\mathrm{PL}[1\textsc{g}]$ allowed us to prove the equivalence between these logics and the corresponding fragments of PL. We shall leverage the same idea here to solve the model-checking problem of $\mathrm{PL}[\textsc{bg}]$. Given a CGS $\mathfrak{G}$ and a sentence $\varphi = \wp\tau\phi$, with binding set $\mathrm{B}_\varphi \triangleq \mathsf{bnd}(\varphi)$, we build a new CGS $\mathsf{GTS}_{\mathrm{BG}}(\mathfrak{G}, \varphi)$, whose plays are bundles of plays from $\mathfrak{G}$, one per binding in ϕ. This is done, intuitively, by composing in parallel as many copies of $\mathfrak{G}$ as there are bindings in ϕ, resulting in positions that correspond to vectors $\widehat{v} \in \mathrm{Ps}^{\mathrm{B}_\varphi}$ of original positions of $\mathfrak{G}$. Agents of the new game coincide with the variables quantified in $\wp$, while actions carries over unchanged. A move from a position $\widehat{v}$ to a position $\widehat{u}$ is then a vector of parallel moves in $\mathfrak{G}$, one per original position contained in $\widehat{v}$, while forbidding incoherent concurrent choices *w.r.t.* the tying operators occurring in τ. Formally, an action assignment $\vec{c} \in \mathrm{Ac}^{\mathsf{vr}(\wp)}$ is V*-incoherent at* $\widehat{v}$ *w.r.t.* φ, where $\mathrm{V} \subset \mathrm{Vr}$, if there are two variables $x_1, x_2 \in \mathrm{V}$ and two bindings $\flat_1, \flat_2 \in \mathrm{B}_\varphi$, with $x_1 \in \mathsf{vr}(\flat_1)$ and $x_2 \in \mathsf{vr}(\flat_2)$, such that $\widehat{v}(\flat_1) = \widehat{v}(\flat_2)$, but $\vec{c}(x_1) \neq \vec{c}(x_2)$. In other words, a concurrent move $\vec{c}$ is V-incoherent at $\widehat{v}$ *w.r.t.* φ, if there are variables in V whose different associated actions in $\vec{c}$ should have been equal, being part of bindings that are indistinguishable at $\widehat{v}$. We say that $\vec{c}$ is $\exists$*-incoherent* (*resp.*, $\forall$*-incoherent*) *at* $\widehat{v}$ *w.r.t.* φ if the leftmost set of variables V in τ, such that $\vec{c}$ is V-incoherent at $\widehat{v}$ *w.r.t.* φ, occurs in a tying operator of type $\langle V \rangle$ (*resp.*, $[V]$). Intuitively, $\vec{c}$ is $\exists/\forall$-incoherent at $\widehat{v}$ *w.r.t.* φ if the first violated tying constraint specified in τ is existential/universal. If $\vec{c}$ is neither $\exists$-incoherent nor $\forall$-incoherent at $\widehat{v}$ *w.r.t.* φ, we say that $\vec{c}$ is *coherent* at $\widehat{v}$ *w.r.t.* φ.

▶ **Construction 2.** *Given a* CGS $\mathfrak{G} = \langle \mathrm{Ag}, \mathrm{Ac}, \mathrm{Ps}, v_I, \delta, \lambda \rangle$ *and a* $\mathrm{PL}[\textsc{bg}]$ *sentence* φ*, with binding set* $\mathrm{B}_\varphi \triangleq \mathsf{bnd}(\varphi)$*, let* $\mathsf{GTS}_{\mathrm{BG}}(\mathfrak{G}, \varphi) \triangleq \langle \widehat{\mathrm{Ag}}, \widehat{\mathrm{Ac}}, \widehat{\mathrm{Ps}}, \widehat{v}_I, \widehat{\delta}, \widehat{\lambda} \rangle$ *be the* CGS *obtained as follows: (a) agents are the variables quantified in* φ, i.e., $\widehat{\mathrm{Ag}} \triangleq \mathsf{vr}(\varphi)$*; (b)* $\widehat{\mathrm{Ac}} \triangleq \mathrm{Ac}$*; (c) positions are* B_φ*-indexed vectors of original positions from* $\mathfrak{G}$*, plus two distinguished sink positions* $\widehat{v}_\exists$ *and* $\widehat{v}_\forall$, i.e., $\widehat{\mathrm{Ps}} \triangleq \{\widehat{v}_\exists, \widehat{v}_\forall\} \cup \mathrm{Ps}^{\mathrm{B}_\varphi}$*; (d) the initial position is the* v_I*-constant vector,* i.e., $\widehat{v}_I \triangleq \{\flat \in \mathrm{B}_\varphi \mapsto v_I\}$*; (e) every position, but the distinguished ones, are labelled with a set of fresh atomic propositions, one per binding and original labelling,* i.e., $\widehat{\lambda}(\widehat{v}_\exists) \triangleq \{p_\exists\}$, $\widehat{\lambda}(\widehat{v}_\forall) \triangleq \{p_\forall\}$*, and* $\widehat{\lambda}(\widehat{v}) \triangleq \{p_\flat \in \mathrm{AP} \mid \flat \in \mathrm{B}_\varphi, p \in \lambda(\widehat{v}(\flat))\}$*; (f) the transition function* $\widehat{\delta}$ *maps every position* $\widehat{v} \in \widehat{\mathrm{Ps}} \setminus \{\widehat{v}_\exists, \widehat{v}_\forall\}$ *and action profile* $\vec{c} \in \mathrm{Ac}^{\mathsf{vr}(\varphi)}$ *coherent at* $\widehat{v}$ w.r.t. φ *to position* $\widehat{u} \in \widehat{\mathrm{Ps}} \setminus \{\widehat{v}_\exists, \widehat{v}_\forall\}$*, where, for each binding* $\flat \in \mathrm{B}_\varphi$*, the position* $\widehat{u}(\flat)$ *is the successor of* $\widehat{v}(\flat)$ *in* $\mathfrak{G}$ *following the action profile* $\vec{c} \circ \flat$*, which associates with each agent* $a \in \mathrm{Ag}$ *the action stipulated by* $\vec{c}$ *for the variable* $\flat(a)$*; formally,*

$$\widehat{\delta}(\widehat{v}, \vec{c}) \triangleq \begin{cases} \widehat{v}_{\mathtt{Q}}, & \textit{if } \widehat{v} = \widehat{v}_{\mathtt{Q}} \textit{ or } \vec{c} \textit{ is } \overline{\mathtt{Q}}\textit{-incoherent at } \widehat{v} \text{ w.r.t. } \varphi\textit{, with } \mathtt{Q} \in \{\exists, \forall\}; \\ \widehat{u}, & \textit{otherwise, where } \widehat{u}(\flat) \triangleq \delta(\widehat{v}(\flat), \vec{c} \circ \flat)\textit{, for all } \flat \in \mathrm{B}_\varphi. \end{cases}$$

The $\mathrm{PL}[1\textsc{g}]$ encoding of the game-theoretic semantics for the $\mathrm{PL}[\textsc{bg}]$ sentence $\varphi = \wp\tau\phi$ is relatively easy to formalise at this point: besides verifying the coherence constraints dictated by the tying prefix τ, we only need to check that the bundles of plays induced by plans in the CGS $\mathsf{GTS}_{\text{BG}}(\mathfrak{G}, \varphi)$ satisfy the matrix ϕ. Checking the constraints amounts to requiring avoidance of the two distinguished sink positions $\widehat{v}_{\exists}$ and $\widehat{v}_{\forall}$. The verification of the matrix is obtained by transforming ϕ into the LTL formula $\widehat{\phi}$, where each goal $\flat\psi$ is replaced by the LTL formula $\widehat{\psi}$, in turn obtained by replacing in ψ every atomic proposition p with $p_{\flat}$, *i.e.*, $\widehat{\phi} \triangleq \phi\left[\flat\psi/\widehat{\psi}\right]$, with $\widehat{\psi} \triangleq \psi\,[p/p_{\flat}]$. Altogether, we get the following:

$$\mathsf{GTS}_{\text{BG}}(\wp\tau\phi) \triangleq \wp\,\flat_{\text{id}}\left((\mathtt{F}\,p_{\exists}) \vee \left((\mathtt{G}\,\neg p_{\forall}) \wedge \widehat{\phi}\right)\right).$$

Since the original PL[BG] sentence φ and its PL[1G] translation $\mathsf{GTS}_{\text{BG}}(\varphi)$ share the same quantifier prefix $\wp$, thanks to Theorem 5, we can prove the correctness of the encoding, on the basis that $\chi \models \tau\phi$ *iff* $\chi \models \flat_{\text{id}}((\mathtt{F}\,p_{\exists}) \vee ((\mathtt{G}\,\neg p_{\forall}) \rightarrow \widehat{\phi}))$, for all $\chi \in \text{Asg}_{\supseteq}(\mathsf{vr}(\wp))$, which can be done by structural induction on $\tau\phi$ (using the simple semantic rules of Definition 2).

▶ **Theorem 13.** $\mathfrak{G} \models \varphi$ iff $\mathsf{GTS}_{\text{BG}}(\mathfrak{G}, \varphi) \models \mathsf{GTS}_{\text{BG}}(\varphi)$, *for every* PL[BG] *sentence* φ.

Once we observe that $|\mathsf{GTS}_{\text{BG}}(\mathfrak{G}, \varphi)| = 2 + |\mathfrak{G}|^{|\mathsf{bnd}(\varphi)|}$ and $|\mathsf{GTS}_{\text{BG}}(\varphi)| = \text{O}(|\varphi|)$, thanks to Theorem 12, we can derive the following result, where FPT means *fixed-parameter tractable*.

▶ **Theorem 14.** *The model-checking problem for* PL[BG] *is* 2-ExpTime-complete($|\varphi|$) *in the length of the specification* φ *and* $\text{FPT}_{|\varphi|}(|\mathfrak{G}|)$ *in the size of the model* $\mathfrak{G}$*, with the length of the specification* φ *as parameter, once the maximum number of bindings is fixed.*

The Conjunctive & Disjunctive Goal Fragments. The simpler conjunctive/disjunctive nature of goal combinations in PL[CG/DG] allows us to considerably improve on the model complexity of the model-checking problem of PL[BG], by removing redundant information from the position space, which is necessary only to handle arbitrary Boolean combinations. This is done by suitably merging ideas from Constructions 1 and 2: from the former we inherit the structure topology, while of the latter we use the criteria for determining the compliance of the choices *w.r.t.* the tying operators (compliance issues are irrelevant in Construction 1, since strategies are considered). We end-up with an *ad hoc* game-theoretic semantics for PL[CG/DG], whose resulting CGS encoding $\mathsf{GTS}_{\text{CDG}}(\mathfrak{G}, \varphi)$ is virtually identical to Construction 1, where the notion of well-typed action assignment is generalised to take into account the coherence constraints introduced for Construction 2. The sentence encoding $\mathsf{GTS}_{\text{CDG}}(\varphi)$ is also identical to the one used for SL[CG/DG] in association with Construction 1.

The following theorem can be obtained as a slight adaptation of the proof of Theorem 9.

▶ **Theorem 15.** $\mathfrak{G} \models \varphi$ iff $\mathsf{GTS}_{\text{CDG}}(\mathfrak{G}, \varphi) \models \mathsf{GTS}_{\text{CDG}}(\varphi)$, *for every* PL[CG/DG] *sentence* φ.

Once we observe that $|\mathsf{GTS}_{\text{CDG}}(\mathfrak{G}, \varphi)| = 2 + 2^{|\mathsf{bnd}(\varphi)|} \cdot |\mathfrak{G}|$ and $|\mathsf{GTS}_{\text{CDG}}(\varphi)| = \text{O}(|\varphi|)$, we can derive the following result, again thanks to Theorem 12.

▶ **Theorem 16.** *The model-checking problem for* PL[CG/DG] *is* 2-ExpTime-complete($|\varphi|$) *in the length of the specification* φ *and* PTime-complete($|\mathfrak{G}|$) *in the size of the model* $\mathfrak{G}$.

6 Conclusion

We have introduced *Plan Logic* as a language for strategic reasoning, alternative to Strategy Logic, based on the notion of *plans* instead of strategies. We show that this conceptual shift is quite beneficial, as the intrinsic linear nature of plans allows for a semantics that guarantees realisability of the satisfiable sentences via *behavioural functional constraints*. To this end, we propose *hyperteams* as a novel semantic framework for strategic reasoning, which enjoys several important model-theoretic properties, *e.g.*, *compositionality* and *determinacy*. Observe that, for instance, the only semantics for SL that tackle the problem is the timeline semantics proposed in [11, 12], which, however, exhibits neither compositionality nor determinacy. The authors of [12] show, indeed, that such a semantics is not adequate already when applied to SL[BG], as it is undetermined on sentences of that fragment. It is worth noting that the hyperteam semantics, unlike the timeline one based on an *ad hoc* Skolem semantics, is a principled approach that has been applied to model very general functional dependencies among variables in other logics, such as QPTL [5] and FOL [4].

We showed that, thanks to the behavioural nature of the semantics, the model-checking problem of PL[BG] is still 2-EXPTIME-COMPLETE, in stark contrast with the non-elementarity of the same problem for SL[BG] [6]. This further highlights the importance of enforcing behavioural constraints. In addition, we study the *conjunctive and disjunctive goal fragments* of PL in direct comparison with the respective fragments of SL. We show their expressive equivalence, by introducing a novel game-theoretic semantics that allows for a direct comparison between the two logics. Thanks to the connection between the game-theoretic semantics of the PL[CG] and SL[CG], on the one hand, and of PL[DG] and SL[DG], on the other, we can improve the model-checking complexity of those fragments to PTIME-COMPLETE in the size of the model (Theorem 16). Note that these fragments strictly include ATL*, a prominent logic in strategic reasoning, which, in turn, is subsumed by the one-goal fragment of both SL and PL. These fragments are quite interesting, as they enable several forms of complex strategic reasoning, such as strategy domination and various forms of equilibria (*e.g.*, Nash equilibria).

References

1 E. Acar, M. Benerecetti, and F. Mogavero. Satisfiability in Strategy Logic Can Be Easier than Model Checking. In *AAAI'19*, pages 2638–2645. AAAI Press, 2019. doi:10.1609/AAAI.V33I01.33012638.

2 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. In *FOCS'97*, pages 100–109. IEEECS, 1997.

3 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

4 D. Bellier, M. Benerecetti, D. Della Monica, and F. Mogavero. Alternating (In)Dependence-Friendly Logic. *APAL*, 174(10):103315:1–58, 2023.

5 D. Bellier, M. Benerecetti, D. Della Monica, and F. Mogavero. Good-for-Game QPTL: An Alternating Hodges Semantics. *TOCL*, 24(1):4:1–57, 2023. doi:10.1145/3565365.

6 P. Bouyer, P. Gardy, and N. Markey. Weighted Strategy Logic with Boolean Goals Over One-Counter Games. In *FSTTCS'15*, LIPIcs 45, pages 69–83. Leibniz-Zentrum fuer Informatik, 2015.

7 K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. In *CONCUR'07*, LNCS 4703, pages 59–73. Springer, 2007. doi:10.1007/978-3-540-74407-8_5.

8 K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. *IC*, 208(6):677–693, 2010. doi:10.1016/J.IC.2009.07.004.

9 D. Dams. Flat Fragments of CTL and CTL*: Separating the Expressive and Distinguishing Powers. *LJIGPL*, 7(1):55–78, 1999. doi:10.1093/JIGPAL/7.1.55.

10 S. Enqvist and V. Goranko. The Temporal Logic of Coalitional Goal Assignments in Concurrent Multiplayer Games. *TOCL*, 23(4):21:1–58, 2022. doi:10.1145/3517128.

11 P. Gardy, P. Bouyer, and N. Markey. Dependences in Strategy Logic. In *STACS'18*, LIPIcs 96, pages 34:1–15. Leibniz-Zentrum fuer Informatik, 2018.

12 P. Gardy, P. Bouyer, and N. Markey. Dependences in Strategy Logic. *TCS*, 64(3):467–507, 2020. doi:10.1007/S00224-019-09926-Y.

13 V. Goranko and S. Vester. Optimal Decision Procedures for Satisfiability in Fragments of Alternating-time Temporal Logics. In *AIML'14*, pages 234–253. College Publications, 2014. URL: `http://www.aiml.net/volumes/volume10/Goranko-Vester.pdf`.

14 W. Hodges. Compositional Semantics for a Language of Imperfect Information. *LJIGPL*, 5(4):539–563, 1997. `doi:10.1093/JIGPAL/5.4.539`.

15 O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000. `doi:10.1145/333979.333987`.

16 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.

17 F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Satisfiability Problem. *LMCS*, 13(1:9):1–37, 2017.

18 F. Mogavero, A. Murano, and M.Y. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.

19 S. Schewe. ATL* Satisfiability is 2ExpTime-Complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer, 2008. `doi:10.1007/978-3-540-70583-3_31`.

20 J.A. Väänänen. *Dependence Logic: A New Approach to Independence Friendly Logic*, volume 70 of *London Mathematical Society Student Texts*. CUP, 2007.

Explicit Commutative ROABPs from Partial Derivatives

Vishwas Bhargava
Department of Computing and Mathematical Sciences, Caltech, Pasadena, CA, USA

Anamay Tengse
School of Computer Sciences, NISER, Bhubaneswar, India

Abstract

The dimension of partial derivatives (Nisan and Wigderson, 1997) is a popular measure for proving lower bounds in algebraic complexity. It is used to give strong lower bounds on the *Waring decomposition* of polynomials (called *Waring rank*). This naturally leads to an interesting open question: does this measure essentially characterize the Waring rank of any polynomial?

The well-studied model of Read-once Oblivious ABPs (ROABPs for short) lends itself to an interesting hierarchy of "sub-models": Any-Order-ROABPs (ARO), Commutative ROABPs, and Diagonal ROABPs. It follows from previous works that for any polynomial, a bound on its Waring rank implies an analogous bound on its Diagonal ROABP complexity (called the *duality trick*), and a bound on its dimension of partial derivatives implies an analogous bound on its "ARO complexity": ROABP complexity in any order (Nisan, 1991). Our work strengthens the latter connection by showing that a bound on the dimension of partial derivatives in fact implies a bound on the commutative ROABP complexity. Thus, we improve our understanding of partial derivatives and move a step closer towards answering the above question.

Our proof builds on the work of Ramya and Tengse (2022) to show that the *commutative-ROABP-width* of any homogeneous polynomial is at most the dimension of its partial derivatives. The technique itself is a generalization of the proof of the *duality trick* due to Saxena (2008).

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Partial derivatives, Apolar ideals, Commuting matrices, Branching programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.10

Funding *Vishwas Bhargava*: Part of this work was done as a postdoc at University of Waterloo, Canada.
Anamay Tengse: Part of this work was done as a postdoc at Reichman University, Herzliya.

Acknowledgements VB thanks Rafael Oliveira and Abhiroop Sanyal for numerous insightful discussions. AT thanks Prerona Chatterjee, C Ramya, and Ramprasad Saptharishi for several fruitful discussions about ROABPs over the recent years. AT is also deeply grateful to Ramprasad Saptharishi, Susmita Biswas and Lulu, for hosting him during a part of this work.

1 Introduction

How many points do we need to evaluate an expression like the following on, to deterministically tell if it is computing the zero polynomial?

$$f(x_1,\ldots,x_n) = (a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n)^d + \cdots + (a_{s,1}x_1 + \cdots + a_{s,n}x_n)^d \quad (1.1)$$

As of now, the answer to this question stands at $(nds)^{O(\log\log n)}$, just a (rather annoying) smidgen away from a legit efficient algorithm. The above bound follows from a combination of the works of Forbes, Saptharishi and Shpilka [5] and Gurjar, Korwar and Saxena [7].

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 10; pp. 10:1–10:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An expression like (1.1) is called a *Waring decomposition for f* of size s; the name comes from "Waring's problem" in number theory[1]. Analogously, for a homogeneous polynomial $f(\mathbf{x})$ of degree d, its *Waring rank* is the smallest number s for which f can be written as a sum of d-th powers of s-many linear forms; that is, the size of its smallest Waring decomposition. The Waring rank of different polynomials has been studied in mathematics for over a century now (see e.g. [8]), and some recent works have even found its applications in parameterized algorithms (e.g. [16]). It is known that any polynomial has a finite Waring rank [4, 2], except for over finite fields of small characteristic. We now also know the Waring rank of monomials exactly [18]. For example, it is known that the monomial $x_1x_2\cdots x_n$ has Waring rank exactly 2^{n-1}.

The corresponding algebraic model of computation: called a "depth-3-powering circuit", was first introduced in algebraic circuit complexity by Saxena [21], who studied it from the perspective of polynomial identity testing (PIT for short). PIT is the algorithmic task mentioned above: determine whether the given circuit computes the identically zero polynomial. In what is sometimes called a "whitebox PIT", the algorithm has access to the circuit itself; Saxena [21] gave an efficient whitebox test for a more general model. In a "blackbox PIT", the algorithm cannot access the expression and can only query it on a few points (independent of the actual circuit), which is exactly the question stated at the start.

Dimension of partial derivatives

All the currently known blackbox PITs for depth 3 powering circuits build on the fact that any n-variate, degree-d polynomial with Waring rank s has at most $s(d+1)$ *dimension of partial derivatives* (see Definition 1.3). The measure was introduced by Nisan and Wigderson [15] as a tool to prove lower bounds against sums of *products* of linear forms, and thus the above statement is implicit from their work. The myriad variants of this measure now form the basis of several strong lower bounds throughout algebraic circuit complexity (see e.g. [23, 20]).

Returning to Waring decompositions, almost all known lower bounds on Waring ranks of different polynomials use the dimension of partial derivatives in one way or the other. In view of this, and given the strong connections between proofs of hardness and derandomization of PIT (see e.g. [10]), it stands to reason that obtaining an efficient blackbox PIT for depth 3 powering circuits requires us to answer the following question.

▶ **Question 1.1.** *Is it the case that any n-variate polynomial with dimension of partial derivatives r has a Waring rank that is at most* $\mathrm{poly}(n,r)$*?*

To the best of our knowledge, there aren't even any candidate negative examples to this question, except for the symbolic determinant: Det_n. The $n \times n$ determinant has a dimension of partial derivatives that is $2^{\Theta(n)}$, but the best known upper bound on its Waring rank stands at $2^{O(n \log n)}$.

The only other "deviation" that these two measures – dimension of partial derivatives and Waring rank – exhibit, comes from their respective connections with a different well-studied model, which we will now see.

1.1 Read-once Oblivious ABPs (ROABPs)

An ROABP is an expression of the form: $\mathbf{u}^{\intercal} \cdot M_1(x_1) \cdot M_2(x_2) \cdots M_n(x_n) \cdot \mathbf{v}$, where $\mathbf{u}, \mathbf{v}$ are vectors over the base field and each $M_j(x_j)$ is a univariate polynomial with matrices as coefficients, as follows.

[1] See this `wikipedia` article for a summary.

$$M_j(x_j) = A_{j,0} + A_{j,1}x_i + A_{j,2}x_i^2 + \cdots + A_{j,d}x_j^d \tag{1.2}$$

That is, there is exactly one "matrix-polynomial" corresponding to each variable. Thus, each variable is "read" exactly once, oblivious to the other variables; hence the name. We formally define ROABPs in Definition 2.1.

Here, the dimension of $\mathbf{u}$, $\mathbf{v}$ and all the $n(d+1)$ many matrices (assumed to be the same without loss of generality) is said to be the *width* of the ROABP, and is typically denoted by w. Note that the width of an ROABP is the single parameter that dictates its complexity, since n and d arise straight from the polynomial being computed. A subtle point here is that for the same polynomial, the smallest possible ROABP-width can vary widely depending on the order in which the variables appear (see Observation 2.8), and hence the order of an ROABP is also an important parameter. Nevertheless, for any polynomial and any order, the exact size of the smallest corresponding ROABP can be obtained using a characterization given by Nisan [13]. As we will soon see, even this characterization is in a way connected to the partial derivatives of the given polynomial; we provide a formal definition and statement in Definition 2.5 and Theorem 2.6.

ROABPs were formally introduced by Forbes and Shpilka [6] as algebraic analogues of ROBPs from the boolean world, where they showed a quasi-polynomial time blackbox PIT for ROABPs, inspired by Nisan's PRG construction against ROBPs [14]. As the name suggests, ROABPs are a special case of "algebraic branching programs" which are an algebraic analogue of the (boolean) branching programs. However, we omit those definitions of ABPs and ROABPs, as seeing them as the matrix-vector product expressions like above would be more useful for the discussions in this paper. We now introduce the structured variants of ROABPs that are relevant to this work.

1.2 Variants of ROABPs

Since the ROABP-complexity of some polynomials depends heavily on the underlying order, we can further cut out a subclass of polynomials that admit poly(n, d)-sized ROABPs: those that admit poly(n, d)-sized ROABPs *in every order*. This class of polynomials is sometimes referred to as "Any-order ROABPs" (AROs for short)[2].

A *syntactic* way of ensuring that a polynomial computed by an efficient ROABP belongs to ARO, is to ensure that all the $n(d+1)$-many coefficient matrices ($A_{j,*}$s in (1.2)) commute with each other under multiplication. This then means that for any $j, j' \in [n]$, we have that $M_j(x_j)M_{j'}(x_{j'}) = M_{j'}(x_{j'})M_j(x_j)$, and then the layers of the same ROABP can be shuffled to work for any order. Such an ROABP with commuting coefficient matrices is called a *commutative ROABP* (commRO for short); a formal definition is in Definition 2.2.

Finally, an easy way to pick coefficient matrices that commute with each other is to choose all of them as diagonal matrices. Such an ROABP is called a *diagonal ROABP* (diagRO for short), defined in Definition 2.3.

These variants of ROABPs appear implicitly in some previous works on ROABPs, but they were explicitly defined and proposed as objects of study in a recent work of Ramya and Tengse [17]. As mentioned earlier, our proof technique also borrows from the algebraic machinery that appears in their work.

We now proceed to look at the connections between Waring rank, dimension of partial derivatives and these structured ROABPs, before stating our main result.

[2] Contrary to what the name suggests, this is not a special type of ROABPs, it is a class of polynomials.

1.3 Waring rank, partial derivatives and ROABPs

The aforementioned whitebox PIT for depth 3 powering circuits due to Saxena [21] has the following result at its core.

▶ **Theorem 1.2** (Duality trick [21, Lemma 1] (Informal)). *For any linear form $\ell(\mathbf{x}) = a_1x_1 + a_2x_2 + \cdots + a_nx_n$, and any d, the polynomial $\ell(\mathbf{x})^d$ can be expressed as:*

$$\ell(\mathbf{x})^d = \sum_{i=1}^{t} \beta_i \cdot g_{i,1}(x_1) \cdot g_{i,2}(x_2) \cdots g_{i,n}(x_n),$$

for constants $\beta_1, \ldots, \beta_t$ and degree-d univariates $g_{1,1}, \ldots, g_{t,n}$, with $t \leq nd + 1$.

Note that this gives an ROABP of width $t = O(nd)$ for the d-th power of any n-variate linear form, by using the $g_{i,j}$s appropriately to obtain each of the matrix polynomials $M_j(x_j)$ and using β_is in the vector $\mathbf{u}$ (or $\mathbf{v}$). In fact, the "coefficient matrices" (the $A_{j,*}$s from (1.2)) of this ROABP are just diagonal matrices. Consequently, an n-variate, degree-d polynomial with Waring rank r has a diagRO of width $O(ndr)$.

The duality trick actually provides diagROs for a more general model called "depth 4 diagonal circuits", and the corresponding whitebox PIT also holds for this more general model. In fact, this relation between powering circuits and diagROs is a crucial component of the current state-of-the-art blackbox PIT for depth 3 powering circuits [5, 7] mentioned earlier.

Given that polynomials with small Waring rank have small diagROs, it is natural to ask what happens to polynomials with small dimension of partial derivatives.

ROABPs and partial derivatives

Suppose we are given an n-variate, degree-d polynomial $f(\mathbf{x})$, whose dimension of partial derivatives is at most r. It turns out, via Nisan's characterization, that such a polynomial has an ROABP of width at most r *in every order* (see Observation 2.7). That is, for any $\sigma \in s_n$, we are guaranteed *some* ROABP, say $R_\sigma(\mathbf{x})$, that computes f in that order. However, it is not clear from this non-constructive upper bound whether the ROABPs R_σ across different σs are related in any way. This brings us to our main result.

1.4 Our contribution

We first formally define the measure: dimension of partial derivatives.

▶ **Definition 1.3** (Dimension of partial derivatives). *For a polynomial $f(x_1, \ldots, x_n)$, the dimension of its partial derivatives is defined as follows.*

$$\dim \partial^{<\infty}(f) := \dim\left(\operatorname{span}_{\mathbb{C}}\{\partial_{\mathbf{e}} f : \mathbf{e} \in \mathbb{N}^n\}\right)$$

Here $\partial_{\mathbf{e}} f$ denotes the partial derivative of f with respect to the monomial $\mathbf{x}^{\mathbf{e}} = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}$.

For a polynomial f, let $\operatorname{commRO}(f)$ denote the width of the smallest commRO that computes it; our main result is as follows.

▶ **Theorem 1.4.** *For any homogeneous polynomial $f(\mathbf{x}) \in \mathbb{C}[\mathbf{x}]$, $\operatorname{commRO}(f) \leq \dim \partial^{<\infty}(f)$.*

Since the dimension of partial derivatives of any homogeneous component of f is at most $\deg(f)$ times that of f (see Lemma 2.4), we get the following result in the general case.

▶ **Corollary 1.5.** *For any $f(\mathbf{x}) \in \mathbb{C}[\mathbf{x}]$ of degree d, $\operatorname{commRO}(f) \leq (d+1)^2 \cdot \dim \partial^{<\infty}(f)$.*

Set multilinear upper bounds

In fact, our method for constructing commutative ROABPs using $\dim \partial^{<\infty}(f)$ also lets us obtain what we call "commutative set-multilinear ABPs" for f with a minor tweak in our proof. Informally, a degree-d polynomial $f(\mathbf{x})$ is called set-multilinear under a partition $\mathbf{x} = \mathbf{x}_1 \sqcup \mathbf{x}_2 \sqcup \cdots \sqcup \mathbf{x}_d$, if each of its monomials contains exactly one variable from each of the $\mathbf{x}_i$s. A(n ordered) set-multilinear ABP is then a product of matrices with linear polynomials as entries, with the variables in those polynomials obeying the partition (see definitions 4.2 and 4.3).

▶ **Theorem 1.6.** *For any set-multilinear polynomial $f(\mathbf{x}) \in \mathbb{C}[\mathbf{x}]$, the commutative-set-multilinear-ABP-width$(f) \leq \dim \partial^{<\infty}(f)$.*

Explicitness

We note that our proof provides an explicit construction of a commRO for any polynomial f, given the *dependencies* between the partial derivatives of f. In fact, as mentioned earlier, this construction itself is a generalization of the proof of the duality trick from Saxena's work [21] (see Remark 3.6). We describe a width-$2^{O(n)}$ commRO, and a commutative set-multilinear ABP for the $n \times n$ determinant to illustrate this point in Section 4.

2 Preliminaries

Throughout the paper, we work with the field of complex numbers, but most of our proofs extend to fields whose characteristic is zero or large enough.

Notation

- For a vector $\mathbf{e} \in \mathbb{N}^n$, we write $\mathbf{t}^{\mathbf{e}}$ for the monomial $t_1^{e_1} t_2^{e_2} \cdots t_n^{e_n}$, where $\mathbf{t}$ is a set of variables. We also use $\mathbf{e}!$ to refer to the product of factorials $e_1! e_2! \cdot e_n!$.
- For a monomial m, we write $\partial_m f$ for the partial derivative $\frac{\partial^{|\mathbf{e}|} f}{\partial \mathbf{t}^{\mathbf{e}}}$. When $m = \mathbf{t}^{\mathbf{e}}$, we shorten it further to $\partial_{\mathbf{e}} f$.

2.1 Formal definitions

▶ **Definition 2.1** (Read-once Oblivious ABP (ROABP))**.** *For any $n, d, w \in \mathbb{N}$, and an n-variate polynomial $f(\mathbf{x})$ of individual degree d, we say that it has a width w ROABP, if there exists a permutation $\sigma \in s_n$ for which there exist matrices $\{A_{j,k}\}$ in $\mathbb{C}^{w \times w}$ for all $j \in [n]$ and $0 \leq k \leq d$, and vectors $\mathbf{u}, \mathbf{v} \in \mathbb{C}^w$, such that the following holds.*

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{u}^{\intercal} \cdot M_{\sigma(1)}(x_{\sigma(1)}) \cdot M_{\sigma(2)}(x_{\sigma(2)}) \cdots M_{\sigma(n)}(x_{\sigma(n)}) \cdot \mathbf{v}, \\ &\quad \textit{where for all } j \in [n], \\ M_j(x_j) &= A_{j,0} + A_{j,1} x_j + A_{j,2} x_j^2 + \cdots + A_{j,d} x_j^d. \end{aligned}$$

We call the matrices $\{A_{j,k}\}$ the coefficient matrices *of the ROABP.*

▶ **Definition 2.2** (Commutative ROABP (commRO))**.** *An ROABP is said to be a* commutative ROABP*, if all its coefficient matrices commute with each other pairwise.*

For a polynomial f, we use commRO(f) *to denote the smallest width w such that there is width-w commRO computing f.*

▶ **Definition 2.3** (Diagonal ROABP (diagRO)). *An ROABP is said to be a* diagonal ROABP*, if all its coefficient matrices are diagonal matrices.*

For a polynomial f, we use $\operatorname{diagRO}(f)$ *to refer to the smallest width w such that there is width-w diagRO computing f.*

Partial Derivatives and the Nisan matrix

▶ **Lemma 2.4.** *Let $f(\mathbf{x})$ be a polynomial of degree d and let $h(\mathbf{x})$ be some homogeneous component of f. Then $\dim \partial^{<\infty}(h) \leq (d+1) \cdot \dim \partial^{<\infty}(f)$.*

Proof. Note that for any nonzero scalar α, the polynomial $f_\alpha(\mathbf{x}) := f(\alpha x_1, \alpha x_2, \ldots, \alpha x_n)$ satisfies $\dim \partial^{<\infty}(f_\alpha) = \dim \partial^{<\infty}(f)$, since it is an invertible operation. Next, for distinct $\alpha_0, \alpha_1, \ldots, \alpha_d \in \mathbb{C}$, we can use interpolation to write h as a linear combination of $f_{\alpha_0}, f_{\alpha_1}, \ldots, f_{\alpha_d}$. Thus, $\dim \partial^{<\infty}(h) \leq \sum_{0 \leq i \leq d} \dim \partial^{<\infty}(f_{\alpha_i}) \leq (d+1) \cdot \dim \partial^{<\infty}(f)$. ◀

▶ **Definition 2.5** (Nisan Matrix [13]). *For an n-variate polynomial $f(\mathbf{x})$ of individual degree d, and a partition $S \sqcup T = [n]$, the (S,T)*-Nisan matrix for *f, $M^f_{(S,T)}$, is a $(d+1)^{|S|} \times (d+1)^{|T|}$ matrix as follows.*

- *The rows are indexed by all the individual degree d monomials over $\{x_i \mid i \in S\}$,*
- *The columns are indexed by all the individual degree d monomials over $\{x_j \mid j \in T\}$,*
- *The entry $M^f_{(S,T)}[m, m']$ is the coefficient of the monomial $m \cdot m'$ in f.*

▶ **Theorem 2.6** (Nisan's characterization [13]). *For any n-variate polynomial $f(\mathbf{x})$, and any order $\sigma \in s_n$ on the variables, define $S_i = \{\sigma(1), \ldots, \sigma(i)\}$ and $T_i = \{\sigma(i+1), \ldots, \sigma(n)\}$ for each $i \in [n]$.*
Then the size of the smallest ROABP for f in the order σ is exactly $\sum_{i \in [n]} \operatorname{rank}(M^f_{(S_i,T_i)})$.
Further, the width of the ROABP is exactly $\max_{i \in [n]} \operatorname{rank}(M^f_{(S_i,T_i)})$.

It is not difficult to see that for any polynomial, the Nisan matrix for any partition is a scaling of a submatrix of a matrix whose rows are all the partial derivatives of that polynomial. This then leads to the following observation, which is a weaker and non-constructive version of Theorem 1.4.

▶ **Observation 2.7.** *Let $n, d \in \mathbb{N}$ be arbitrary and $\mathbb{F}$ be any field of characteristic 0 or greater than d. Then for any n-variate $f(\mathbf{x})$ of individual degree d, and any partition $S \sqcup T = [n]$, $\operatorname{rank}(M^f_{(S,T)}) \leq \dim \partial^{<\infty}(f)$.*

Thus, any polynomial f has an ROABP in every order of width at most $\dim \partial^{<\infty}(f)$.

▶ **Observation 2.8.** *The polynomial $(x_1 + y_1)(x_2 + y_2) \cdots (x_n + y_n)$ has width-2 ROABPs in the order $(x_1, y_1, \ldots, x_n, y_n)$, but requires width 2^n in the order $(x_1, \ldots, x_n, y_1, \ldots, y_n)$.*

2.2 Concepts from algebra

We will need a few concepts from elementary algebraic geometry; the reader may refer to any standard texts for more details on these concepts (e.g. [3, 8]).

▶ **Definition 2.9** (Ideal). *For a set of polynomials $\{f_1, \ldots, f_s\} \subset \mathbb{C}[\mathbf{x}]$, the ideal generated by them is the smallest set of polynomials I that satisfies the following.*

- *$\forall g \in \mathbb{C}[\mathbf{x}]$ and $\forall f \in I$, $fg \in I$.*
- *$\forall f, f' \in I$, we have $f + f' \in I$.*

The ideal is denoted by $\langle \{f_1, \ldots, f_s\} \rangle$.

▶ **Definition 2.10** (Variety of an ideal). *For an ideal $I \subset \mathbb{C}[\mathbf{x}]$, the* variety of *$I$, written as $\mathbf{V}(I)$, is the largest set of points $V \subset \mathbb{C}^{|\mathbf{x}|}$ such that $\forall f \in I$ and $\forall \mathbf{a} \in V$, $f(\mathbf{a}) = 0$.*

Derivative operators

▶ **Definition 2.11** (Derivative Operator). *A derivative operator is a linear combination of finitely many partial derivatives of the form* $D = \sum_{i=1}^{r} \alpha_i \partial_{m_i}$. *It acts on polynomials naturally:* $Df = \sum_{i=1}^{r} \alpha_i \partial_{m_i} f$.

Clearly, for any polynomial $g = \sum_m g_m m$, *we can define a derivative operator* $D_g = \sum_m g_m \partial_m$, *and vice versa. Therefore, we always refer to a derivative operator as* D_g *with an implicit polynomial* g.

▶ **Definition 2.12** (Closed space of derivative operators). *A vector space of operators* Δ *is said to be closed if for every* $D_g \in \Delta$, *and any monomial* m *such that* $g' := \partial_m g \not\equiv 0$, *the corresponding operator* $D_{g'}$ *is also in* Δ.

▶ **Observation 2.13.** *For any* $f(\mathbf{t}), g(\mathbf{t}) \in \mathbb{C}[\mathbf{t}]$, *we have the following.*

$$(D_f g)(\mathbf{0}) = \sum_{\mathbf{e}} \mathrm{coeff}_f(\mathbf{t}^{\mathbf{e}}) \cdot \mathbf{e}! \cdot \mathrm{coeff}_g(\mathbf{t}^{\mathbf{e}}) = (D_g f)(\mathbf{0})$$

Primary ideals and derivative operators

The following result follows from the joint works of Möller, Marinari and Mora [11], and Möller and Stetter [12]. A proof, with a statement as follows, can be found in [17].

▶ **Theorem 2.14** ([11, 12]). *Let* $J \subseteq \mathbb{C}[\mathbf{t}]$ *be an ideal with the variety* $\mathbf{V}(J) = \{\mathbf{0}\}$, *and suppose that the quotient ring* $R_J := {}^{\mathbb{C}[\mathbf{t}]}/_{J}$ *is a* w*-dimensional vector space over* $\mathbb{C}$. *Then, there exists a* w*-dimensional* $\mathbb{C}$*-vector space of derivative of operators* $\Delta(J)$ *that characterizes the quotient ring* R_J.

That is, for any basis $\{D_1, \ldots, D_w\}$ *of* $\Delta(J)$, *there is an* invertible *matrix* $M \in \mathbb{C}^{w \times w}$ *such that for any polynomial* $g(\mathbf{t}) \in \mathbb{C}[\mathbf{t}]$,

$$[D_1(g)(\mathbf{0}) \;\; D_2(g)(\mathbf{0}) \;\; \cdots \;\; D_w(g)(\mathbf{0})]^{\intercal} = M \cdot \overline{\mathrm{coeff}}([g]),$$

where $\overline{\mathrm{coeff}}([g])$ *is the coefficient vector of* $[g] := (g \bmod J)$.

The above correspondence works for any point $\mathbf{a} \in \mathbb{C}^n$ in the variety; we will work with $\mathbf{0}$ to keep the exposition simple, since that is the only case relevant for our application. We also note that the space of derivative operators $\Delta(J)$ corresponding to an ideal J can equivalently be defined as follows; and there is also a correspondence in the other direction: "from $\Delta(J)$ to J".

▶ **Definition 2.15** (Operator space of an ideal). *For an ideal* $J \subseteq \mathbb{C}[\mathbf{t}]$ *with* $\mathbf{V}(J) = \{\mathbf{0}\}$, *we define the corresponding space of derivative operators* $\Delta(J)$ *as follows.*

$$\Delta(J) := \{D_g \in \mathbb{C}[\partial \mathbf{t}] : \forall h \in J, (D_g h)(\mathbf{0}) = 0\}$$

▶ **Definition 2.16** (Ideal of an operator space). *Let* Δ *be a closed space of derivative operators. We define the corresponding annihilating ideal (at the point* $\mathbf{0}$*), denoted by* $\mathbf{I_0}(\Delta)$ *as follows.*

$$\mathbf{I_0}(\Delta) := \{h \in \mathbb{C}[\mathbf{t}] : \forall D \in \Delta, (Dh)(\mathbf{0}) = 0\}$$

2.3 Multiplication tables: from ideals to matrices

Univariate ideals

Let $J = \langle p(t) \rangle \subseteq \mathbb{C}[t]$ be an ideal, and consider the quotient ring $R := {}^{\mathbb{C}[t]}/_{J}$. If $p(t)$ has degree d, then the *multiplication table* for t in the ring R, is a $d \times d$ matrix whose *minimal polynomial* is $p(t)$. Such a matrix, say A, can easily be defined by setting $A_{i,j} = \text{coeff}_{t^j}\left(\left[t \cdot t^i\right]\right)$, for all $0 \leq i, j \leq (d-1)$. Here, $[t \cdot t^i]$ is $(t^{i+1} \bmod J)$.

For instance, when $p(t) = t^5 - 10t^4 - 7t^3 + 2t^2 - 3$, the multiplication table would be the following 5×5 matrix; it can be checked that $p(t)$ is indeed the minimal polynomial of A.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 3 & 0 & -2 & 7 & 10 \end{bmatrix}$$

Further, A satisfies that $g(A)_{i,j}$ is exactly $\text{coeff}_{t^j}\left(\left[g(t) \cdot t^i\right]\right)$, for any $g(t) \in \mathbb{C}[t]$. In particular, this means that the first row of $g(A)$ is precisely the coefficient vector of $[g(t)]$.

Multivariate ideals

One key change when we move to the multivariate setting, is that there is no inherent ordering on the monomials; so we have to choose one. A *monomial ordering* is any "total order" on monomials, which respects divisions and has 1 as the least monomial. We will work with the degree-wise lexicographical ordering ("deg-lex") with respect to $t_1 \prec t_2 \prec \cdots \prec t_r$. We use this monomial ordering to uniquely identify the leading ("greatest") and trailing ("least") monomials in any polynomial in $\mathbb{C}[t_1, \ldots, t_r]$.

This then allows us to identify a set of leading monomials of the ideal, and then define what is called a normal set and the quotient ring corresponding to the ideal, as follows.

▶ **Definition 2.17** (Leading monomials and normal set). *Given an ideal $I \subset \mathbb{C}[\mathbf{x}]$, and a monomial ordering, the set of its leading monomials is defined as* $\text{LM}(I) := \{\text{LM}(f) \mid f \in I\}$.

The complement of $\text{LM}(I)$ *is called the* normal set of I, *denoted by* $\text{N}(I)$.

▶ **Definition 2.18** (Quotient ring). *For any ideal $I \subset \mathbb{C}[\mathbf{x}]$, and any polynomial $g \in \mathbb{C}[\mathbf{x}]$, we can define $g \bmod I$ to be the polynomial g_0 all of whose monomials are from $\text{N}(I)$ and which satisfies $g - g_0 \in I$. We denote the polynomial $g \bmod I$ by $[g]$ when the ideal is clear from the context.*

We can thus define the quotient ring *corresponding to I denoted by ${}^{\mathbb{C}[\mathbf{x}]}/_{I}$, by reducing each polynomial in $\mathbb{C}[\mathbf{x}]$ modulo I.*

Intuitively, the quotient ring ${}^{\mathbb{C}[\mathbf{t}]}/_{J}$ is obtained by "setting all polynomials in J to zero". Then any monomial from $\text{LM}(J)$ can be written in terms of those in the normal set, and the polynomials in the quotient ring are supported entirely on the monomials from $\text{N}(J)$.

Let the normal set of $J \in \mathbb{C}[t_1, \ldots, t_r]$ be $\{m_1, \ldots, m_w\}$, where $1 = m_1 \prec m_2 \prec \cdots \prec m_w$. The multiplication tables $A_1, A_2, \ldots, A_r$ for J are then the $w \times w$ matrices that satisfy the following, for every $\ell \in [r]$, and all $i, j \in [w]$.

$$A_\ell(i,j) = \text{coeff}_{m_j}([t_\ell \cdot m_i])$$

Just as before, for any polynomial $g(\mathbf{t})$, we have that $g(A_1, \ldots, A_r)(i,j) = \text{coeff}_{m_j}([g \cdot m_i])$. Thus, the first row of any matrix of the form $g(A_1, \ldots, A_r)$ is just the coefficient vector of $g(\mathbf{t}) \bmod J$, for any polynomial g.

3 Constructing commutative ROABPs from apolarities

3.1 Apolar ideal of a polynomial

▶ **Definition 3.1** (Apolar Ideal). *Let $f(t_1, \ldots, t_n)$ be a homogeneous polynomial of degree d. The* apolar ideal *of f is defined as follows.*

$$f^{\perp} := \langle \{h(\mathbf{t}) \in \mathbb{C}[\mathbf{t}] : D_h f \equiv 0\} \rangle$$

▶ **Observation 3.2.** *For any polynomial $f(\mathbf{t})$, the variety of its apolar ideal, $\mathbf{V}(f^{\perp})$, is a single point $\mathbf{0}$.*

Proof. For each $i \in [n]$, $t_i^{d+1} \in f^{\perp}$ where $d = \deg(f)$; so the variety is contained in $\{\mathbf{0}\}$. Also, when $f \not\equiv 0$, any polynomial in $f^{\perp}$ has a zero constant term because a nonzero polynomial cannot be linearly dependent on its derivatives. Hence, $\mathbf{V}(f^{\perp}) = \{\mathbf{0}\}$. ◀

Note that the apolar ideal is a polynomial ideal, but is defined using derivative operators. The apolar ideal of f is related to its partial derivatives in the following way.

▶ **Lemma 3.3** (Apolar ideal and partial derivatives). *Let $f(\mathbf{t})$ be a homogeneous polynomial with $\{g_1, g_2, \ldots, g_w\}$ being a basis for its space of partial derivatives of all degrees. For the corresponding closed space of derivative operators $\Delta_f := \operatorname{span}_{\mathbb{C}} \{D_{g_1}, D_{g_2}, \ldots, D_{g_w}\}$, and its annihilating ideal $J := \mathbf{I_0}(\Delta_f)$, we have that $J = f^{\perp}$ and equivalently, $\Delta_f = \Delta(f^{\perp})$.*

Proof. We can assume that $f = g_1$ without loss of any generality.

$f^{\perp} \subseteq J$. Let $h \in f^{\perp}$, and say $g = \partial_m f$ is some arbitrary partial derivative of f, so $D_g \in \Delta_f$.
Now $(D_g h)(\mathbf{0}) = \sum_{\mathbf{e}} \mathrm{coeff}_h(\mathbf{t}^{\mathbf{e}}) \cdot \mathbf{e}! \cdot \mathrm{coeff}_g(\mathbf{t}^{\mathbf{e}}) = (D_h g)(\mathbf{0})$, from Observation 2.13. Further, $D_h g = D_h \partial_m f = D_{(h \cdot m)} f$, and since $m \cdot h \in f^{\perp}$, $D_g h = D_{(h \cdot m)} f = 0$. Since our choice of g and h was arbitrary, this is true for each $g \in \Delta_f$, and each $h \in J$, showing that $f^{\perp} \subseteq J$.

$J \subseteq f^{\perp}$. Let $h \in J$ be arbitrary, and consider $D_h f = \sum_{\mathbf{e}} \sum_{\mathbf{e}'} \mathrm{coeff}_h(\mathbf{t}^{\mathbf{e}}) \cdot \mathrm{coeff}_f(\mathbf{t}^{\mathbf{e}'}) \cdot \partial_{\mathbf{e}}(\mathbf{t}^{\mathbf{e}'})$.

$$\begin{aligned} D_h f &= \sum_{\mathbf{e}} \sum_{\mathbf{e}' \geq \mathbf{e}} \mathrm{coeff}_h(\mathbf{t}^{\mathbf{e}}) \cdot \mathrm{coeff}_f(\mathbf{t}^{\mathbf{e}'}) \cdot \partial_{\mathbf{e}}(\mathbf{t}^{\mathbf{e}'}) \\ &= \sum_{\mathbf{e}} \sum_{\mathbf{e}' \geq \mathbf{e}} \mathrm{coeff}_h(\mathbf{t}^{\mathbf{e}}) \cdot \mathrm{coeff}_f(\mathbf{t}^{\mathbf{e}'}) \cdot \frac{\mathbf{e}'!}{(\mathbf{e}' - \mathbf{e})!} \cdot (\mathbf{t}^{\mathbf{e}' - \mathbf{e}}) \\ &= \sum_{\mathbf{e}_0 := \mathbf{e}' - \mathbf{e}} \left[\sum_{\mathbf{e}} \mathrm{coeff}_h(\mathbf{t}^{\mathbf{e}}) \cdot (\mathbf{e} + \mathbf{e}_0)! \cdot \mathrm{coeff}_f(\mathbf{t}^{\mathbf{e} + \mathbf{e}_0}) \cdot \left(\frac{\mathbf{t}^{\mathbf{e}_0}}{\mathbf{e}_0!} \right) \right] \end{aligned}$$

Now let $g_0 := \partial_{\mathbf{e}_0}(f)$, and note that $\mathrm{coeff}_{g_0}(\mathbf{t}^{\mathbf{e}}) = \frac{(\mathbf{e}_0 + \mathbf{e})!}{\mathbf{e}!} \cdot \mathrm{coeff}_f(\mathbf{t}^{\mathbf{e}_0 + \mathbf{e}})$. Therefore, we can further simplify our expression for $D_h f$ as follows.

$$\begin{aligned} D_h f &= \sum_{\mathbf{e}_0} \frac{\mathbf{t}^{\mathbf{e}_0}}{\mathbf{e}_0!} \cdot \left(\sum_{\mathbf{e}} \mathrm{coeff}_h(\mathbf{t}^{\mathbf{e}}) \cdot \mathbf{e}! \cdot \mathrm{coeff}_{g_0}(\mathbf{t}^{\mathbf{e}}) \right) \\ &= \sum_{\mathbf{e}_0} \frac{\mathbf{t}^{\mathbf{e}_0}}{\mathbf{e}_0!} \cdot (D_{g_0} h)(\mathbf{0}) && \text{(Using Observation 2.13)} \\ &= \sum_{\mathbf{e}_0} 0 \cdot \frac{\mathbf{t}^{\mathbf{e}_0}}{\mathbf{e}_0!} \equiv 0 && (D_{g_0} \in \Delta_f \text{ and } h \in J) \end{aligned}$$

Thus, $h \in f^{\perp}$. ◀

3.2 Proof of Theorem 1.4

We are now ready state the general recipe for constructing a commutative ROABP for any homogeneous $f(x_1, \ldots, x_n)$ of degree d. We start by defining the following polynomial over $\mathbf{x}$, and an auxiliary set of variables $\mathbf{t} = \{t_1, \ldots, t_n\}$, which is the product of the degree-d-truncations of the Taylor series of $e^{t_i x_i}$s.

$$G(\mathbf{x}, \mathbf{t}) := \prod_{i=1}^{n} \left(1 + t_i x_i + \frac{1}{2!} t_i^2 x_i^2 + \cdots + \frac{1}{(d-1)!} t_i^{d-1} x_i^{d-1} + \frac{1}{d!} t_i^d x_i^d\right) \tag{3.1}$$

▶ **Observation 3.4.** *Let $D := D_f = f(\partial t_1, \partial t_2, \ldots, \partial t_n)$. Then $(D \circ G) = f(\mathbf{x})$.*

▶ **Lemma 3.5.** *Suppose $f(\mathbf{x})$ is a homogeneous polynomial with dimension of partial derivatives exactly w. Then, there exists a vector $\mathbf{v} \in \mathbb{C}^w$, such that for the multiplication tables $A_1, \ldots, A_n$ of the apolar ideal $f^\perp$, we have that*

$$\sum_{j \in [w]} v_j \cdot G(A_1, \ldots, A_n, x_1, \ldots, x_n)[1, j] = f(\mathbf{x}).$$

Proof. Since $A_1, \ldots, A_n$ are multiplication tables of $f^\perp$, we get that the first row of $G(\mathbf{A}, \mathbf{x})$ is exactly the coefficient vector of $(G(\mathbf{t}, \mathbf{x}) \bmod f^\perp(\mathbf{t}))$ which is an object in $\mathbb{C}[\mathbf{t}][\mathbf{x}]/f^\perp(\mathbf{t}) = (\mathbb{C}[\mathbf{t}]/f^\perp)[\mathbf{x}]$.

Next, suppose that $\{g_1(\mathbf{t}), \ldots, g_w(\mathbf{t})\}$ is a basis for the partial derivatives of $f(\mathbf{t})$. Then by Lemma 3.3, we know that $\Delta(f^\perp)$ has a basis given by the operators $\{D_{g_1}, \ldots, D_{g_w}\}$. Also, the variety of $f^\perp$ is exactly the singleton set $\{\mathbf{0}\}$. Thus, by Theorem 2.14, the coefficients of $[G(\mathbf{t}, \mathbf{x})] := G(\mathbf{t}, \mathbf{x}) \bmod f^\perp(\mathbf{t})$ are spanned by $D_{g_1}(G)(\mathbf{0}), D_{g_2}(G)(\mathbf{0}), \ldots, D_{g_w}(G)(\mathbf{0})$. More importantly, the set of $D_{g_i}(G)$'s is spanned by the coefficients of $[G(\mathbf{t}, \mathbf{x})]$, and further, $D_{g_1}(G) = D_f(G) = f(\mathbf{x})$.

Thus, the vector $\mathbf{v}$ can be obtained from the matrix M guaranteed by Theorem 2.14, as claimed. ◀

This proves the main theorem, restated below.

▶ **Theorem 1.4.** *For any homogeneous polynomial $f(\mathbf{x}) \in \mathbb{C}[\mathbf{x}]$, $\mathrm{commRO}(f) \leq \dim \partial^{<\infty}(f)$.*

▶ Remark 3.6. Note that Saxena's proof of the duality trick [21, Lemma 1] can be seen as starting with the same "template polynomial" as (3.1), homogenizing it through interpolation, and then just evaluating it on the "points" given by the Waring decomposition for f. Since it is known by the *Apolarity lemma* (see e.g. [8]) that the points given by a Waring decomposition of f define a radical ideal J that sits inside $f^\perp$, the action of evaluating on those points can be viewed as going modulo J.

In that sense, our proof generalizes this method by directly going modulo $f^\perp$. As this uses a less strict property of f, the resulting expression is less simple, and is therefore a commRO instead of a diagRO.

4 The Determinant

In this section, we will use the proof of Theorem 1.4 to construct explicit commutative ROABPs. In particular, we will construct a commutative ROABP for the determinant (Det_n) of width $2^{\Theta(n)}$.

The choice of this example is deliberate, as the determinant is the only candidate where there is an asymptotic gap between Waring rank upper bounds and partial derivative dimension.

The determinant of n-dimensional symbolic matrix has partial derivative dimension exactly $\binom{2n}{n} = 2^{\Theta(n)}$. But, the best upper bound for the Waring rank of the determinant is $2^{O(n\log n)}$. In fact, there are reasons to believe that the Waring rank of the determinant is $2^{\omega(n)}$. This is due to the fact that the set-multilinear depth-3 complexity or *Tensor rank* of Det_n, and (to the best of our knowledge) even the best constant-depth multilinear formula that we know of for the determinant, is $2^{O(n\log n)}$. See [9, 19] for details on tensor rank and syntactic multilinear formulas of the determinant.

For Det_n, Theorem 1.4 directly gives a commutative ROABP of width $2^{\Theta(n)}$. We show the explicit calculations behind this commutative ROABP below. Let's recall what our overall step-by-step process will be for any polynomial $f \in \mathbb{C}[\mathbf{x}]$:

1. Compute the closed derivative space $\Delta = \partial^{<\infty} f$, the apolar ideal corresponding to it $I_{\mathbf{0}}(\Delta) = f^{\perp}$ and the normal set of $\mathrm{N}(f^{\perp})$. Let, $m := |\mathrm{N}(f^{\perp})| = |\Delta|$.
2. Compute the multiplication tables (M_i) corresponding to each of the variables.
3. The final commutative ROABP of $f \equiv \mathbf{a}^{\intercal} \cdot \prod_{i\in[n]}(1 + x_i M_i) \cdot \mathbf{b}$ for $\mathbf{a}, \mathbf{b} \in \mathbb{C}^m$.

4.1 Derivative Space, Apolar Ideal, and its Normal Set

In this subsection, we will state and discuss some facts about the derivative space, apolar ideal, and the normal set of the apolar ideal of the determinant. We will use X to denote the $n \times n$ symbolic matrix, that is, $X = (x_{i,j})_{i,j\in[n]}$. Similarly, let $U = (t_{i,j})_{i,j\in[n]}$ be a symbolic matrix in t-variables. Let S, T be arbitrary subsets of $[n]$. We will denote the minor (of X) picked by selecting rows from S and columns from T by $X_{S,T}$.

We will start with the well-known fact that the derivative space of determinant is just the determinant of its minors. Formally, the following set is a basis of $\partial^{<\infty} \mathsf{Det}_n(X)$,

$$\{\mathsf{Det}(X_{S,T}) : \text{ for } S, T \subseteq [n] \text{ such that } |S| = |T|\} .$$

The apolar ideal of the determinant is generated by permanents of 2×2 minors and certain unacceptable degree two monomials, as stated formally below.

▶ **Theorem 4.1** (e.g. [22, Theorem 2.12]). $\mathsf{Det}_n^{\perp}(X) = \langle \mathcal{P}_X, \mathcal{U}_X \rangle$*, where* $\mathcal{P}_X$ *is the collection of permanents of all* 2×2 *minors of* X*, and* $\mathcal{U}_X$ *denotes all quadratic unacceptable monomials, that is, monomials that don't divide any monomial in the support of* $\mathsf{Det}_n(X)$*.*

Now, for the normal set computation, we will focus on the degree-wise lexicographical ordering of monomials ("deg-lex"); the ordering on variables is in the "row-major" form:

$$x_{1,1} \succ x_{1,2} \succ \ldots \succ x_{1,n} \succ x_{2,1} \succ \ldots \succ x_{n,n}.$$

The trailing monomial of $\mathsf{Det}(X_{S,T})$ in this ordering is just the product of the anti-diagonal entries of $X_{S,T}$. To see this, note that the only variable you can pick from the first row is the last element. Now that we have picked something from the last column and the first row, we can strip them off as none of the variables can contribute anymore. Now focus on the resulting minor (after stripping) and proceed by induction. Let's denote this trailing monomial by $\tau_{S,T} := \text{anti-diag}(X_{S,T})$.

We now claim that the normal set of J is just a collection of these anti-diagonal monomials corresponding to all minors, as follows.

$$\mathrm{N}(J) = \{\text{anti-diag}(X_{S,T}) : \text{ for } S, T \subseteq [n] \text{ such that } |S| = |T|\}$$

To see this, observe that any non-anti-diagonal monomial (for any minor) will be a multiple of the leading term of a 2×2 minor's permanent and thus will be in $\mathrm{LM}(\mathsf{Det}^{\perp})$. At the same time, the anti-diagonal monomial will never be a multiple of such terms, so it is never in $\mathrm{LM}(\mathsf{Det}^{\perp})$. Note that here, $|\Delta| = |\mathrm{N}(J)| = \binom{2n}{n}$.

4.2 Multiplication tables for the apolar ideal

Let $A_{i,j}$ be the matrix corresponding to $t_{i,j}$ with dimension $|\mathrm{N}(J)| \times |\mathrm{N}(J)|$. For any row of $A_{i,j}$, indexed by (S,T) such that $|S| = |T|$, we have that

$$row(S,T)(A_{i,j}) = \begin{cases} 0 & \text{if } i \in S \text{ or } j \in T \\ \mathrm{sgn}(\tau_{S',T'}) \cdot \mathrm{sgn}(\tau_{S,T} \cdot x_{i,j}) \cdot \tau_{(S\cup\{i\}, T\cup\{j\})} & \text{if } i \notin S, j \notin T \end{cases}.$$

Here, sgn of any monomial denotes the sign of its corresponding permutation (obtained by viewing $S, T \equiv \{1, \ldots, |S|\}$). To see this, note that by definition $row(S,T)(A_{i,j})$ is just the coefficient vector of $(t_{i,j} \cdot \tau_{S,T} \bmod J)$.

Thus, as discussed (in proof of Theorem 1.4) we get

$$\mathsf{Det}_n(X) = \mathbf{a}^{\mathsf{T}} \cdot \prod_{i,j \in [n]} (I + A_{i,j} x_{i,j}) \cdot \mathbf{b} \qquad \text{for } \mathbf{a}, \mathbf{b} \in \mathbb{C}^{\binom{2n}{n}}.$$

Below, we give an explicit description of the commutative ROABP for Det_2. Here, the normal set is $\mathrm{N}(\mathsf{Det}_2^{\perp}) = \{1, x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}, x_{1,2}x_{2,1}\}$. Running our analysis to compute the multiplication tables followed by then replacing it in the template polynomial yields that the $\mathsf{Det}_2(X)$ is the $(1,n)$-th entry of the product of the following four matrices (in any order).

$$M_{1,1} = \begin{pmatrix} 1 & x_{1,1} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -x_{1,1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; M_{1,2} = \begin{pmatrix} 1 & 0 & x_{1,2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & x_{1,2} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

$$M_{2,1} = \begin{pmatrix} 1 & 0 & 0 & x_{2,1} & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & x_{2,1} \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \; M_{2,2} = \begin{pmatrix} 1 & 0 & 0 & 0 & x_{2,2} & 0 \\ 0 & 1 & 0 & 0 & 0 & -x_{2,2} \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

4.3 Commutative Set-multilinear ABP

The commutative matrices $A_{i,j}$ generated in the previous subsections using multiplication tables from $\mathsf{Det}_n^{\perp}$ can in fact be used to design a *commutative* set-multilinear ABP for Det as well.

The benefit of studying this model stems from the fact that if we could somehow "diagonalize" these commutative matrices, then that would indeed give a set-multilinear depth-3 representation of the determinant of size $2^{O(n)}$. That, in turn, would yield that the Waring rank of the determinant is also $2^{O(n)}$. By "diagonalizable," we simply mean if the above matrices can be replaced by diagonal matrices with at most a polynomial blow-up in the dimension.

In fact, the commutative matrices constructed for Det_n are provably not diagonalizable by invertible transformations. But for Perm_n, the matrices that we will get by running our entire analysis are indeed "diagonalizable"! In the sense that we can replace them by diagonal matrices of similar dimension to compute Perm_n.

Let $\sqcup_{j\in[d]}\mathbf{x}_j$ be a partition of the set $\mathbf{x}$ of input variables. Then a polynomial is set-multilinear under partition $\sqcup_{j\in[d]}\mathbf{x}_j$ if each monomial of the polynomial picks up *exactly* one variable from each part in the partition. Note that, Det is set-multilinear w.r.t. the variable partition being the row variables (or column variables).

▶ **Definition 4.2** (Set-multilinear ABP (smABP)). *Let $n, d, w \in \mathbb{N}$, and let $f(\mathbf{x})$ be an n-variate set-multilinear polynomial under the partition $\mathbf{x}_1 \sqcup \mathbf{x}_2 \sqcup \cdots \sqcup \mathbf{x}_d$. We say that f has a width w set-multilinear ABP*[3]*, if there exists a permutation $\sigma \in s_d$ for which there exist matrices $\{A_{j,k}\}$ in $\mathbb{C}^{w\times w}$ for all $j \in [d]$ and $1 \le k \le |\mathbf{x}_j|$, and vectors $\mathbf{u}, \mathbf{v} \in \mathbb{C}^w$, such that the following holds.*

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{u}^{\mathsf{T}} \cdot M_{\sigma(1)}(\mathbf{x}_{\sigma(1)}) \cdot M_{\sigma(2)}(\mathbf{x}_{\sigma(2)}) \cdots M_{\sigma(d)}(\mathbf{x}_{\sigma(d)}) \cdot \mathbf{v}, \\ &\quad \textit{where for all } j \in [d], \\ M_j(\mathbf{x}_j) &= A_{j,1}x_{j,1} + A_{j,2}x_{j,2} + \cdots + A_{j,|\mathbf{x}_j|}x_{j,|\mathbf{x}_j|}. \end{aligned}$$

We call the matrices $\{A_{j,k}\}$ the coefficient matrices *of the smABP.*

▶ **Definition 4.3** (Commutative smABP). *An smABP is said to be a* commutative smABP *if all its coefficient matrices pairwise commute with each other.*

Now, we will show that by simply changing the template polynomial (from (3.1)) appropriately, we can get a commutative set-multilinear ABP representation for Det_n.

$$\text{Define, } G(\mathbf{x},\mathbf{t}) := \prod_{i=1}^{n}\left(\sum_{j\in[n]} t_{i,j}x_{i,j}\right). \tag{4.1}$$

Again, just like Observation 3.4, we have that $\mathsf{Det}_n(\partial_{t_{1,1}}, \partial_{t_{1,2}}, \ldots, \partial_{t_{n,n}}) \circ G = \mathsf{Det}_n(X)$. And this gives that,

$$\mathsf{Det}_n(X) = \mathbf{a}^{\mathsf{T}} \cdot \prod_{i=1}^{n}\left(\sum_{j\in[n]} A_{i,j}x_{i,j}\right) \cdot \mathbf{b} \qquad \text{for some } \mathbf{a}, \mathbf{b} \in \mathbb{C}^{\binom{2n}{n}}.$$

We remark that the above analysis works for any set-multilinear polynomial, along the same lines as the proof of Theorem 1.4. This directly gives us the following theorem.

▶ **Theorem 1.6.** *For any set-multilinear polynomial $f(\mathbf{x}) \in \mathbb{C}[\mathbf{x}]$, the commutative-set-multilinear-ABP-width$(f) \le \dim \partial^{<\infty}(f)$.*

5 Discussion

In summary, we utilize the knowledge of commuting matrices outlined in [17] to provide a generic recipe for explicit constructions of commutative branching programs. For the specific setting of ROABPs, this improves upon the earlier known connection (Observation 2.7) and takes us a step closer towards answering Question 1.1. An immediate direction for further study is as follows.

[3] Strictly speaking this defines *ordered* set-multilinear algebraic branching programs, but we drop this detail for brevity.

Can we show any bounds on the commRO width of a polynomial in terms of its diagRO width? In addition to shedding further light on Question 1.1, such a result should also provide us with a new hardness measure for structured ROABPs and possibly even depth 3 powering circuits, particularly if the bound is a super-linear lower bound on diagRO width. As alluded to in the introduction, perhaps a new hardness measure against depth 3 powering circuits is what is required to fully derandomize blackbox PIT for the model. A concrete way in which this is true is that a polynomial time blackbox PIT for width-w, degree-d, $O(\log dw)$-variate diagRO would give a polynomial time blackbox PIT for depth 3 powering circuits [1, Lemma 2.12]. This gives the lower bound question a much larger and more interesting context.

References

1 Pranav Bisht and Nitin Saxena. Blackbox identity testing for sum of special roabps and its border class. *Computational Complexity*, 30(1):8, 2021. doi:10.1007/s00037-021-00209-y.

2 Enrico Carlini, Maria Virginia Catalisano, and Anthony V. Geramita. The solution to the waring problem for monomials and the sum of coprime monomials. *Journal of Algebra*, 370:5–14, 2012. doi:10.1016/j.jalgebra.2012.07.028.

3 David A. Cox, John B. Little, and Donal O'Shea. *Ideals, Varieties and Algorithms*. Undergraduate texts in mathematics. Springer, 2007. doi:10.1007/978-0-387-35651-8.

4 Ismor Fischer. Sums of like powers of multivariate linear forms. *Mathematics Magazine*, 67(1):59–61, 1994.

5 Michael A. Forbes, Ramprasad Saptharishi, and Amir Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 867–875, 2014. doi:10.1145/2591796.2591816.

6 Michael A. Forbes and Amir Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 243–252, 2013. Full version at arXiv:1209.2408. doi:10.1109/FOCS.2013.34.

7 Rohit Gurjar, Arpita Korwar, and Nitin Saxena. Identity testing for constant-width, and commutative, read-once oblivious abps. *Theory of Computing*, 13(1):1–21, 2017. Preliminary version in the *31st Annual Computational Complexity Conference (CCC 2016)*. arXiv:1601.08031. doi:10.4086/toc.2017.v013a002.

8 Anthony Iarrobino and Vassil Kanev. *Power Sums, Gorenstein Algebras, and Determinantal Loci*. Springer-Verlag Berlin Heidelberg, 1999. doi:10.1007/BFb0093426.

9 Siddharth Krishna and Visu Makam. On the tensor rank of 3s×3 permanent and determinant. *CoRR*, abs/1801.00496, 2018. arXiv:1801.00496.

10 Mrinal Kumar and Ramprasad Saptharishi. Hardness-randomness tradeoffs for algebraic computation. *Bulletin of EATCS*, 1(129), 2019.

11 M.G. Marinari, H.M. Möller, and T. Mora. Gröbner bases of ideals defined by functionals with an application to ideals of projective points. *Applicable Algebra in Engineering, Communication and Computing*, 4(2):103–145, 1993. doi:10.1007/BF01386834.

12 H. Michael Möller and Hans J. Stetter. Multivariate polynomial equations with multiple zeros solved by matrix eigenproblems. *Numerische Mathematik*, 70, 1995. doi:10.1007/s002110050122.

13 Noam Nisan. Lower bounds for non-commutative computation. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC 1991)*, pages 410–418, 1991. Available on citeseer:10.1.1.17.5067. doi:10.1145/103418.103462.

14 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

15 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997. Available on citeseer:10.1.1.90.2644. doi:10.1007/BF01294256.

16 Kevin Pratt. Waring rank, parameterized and exact algorithms. In David Zuckerman, editor, *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, pages 806–823. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00053.

17 C. Ramya and Anamay Tengse. On finer separations between subclasses of read-once oblivious abps. 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 53:1–53:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.STACS.2022.53.

18 Kristian Ranestad and Frank-Olaf Schreyer. On the rank of a symmetric form. *Journal of Algebra*, 346(1):340–342, 2011.

19 Ran Raz. Tensor-rank and lower bounds for arithmetic formulas. In *Proceedings of the 42nd Annual ACM Symposium on Theory of Computing (STOC 2010)*, pages 659–666, 2010. doi:10.1145/2535928.

20 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. Github survey, 2015. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/.

21 Nitin Saxena. Diagonal Circuit Identity Testing and Lower Bounds. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, pages 60–71, 2008. doi:10.1007/978-3-540-70575-8_6.

22 Sepideh Masoumeh Shafiei. Apolarity for determinants and permanents of generic matrices. *Journal of Commutative Algebra*, 7(1):89–123, 2015. URL: https://www.jstor.org/stable/26174731.

23 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5:207–388, March 2010. doi:10.1561/0400000039.

Many Flavors of Edit Distance

Sudatta Bhattacharya
Charles University, Prague, Czech Republic

Sanjana Dey
National University of Singapore, Singapore

Elazar Goldenberg
The Academic College of Tel-Aviv-Yaffo, Israel

Michal Koucký
Charles University, Prague, Czech Republic

Abstract

Several measures exist for string similarity, including notable ones like the edit distance and the indel distance. The former measures the count of insertions, deletions, and substitutions required to transform one string into another, while the latter specifically quantifies the number of insertions and deletions. Many algorithmic solutions explicitly address one of these measures, and frequently techniques applicable to one can also be adapted to work with the other. In this paper, we investigate whether there exists a standardized approach for applying results from one setting to another. Specifically, we demonstrate the capability to reduce questions regarding string similarity over arbitrary alphabets to equivalent questions over a binary alphabet. Furthermore, we illustrate how to transform questions concerning indel distance into equivalent questions based on edit distance. This complements an earlier result of Tiskin (2007) which addresses the inverse direction.

2012 ACM Subject Classification Theory of computation → Random projections and metric embeddings

Keywords and phrases Edit distance, Indel distance, Embedding, LCS, Alphabet Reduction

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.11

Related Version *Full Version*: `https://arxiv.org/abs/2410.09877`

Funding *Sudatta Bhattacharya*: Partially supported by the project of Czech Science Foundation no. 19-27871X, 24-10306S and by the project GAUK125424 of the Charles University Grant Agency.
Michal Koucký: Partially supported by the project of Czech Science Foundation no. 19-27871X and 24-10306S.

Acknowledgements The project started at EPAC Workshop: Algorithms and Complexity partially supported by the project of Czech Science Foundation no. 19-27871X.

1 Introduction

String-related metrics, such as edit distance, longest common subsequence distance, are pivotal in numerous applications that deal with text or sequence data. Edit distance metric, also referred to as Levenshtein distance [18], quantifies the minimum number of single-character edits (insertions, deletions, or substitutions) necessary to convert one string into another. This metric finds extensive application in spell checking and correction systems, DNA sequence alignment and bioinformatics for comparing genetic sequences, as well as in natural language processing tasks such as machine translation and text summarization, among other fields. The longest common subsequence metric, also known as *Indel* or *LCS distance*, evaluates the disparity between two strings by determining the minimum number of single-character edits while forbidding substitutions. This metric has diverse applications, including text comparison and plagiarism detection, DNA and protein sequence analysis for recognizing shared regions or motifs, music analysis to uncover similarities between musical sequences, and document clustering and classification based on content similarity.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 11; pp. 11:1–11:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

From a computational complexity perspective computing distances under the Edit or Indel metric typically takes quadratic time complexity, as initially demonstrated by Wagner [23]. Subsequent research has marginally improved this complexity by reducing logarithmic factors, as evidenced by Masek and Paterson [19] and Grabowski [15]. Additionally, Backurs and Indyk [5] demonstrated that a truly sub-quadratic algorithm $O(n^{2-\delta})$ for some $\delta > 0$ would lead to a $2^{(1-\gamma)n}$-time algorithm for CNF-satisfiability, contradicting the Strong Exponential Time Hypothesis. Similarly, Abboud et al. [1] established a similar result for computing the Indel metric between string pairs. Notably, obtaining an efficient isometric embedding for the edit metric into the Indel metric would effortlessly yield the latter result.

Extensive research has been conducted on approximating edit distance, with studies dating back to the work of Landau et al. [17, 6, 8, 4, 7, 9, 11, 14, 16, 10], ultimately culminating in the breakthrough result of Andoni and Nosatzki [3], which offers a (large) constant factor approximation in nearly linear time. However, approximating the Indel distance has not received similar attention, and although one expects the same techniques should provide similar results for Indel distance, one would need to check all the details of the construction to verify the exact properties of such a result. This exhibits a general pattern where results for one of the measures can often be adapted for the other but there is no simple tool that would guarantee such an automatic transformation.

Similar pattern emerges when dealing with the string measures over different size alphabets. For example, the hardness result elucidated by Backurs and Indyk [5] is constrained to some "large" constant-size alphabets. Subsequent research has revealed that the computational task of computing edit distance is also hard for binary alphabets using an ad hoc approach. Once more, the possibility of achieving an efficient isometric embedding between strings residing in large alphabets and those in smaller ones could potentially resolve these questions automatically. Another scenario where the alphabet size becomes relevant is in the simple linear-time approximation algorithm for the length of the LCS. The naive algorithm provides a $|\Sigma|$-approximation, where Σ represents the alphabet to which the strings reside. Therefore, if one can embed strings from a large alphabet into those from a smaller alphabet while approximately preserving distances, it may lead to an improvement in the approximation factor. Given the current circumstances, we pose the following questions, which we then delve into extensively:

▶ **Question 1.** *Does an isometric embedding exist between the edit metric and the Indel metric? Can it be computed efficiently?*

▶ **Question 2.** *Does an isometric embedding exist between edit metric on arbitrary alphabets and the edit distance on binary alphabets? Can it be computed efficiently?*

1.1 Our Contribution

This paper introduces multiple mappings that establish connections between various string metrics. Specifically, we transform points residing within a designated input metric space (M, d) into points within an output metric space (M', d'), ensuring that the distance between any pair of points in M under d, is (approximately) preserved on the output pairs under d'. In this paper, we introduce a relaxation of the concept of isometric embedding, permitting scaling factors, as outlined below.

We say an embedding $E : M \to M'$ is **scaled isometric** if there exists a function $f : \mathbb{N} \to \mathbb{N}$ that maps distances from the original space to the embedded one, such that for every pair of points $x, y \in M$ we have: $d'(E(x), E(y)) = f(d(x, y))$. This implies that such an embedding can be utilized to reduce the computation of distances in the original metric M to computing distances under M', provided that f is invertible and computationally tractable.

A special case of scaling involves preserving the normalized distances. That is, for any metric on a string space, it is intuitive to define the normalized distance between a pair of strings of a specified length, as their distance divided by the maximum distance of pairs of that same length. Subsequently, for an embedding $E : M \to M'$ that preserves lengths (i.e., the length of the output string is a function of the input string), we further assert that it is **normalized scaled isometric** if, for every pair from M, the normalized distance of the embedded strings preserves the normalized distance of the original ones.

Utilizing normalized scaled isometric embedding can facilitate the computational process of approximating distances in the original metric M to compute approximate distances under M', even if the normalized distances are not perfectly preserved by the embedding but distorted by a (multiplicative) factor of c. In such a case, any c' approximation algorithm for the metric M' could thus be transformed into a $c \cdot c'$ approximation algorithm for M.

1.1.1 Our Results

In the sequel, we utilize the notation Δ_{indel} to represent the Indel distance between a pair of strings, Δ_{edit} to denote their edit distance. Additionally, we use $\widetilde{\Delta}_{indel}$ and $\widetilde{\Delta}_{edit}$ to represent the normalized Indel distance between a pair of strings (for precise definitions, refer to Section 2).

Alphabet Reduction – Succinct Embedding. Our first result pertains to alphabet reduction achieving normalized scaled isometric embedding. In this context, as elaborated in Section 3, any embedding contracts the normalized distances of some of the pairs. Consequently, we shift our focus to approximate normalized scaled isometric embedding, where we permit slight distortions in the normalized distances. Our main result is outlined below:

▶ **Theorem 3** (Alphabet Reduction - Succinct Embedding). *Let Γ be a finite alphabet, and let $0 < \varepsilon < 1/4$. There exists an alphabet Σ, where $|\Sigma| = O(\frac{1}{\varepsilon^2})$ and there exists $E : \Gamma^* \to \Sigma^*$ satisfying:*

$$\forall X, Y \in \Gamma^* : \ \widetilde{\Delta}_{indel}(E(X), E(Y)) \in \left[(1-\varepsilon)\widetilde{\Delta}_{indel}(X,Y), \widetilde{\Delta}_{indel}(X,Y)\right].$$

Moreover, for every $X \in \Gamma^n$ we have: $|E(X)| = O(n \log(|\Gamma|))$.

Observe that embedded string length is optimal up to logarithmic factors. The size of the alphabet Σ must exceed $1/\varepsilon$, as otherwise, by a claim proved later, there will be a pair of strings whose distance will be contracted by at least $\left(1 - \frac{1}{|\Sigma|}\right)$-factor.

As a result, we find that when focusing on Δ_{indel} approximation, we can, without loss of generality, limit our scope to a constant alphabet size without significantly impacting the quality of the approximation. This is captured in the following corollary which we provide without a proof.

▶ **Corollary 4.** *Let Γ be a finite alphabet, and let $0 < \varepsilon < 1/4$. Suppose that there exists an algorithm* ALG *that provides a c-factor approximation for the Δ_{indel} metric for any pairs of strings in Σ of total length N, where $\Sigma = O(1/\varepsilon^2)$, running in time $t(N)$.*

Then there exists an algorithm ALG$'$ *that, given any pair of string in Γ of total length n, provides a $c + \varepsilon$-approximation for their Δ_{indel} distance in time $t(n \log|\Gamma|)$.*

The proof strategy of Theorem 3 is as follows: Initially, we construct an error-correcting code within the smaller alphabet Σ, where $|\Sigma| = O(1/\varepsilon^2)$. This code ensures that every distinct pair of codewords shares only a small portion of LCS, indicating a significant Δ_{indel}

between them. The code comprises $|\Gamma|$ words, with its dimension being logarithmic in the size of Γ. We interpret this code as a mapping from characters within Γ to short strings in Σ. The existence of such a code can be demonstrated using the probabilistic method. The construction is almost tight in terms of the smaller alphabet size: it is not hard to show that for any code $C \subseteq \Sigma^k$, if $|C| > |\Sigma|$, then there exist $c \neq c' \in C$, satisfying: $\Delta_{indel}(c, c') < (1 - \frac{1}{|\Sigma|})2k$.

Employing the code construction, we embed input strings in a straightforward and natural manner: Consider a string residing in Γ^*, then each of its characters is sequentially encoded using the code. This encoding ensures that identical characters are mapped to the same codeword, while the code's distance guarantees that distinct characters may have only a few shared matches, akin to the global nature of Δ_{indel} alignments. If there were no matches between distinct characters' encodings, any alignment for the embedded strings could be converted into an alignment between the input strings without any distortion in the normalized costs. However, these few matches between non-matching codewords introduce a slight distortion and complicate the proof.

Alphabet Reduction – Binary Alphabets. Our previous method relied on a local approach, wherein the characters of the string from the large alphabet were encoded sequentially and independently. It appears that such a local strategy may not result in a normalized scaled isometric embedding into a small, particularly binary alphabet. As codes in such alphabets have a relative distance of at most 1/2, and this distance affects the distortion of the normalized distances. Therefore, achieving alphabet reduction into binary alphabets requires a scaling that is not normalized, as well as a more global approach that does not encode the characters sequentially and independently. However, there's a caveat: the encoding still needs to be performed independently on each string rather than on a pair of strings. Specifically, if we take a particular string X its encoding will remain the same regardless of the second string Y. This aspect forms the focal point of our forthcoming result.

For clarity, we present our results for the Δ_{indel} metric, with a similar approach applicable to the Δ_{edit} metric. Our main result states that one can embed Δ_{indel} over any alphabet Σ into binary alphabet. We employ asymmetric embedding and prove that the distances are preserved up to some scaling function. The dimension of the embedded strings is quasi-polynomial, making this result more of a proof of concept at the moment. Nonetheless, we find it conceptually intriguing and pose the question of decreasing the target dimension as an open question. Our main result is as follows:

▶ **Theorem 5** (Informal statement of Theorem 16). *For any alphabet Σ and for every $n \in \mathbb{N}$ there exist functions $G, H : \Sigma^n \to \{0,1\}^N$, $f : [n] \to [N]$ where $N = n^{O(\log n)}$ such that for any $X, Y \in \Sigma^n$,*

$$\Delta_{indel}(G(X), H(Y)) = f(\Delta_{indel}(X, Y)).$$

Our initial consideration revolves around the fact that deciding whether $\Delta_{indel}(X, Y) \leq k$ for two strings $X, Y \in \Sigma^n$ and a threshold parameter k, can be accomplished by a Turing machine utilizing $\log(n)$-space. This capability can then be translated into a SAT formula of $\log^2(n)$-depth.

The foundation of our construction converts this formula into a pair of binary strings X', Y' of quasi-polynomial length, where X' (respectively, Y') depends solely on X (Y) and the Indel distance between X', Y' is contingent on the distance between X, Y. This is achieved by recursively transforming the formula into such a pair of strings gate by gate. Two essential components, referred to as *AND*- and *OR*-gadgets, implement this process.

The input for the AND-gadget consists of two pairs of strings $(X_0, Y_0), (X_1, Y_1)$, which can be thought of as outputs from previous levels. Here the X_i's only depend on X and the Y_i's only depend on Y. Moreover, we are guaranteed that $\Delta_{indel}(X_i, Y_i)$ can only take two values $\{F, T\}$, where $F < T$. The goal is to concatenate the X_i into a single string X and the Y_i's into a different string Y such that: $\Delta_{indel}(X, Y)$ can also take value in $\{F', T'\}$, where $F' < T'$ and: $\Delta_{indel}(X, Y) = T'$ if and only if $\Delta_{indel}(X_0, Y_0) = T \wedge \Delta_{indel}(X_1, Y_1) = T$. Similarly for the OR-gadget. Our construction of the LCS instance from the Formula Evaluation is similar to that of Abboud and Bringmann [2] which considers reduction of the Formula Satisfiability to LCS. The purpose of our reduction is different, though.

Scaled Isometric Embedding of Indel into Edit Metrics. While our previous results aimed on reducing the alphabet size while keeping the underlying metric (either Δ_{indel} or Δ_{edit}), this section focuses on converting one metric into another. Tiskin [21] in section 6.1 proposed a straightforward embedding from the Δ_{edit} metric to the Δ_{indel} metric. This inspired our exploration into embedding in the reverse direction i.e. from the Δ_{indel} metric to the Δ_{edit} metric. Our primary contribution in this realm is a scaled isometric embedding from the Δ_{indel} metric to Δ_{edit}, as outlined below.

▶ **Theorem 6** (Indel Into Edit Metrics Embedding - Approximate embedding). *For any alphabet Σ, $n \in \mathbb{N}$ and $\varepsilon \in (0, 1]$, there exist mappings $E : \Sigma^n \to \Sigma^n$ and $E' : \Sigma^n \to (\Sigma \cup \{\$\})^N$, where $N = \Theta(n/\varepsilon)$, such that for any $X, Y \in \Sigma^n$, we have*

$$\Delta_{edit}(E(X), E'(Y)) = N - n + k, \text{ where } k \in \left[\frac{\Delta_{indel}(X, Y)}{2}, (1 + \varepsilon)\frac{\Delta_{indel}(X, Y)}{2}\right).$$

Observe that while plugging $\varepsilon \leq \frac{1}{n}$ we obtain a scaled isometry at the expense of a quadratic increase in the length of the second string. Conversely, for constant values of ε, N scales as $O(n)$ albeit with the trade-off of only approximately preserving distances within a constant factor. For intermediate values of ε, we can compromise between the accuracy and the stretch length.

Let us revisit Tiskin's (section 6.1 of [21]) construction of the reverse embedding, namely, from Δ_{edit} into Δ_{indel}. The embedding proceeds as follows: a special character \$ is appended after every symbol of each string. It is easy to check that for each pair of strings, the Δ_{indel} between the embedded pair of strings equals twice the Δ_{edit} between the original pair. In our construction, one string remains unaltered, while for the second string, we append after every symbol a block of length n consisting of the special character.

The core of the proof demonstrates the conversion of any Δ_{indel}-alignment for the input strings into an Δ_{edit}-alignment for the embedded strings, preserving the distances up to a scaling factor. This process involves replacing any deletions originally performed on the first string by substituting the characters with the special inserted character. Deletions made on the second string remain unaffected.

1.2 Related Work

The problem of embedding edit distance into other distance measures, such as Hamming distance, ℓ_1, etc., has attracted significant attention in the literature. Let us briefly survey some of these approaches.

Chakraborty et al. [12] introduced a randomized embedding scheme from the edit distance to the Hamming distance. This embedding transforms strings from a given alphabet into strings that are three times longer. For each pair of strings embedded using the same random sequence, with high probability the edit distance between the embedded strings is at most quadratic in the Hamming distance of the original strings. Batu et al. [8] introduced a dimensionality reduction technique: Given a parameter $r > 1$, they reduce the dimension by a factor of r at the expense of distorting the distances by the same factor. They employed the locally consistent parsing technique for their embedding. Ostrovsky and Rabani [20] presented a polynomial time embedding from edit distance to ℓ_1 distance with a distortion of $\mathcal{O}(2^{\sqrt{\log n \log\log n}})$. They proposed a randomized embedding where the length of the output strings is quadratic in the input strings, and the distances are preserved, with high probability, up to the distortion factor.

1.3 Future Directions

Introducing a Robust Concept of Approximation: Transitioning from Approximating Δ_{edit} into Approximating Δ_{indel}. Recall that one of the reasons we aimed to isometrically embed the Δ_{indel} metric into the Δ_{edit} metric stemmed from the abundance of approximation results for Δ_{edit} that might not easily extend to the Δ_{indel} metric. A natural approach, based on our embedding result, is to approximate the Δ_{indel} distance between X, Y by approximating the Δ_{edit} between the embedded strings. However, this is not an immediate consequence due to the substantial disparity in length between the embedded strings and the notion of approximation in this case, as detailed next:

Recall that in Theorem 6 the scaling mechanism is not normalized, i.e, the embedding function did not preserve normalized distances, but instead:

$$\Delta_{edit}(E_1(X), E_3(Y)) = N - n + k, \text{ where } k \in \left[\frac{\Delta_{indel}(X,Y)}{2}, \ldots, (1+\varepsilon)\frac{\Delta_{indel}(X,Y)}{2}\right]$$

where N, n are the length of the embedded strings, and $N = \Theta(\frac{n}{\varepsilon})$. Observe that the Δ_{edit} between the embedded strings lies in the range of $[N - n, N]$.

Considering the substantial difference in length between the embedded strings, an algorithm that consistently outputs the value $N - n$, regardless of the embedded strings, already yields a $1 + O(\varepsilon)$-approximation for the distance between the embedded strings. Certainly, such an outcome provides no information about the Δ_{indel} of the original strings. Therefore, we introduce a more robust notion of approximation that generally addresses the discrepancy in string lengths:

▶ **Definition 7** (A Robust Notion of Approximation). *Let $c > 1$, let Σ be a finite set, and let $X, Y \in \Sigma^*$. Define $|X| = N, |Y| = n$, and assume $N \geq n$. Define $k_{X,Y}$ such that: $\Delta_{edit}(X, Y) = N - n + k_{X,Y}$.*

An algorithm is considered to provide a robust c-approximation for Δ_{edit} if for all pairs X, Y it outputs k' such that: $k' \in [k_{X,Y}, ck_{X,Y}]$.

We assert that for any value of ε, any algorithm ALG that provides a robust c-approximation for Δ_{edit} yields an algorithm ALG$'$ that provides $(1+\varepsilon)c$-approximation for Δ_{indel}. Moreover, if the running time ALG on input strings of lengths N, n is $t(N, n)$, then the running time of ALG$'$ is $t\left(\frac{n}{\varepsilon}, n\right)$. The construction of ALG$'$ is straightforward: on input strings X, Y we first apply the embedding, then apply ALG on the resulting strings and finally output: $2k'$.

We leave the quest of discovering a robust approximation algorithm for Δ_{edit} as an open question, which falls outside the scope of this paper.

1.4 Organization Of The Paper

We structure the paper as follows. In Section 3 we prove our main result, namely Theorem 3, discussing normalized scaled isometric embedding between large and small alphabets. In Section 4 we establish Theorem 5 focusing on alphabet reduction with binary alphabets. We also demonstrate the existence of an indel to edit scaled isometric embedding, as stated in Theorem 6 the proof of which is in the full version of the paper.

2 Preliminaries and Notations

In this section we introduce the notations that is used throughout the rest of the paper. For any string $X = x_1 x_2 \dots x_n$ and integers i, j, $X[i]$ [1] denotes x_i, $X[i, j]$ represents substring $X' = x_i \dots x_j$ of X, and $X[i, j) = X[i, j-1]$. "$\cdot$"-operator denotes concatenation, e.g $X \cdot Y$ is the concatenation of two strings X and Y. Λ denotes the empty string.

Edit Distance with Substitutions (Δ_{edit}). For strings $X, Y \in \Sigma^*$, $\Delta_{edit}(X, Y)$ is defined as the minimal number of edit operations required to transform X into Y. The set of edit operations includes character insertion, deletion, and substitutions.

Indel Distance (Δ_{indel}). For strings $X, Y \in \Sigma^*$, $\Delta_{indel}(X, Y)$ is defined as the LCS (Longest Common Subsequence) metric between X and Y. It counts the minimal number of edit operations needed to convert the strings, where substitutions are excluded.

Normalized Distance. To assess the distance between each pair of strings in a standardized manner, it is advantageous to express it as a normalized value within the range $[0, 1]$. To achieve this, we introduce the following definition:

$$\widetilde{\Delta}_{edit}(X, Y) = \frac{\Delta_{edit}(X, Y)}{\max(|X|, |Y|)}, \quad \widetilde{\Delta}_{indel}(X, Y) = \frac{\Delta_{indel}(X, Y)}{|X| + |Y|}$$

A string X' is considered a **subsequence** of another string X if $\Delta_{indel}(X, X') = |X| - |X'|$. X' is considered a **substring** of X if X' is a contiguous subsequence of X.

Alignment. For an alphabet Σ and any two strings $X, Y \in \Sigma^*$, an Δ_{edit} *alignment of X and Y* is a a sequence of edit operations (insertions, deletions, and substitutions) that transform the string X into Y. The cost of the alignment is determined by the number of edit operations. An alignment is optimal if it achieves the lowest possible cost. Observe that for each character of X that wasn't deleted or substituted, can be matched with a unique character from Y. The collection of matched characters is referred to as the matching characters of the alignment.

Similarly we define an Δ_{indel} *alignment of X and Y* as a sequence of edit operations, with the exception that substitutions are not permitted. The cost, optimality, and matching characters of an alignment are defined analogously. See Figure 1 for an example.

[1] We use x_i or X_i or $X[i]$ to denote the i^{th} character of the string X interchangeably.

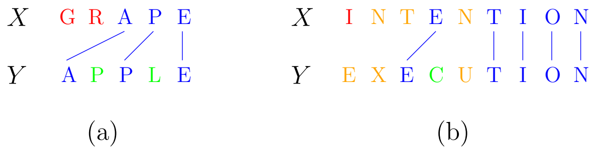


Figure 1 Example for (a) Δ_{indel} alignment and (b) Δ_{edit} alignment (the matched characters are highlighted in blue, the deleted characters in red and the substituted characters in orange).

3 Alphabet Reduction

Within this section, we tackle the task of embedding strings from a sizable alphabet into a smaller one while preserving the global nature of the original metric space. Our main result demonstrates that it's possible to embed strings from any large alphabet Γ into strings of a smaller alphabet Σ, where the length of the strings remains approximately unchanged, and the normalized distances are distorted by at most a factor of $(1+\varepsilon)$. The size of the alphabet Σ increases quadratically with $1/\varepsilon$. To provide clarity, we present our results for the Δ_{indel} metric; a similar approach can be applied to the Δ_{edit} metric. This section is structured as follows: In subsection 3.1 we outline our findings regarding normalized scaled alphabet reductions, covering both lower and upper bounds. Section 3.2 discusses our upper bounds, while Section 3.3 addresses lower bounds.

3.1 Normalized Scaled Isometric Embedding – Our Finding

An embedding E is said to preserve lengths, if there exists: $\ell : \mathbb{N} \to \mathbb{N}$ such that: $\forall X \in \Gamma^n : |E(X)| = \ell(n)$. It is non-shrinking if $\ell(n) \geq n$. It is natural to focus on non-shrinking embeddings otherwise if the embedding maps from a large alphabet to smaller one some distinct strings will get mapped to the same string.

The following claim shows that any length-preserving embedding, mapping strings from large alphabet into strings of smaller one, necessarily contracts the normalized distances between certain pairs of strings. The proof of the claim is deferred to Section 3.3.

▷ Claim 8. Let Γ, Σ be finite alphabets, such that: $|\Gamma| > |\Sigma|$, and let $E : \Gamma^* \to \Sigma^*$, which is a length-preserving embedding, then for any $n \in \mathbb{N}$ we have:

$$\exists X, Y \in \Gamma^n : \widetilde{\Delta}_{indel}(E(X), E(Y)) < \widetilde{\Delta}_{indel}(X, Y).$$

Therefore, we redirect our attention to approximate embedding, where we allow for a slight distortion in distances. Our main result is as follows:

▶ **Theorem 3** (Alphabet Reduction - Succinct Embedding). *Let Γ be a finite alphabet, and let $0 < \varepsilon < 1/4$. There exists an alphabet Σ, where $|\Sigma| = O(\frac{1}{\varepsilon^2})$ and there exists $E : \Gamma^* \to \Sigma^*$ satisfying:*

$$\forall X, Y \in \Gamma^* : \ \widetilde{\Delta}_{indel}(E(X), E(Y)) \in \left[(1-\varepsilon)\widetilde{\Delta}_{indel}(X, Y), \widetilde{\Delta}_{indel}(X, Y)\right].$$

Moreover, for every $X \in \Gamma^n$ we have: $|E(X)| = O(n \log(|\Gamma|))$.

Note that the distortion is "one-sided" in the sense that the normalized distances of the embedded strings cannot surpass the normalized distance of the original strings. However, for the lower bound, a $(1-\varepsilon)$-factor may be incurred. Furthermore, we demonstrate that for any embedding, the normalized distances cannot be uniformly scaled by a fixed factor. In particular, we demonstrate that there exist pairs of strings whose normalized distances are reduced, while for other pairs, their normalized distances converge to each other arbitrarily closely. This, in turn, illustrates that we cannot deduce the value of $\widetilde{\Delta}_{indel}(X,Y)$ directly from the value of $\widetilde{\Delta}_{indel}(E(X),E(Y))$ by a simple scaling.

▷ Claim 9. Let Γ,Σ be finite alphabets, such that: $|\Gamma|>|\Sigma|$, and let $E:\Gamma^*\to\Sigma^*$, which is a length-preserving non-shrinking embedding. We have:

1. For any $n\in\mathbb{N}$, there exist $X,Y\in\Gamma^n$ such that
$$\widetilde{\Delta}_{indel}(E(X),E(Y))\le(1-\frac{1}{|\Sigma|})\widetilde{\Delta}_{indel}(X,Y).$$
2. For any sequence $Z_1,Z_2,\dots$ where $|Z_n|=n$,
$$\lim_{n\to\infty}\widetilde{\Delta}_{indel}(E(Z_n),E(\Lambda))-\widetilde{\Delta}_{indel}(Z_n,\Lambda)=0,\text{where }\Lambda\text{ denotes the empty string.}$$

3.2 Upper Bounds

The crux of Theorem 3 lies in the existence of an error correcting code with respect to the Δ_{indel} metric, even when the alphabet size is small. More specifically, given a proximity parameter $\varepsilon>0$, and $|\Gamma|$, we pick a set C of strings residing in Σ^k of cardinality $|\Gamma|$, with large pairwise distance. The construction of C follows a greedy approach reminiscent of the Gilbert-Varshamov bound [13, 22]. Properties of the code are summarized in the next statement.

▶ **Lemma 10.** *For any $\varepsilon<1/2$, let Σ be a finite alphabet satisfying $|\Sigma|>32/\varepsilon^2$. For every $n\in\mathbb{N}$, there exists $k\in\mathbb{N}$ with $k=O(\log n)$ for which the following conditions are satisfied:*

1. *There exists a code $C_{n,\varepsilon}\subseteq\Sigma^k$ with $|C_{n,\varepsilon}|=n$.*
2. *$\forall c\neq c'\in C_{n,\varepsilon}:\ \Delta_{indel}(c,c')\ge(2-\varepsilon)k$.*

The proof of Lemma 10 is given in the full version of the paper.

Endowed with the existence of such an error correcting code we assign a distinct codeword to each of the characters in the larger alphabet Γ. The embedding procedure is as follows: Given a string $X\in\Gamma^n$, we embed it into a string in $(\Sigma^k)^n$, where the encoding of X is formed by concatenating the codewords assigned to each of its characters. The presentation of the embedding, along with its proof of correctness, is provided in Section 3.2.1.

3.2.1 The Embedding

Let Γ be an alphabet, and let ε be a proximity parameter, and let $C:=C_{|\Gamma|,\varepsilon}$ be the code whose existence is guaranteed by Lemma 10. We interpret the code C as a function mapping characters from Γ into (short) strings in Σ^k. The embedding proceeds as follows: each string in Γ^* is encoded sequentially character by character, where the encoding of each character is performed using the code C. Our main technical lemma is as follows:

▶ **Lemma 11.** *For any finite alphabet Γ and $\varepsilon>0$, consider the encoding $C_{|\Gamma|,\varepsilon}:\Gamma\to\Sigma^k$ as implied by Lemma 10. For any string $X\in\Gamma^*$ define: $E(X)=C(X_1)\dots C(X_n)$ (where $n=|X|$).*

Then for any pair of strings $X,Y\in\Gamma^*$ the following inequality holds:

$$(1-48\varepsilon)\Delta_{indel}(X,Y)\le\Delta_{indel}(E(X),E(Y))\le\Delta_{indel}(X,Y)$$

Moreover, for every $X\in\Gamma^n$ we have: $|E(X)|=O(n\log|\Gamma|)$.

In the sequel, an Δ_{indel}-alignment converting $E(X)$ into $E(Y)$ is simply referred as an alignment. In the course of the proof, we introduce the concepts of blocks and block-structured alignments. For $X \in \Gamma^n$ we define the i-th block of $E(X)$ to be the substring of $E(X)$ corresponding to X_i, namely it equals $C(X_i)$. Furthermore, given an alignment $\mathcal{A}$ that transforms $E(X)$ into $E(Y)$, we label it as a *block-structured* alignment if, for each i-th block in $E(X)$, the alignment either fully matches all the characters of the block to some block j in $E(Y)$ or entirely deletes the i-th block. It is clear that block-structured alignments for the embedded strings correspond one-to-one with alignments for the original strings, and their normalized distance remains unchanged. To prove our main technical lemma we transform any alignment converting $E(X)$ to $E(Y)$ into a block-structured one without significantly increasing its cost.

We will introduce certain notations to facilitate the presentation of the proof. For any $X \in \Gamma^n$ we employ lowercase letters such as: i, j, k etc. to represent indices of X_i. We utilize tuples from $[n] \times [k]$ to represent indices of $E(X)$, where the first index signifies the block index denoted by lowercase letters, and the second one describes the index within the block represented by a lowercase Greek letter.

The following claim, stated without a formal proof, will be useful in the subsequent proof.

▷ Claim 12. For an alignment $\mathcal{A}$ transforming $E(X)$ into $E(Y)$, we have that the set of matching characters has to be monotone, indicating that for $(i, \alpha) < (i', \alpha')$ in lexicographical order if (i, α) is matched to (j, β) and (i', α') to (j', β') by $\mathcal{A}$, we must have: $(j, \beta) < (j', \beta')$.

Given an alignment $\mathcal{A}$, we partition $E(Y)$ into n segments based on its matching blocks in $E(X)$. Define the i-th segment as follows: if no character of the i-th block of $E(X)$ is matched under $\mathcal{A}$, then the i-th segment is empty. Otherwise, let j denote the first block of $E(Y)$ that includes a matching character for one of the characters in the i-th block. If i is the smallest block containing a matching coordinate within j, then the i-th segment starts at $(j, 1)$, otherwise it starts at the first coordinate within the j-th block matching with the i-th block.

The ending point of the segment is defined similarly: Let j' be the initial block of $E(Y)$ containing a match for the $(i+1)$-th block of $E(X)$. If the i-th block does not match any of the j'-th coordinates, then the i-th segment ends at $(j-1, k)$. Otherwise, it ends at the coordinate preceding the first match of $(i+1)$ and j. We define the starting point of the first non-empty segment as $(1, 1)$ and the end point of the last non-empty segment as (n, k). See Figure 2 for an illustration.

Additionally, we define $cost_{\mathcal{A}}(i)$, the cost of the i-th block, as the sum of unmatched coordinates in $E(X)$ within its i-th block and the unmatched coordinates in $E(Y)$ within its i-th segment. For example, in the illustration provided in Figure 2, we have $cost_{\mathcal{A}}(1) = 0 + 3$ since all the characters of the first block of $E(X)$ are matched, and there are 3 unmatched coordinates in the first segment of $E(Y)$. As the decomposition of $E(Y)$ results in disjoint parts, the sum of the costs across the different blocks equals the cost of $\mathcal{A}$, which we denote by: $cost(\mathcal{A})$.

3.2.1.1 Converting Into Block-Structured Alignments

In this section, we introduce an algorithm that takes an arbitrary alignment $\mathcal{A}$ transforming $E(X)$ into $E(Y)$ as input and produces an alignment $\mathcal{A}^*$. The resulting alignment $\mathcal{A}^*$ is block-structured, and its cost does not substantially exceed that of $\mathcal{A}$.

To design the new matching we need few definitions. We say that blocks i and j are *partially matched* by an alignment $\mathcal{A}$ if there exists a pair (i, α) and (j, β) matched by $\mathcal{A}$. Furthermore, i and j are *significantly matched* by $\mathcal{A}$ if more than εk characters in the i-th

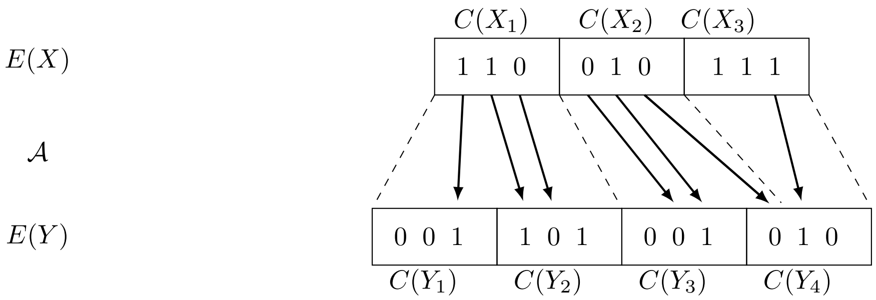


Figure 2 An illustration of the matching between the strings $E(X)$ and $E(Y)$. Arrows indicate matching coordinates, and dashed lines represent the beginning/end points of the segments. The first segment starts at $(1, 1)$ and ends at $(2, 3)$, as the first matched coordinate in the second block of $E(X)$ is mapped to the third block, and no character from the first block of $E(X)$ is mapped to that block. The second segment starts at $(3, 1)$ and ends at $(4, 1)$ (as the first matched coordinate in the second block of $E(X)$ is mapped to the third block, and there exists a character from the second block of $E(X)$ that is mapped to that block).

block of X are matched into the j-th block; we say that i and j are *perfectly matched* if $\mathcal{A}$ matches every character in $E(X)$ into a character in $E(Y)$. The algorithm operates in two stages. In the first stage, the algorithm iteratively takes two significantly matched blocks that are not perfectly matched, removes all matches that the characters of the two blocks participate in, and introduces a perfect match between the two blocks. In the second stage, we remove all the matches from blocks that are not matched perfectly.

A key observation is that if i and j are significantly matched then we have the following inequality: $\Delta_{indel}(C(X_i), C(Y_j)) < (2 - \varepsilon)k$. Therefore, by the distance guarantee regarding C, we must have $X_i = Y_j$. The algorithm is given next. For the sake of the analysis its second stage is divided into two cases.

3.2.1.2 The Correctness of the Algorithm

We break down the proof of correctness into three claims. Claim 13 states that the output produced by Algorithm 1 is a block-structured alignment. The subsequent two claims provide bounds on the cost difference between the input and output alignments.

▷ Claim 13. The alignment $\mathcal{A}'$ produced by Algorithm 1 from any alignment $\mathcal{A}$ converting $E(X)$ into $E(Y)$ is a block-structured alignment converting $E(X)$ into $E(Y)$.

The proof of Lemma 11 is derived from the following claims, let us first state the claims.

▷ Claim 14. Let $\varepsilon < 1/4$ and let $\mathcal{A}$ be any alignment converting $E(X)$ into $E(Y)$. Let $\mathcal{A}_{\text{I}}$ be the resulting alignment obtained by applying the algorithm described in stage I on $\mathcal{A}$ with a proximity parameter of ε. Then,

$$cost(\mathcal{A}_{\text{I}}) \leq (1 + 4\varepsilon)cost(\mathcal{A}).$$

▷ Claim 15. Let $\mathcal{A}_{\text{I}}$ be any resulting alignment obtained by applying the algorithm described in stage I on some alignment $\mathcal{A}$ with a proximity parameter of ε. Let $\mathcal{A}_{\text{II}}$ be the resulting alignment obtained by applying the algorithm described in stage II on $\mathcal{A}_{\text{I}}$ with a proximity parameter of ε. Then,

$$cost(\mathcal{A}_{\text{II}}) \leq (1 + 4\varepsilon)cost(\mathcal{A}_{\text{I}}).$$

The proofs of claims 13, 14 and 15 are given in the full version of the paper.

Algorithm 1 Converting Into Block-Structured Alignments.

```
Data: An alignment $\mathcal{A}$ converting $E(X)$ into $E(Y)$
Result: An alignment $\mathcal{A}'$ that is block-structured, converting $E(X)$ into $E(Y)$
$\mathcal{A}' \leftarrow A$;
;                                                        /* Stage I: */
for $i = 1 \cdots |X|$ do
    if $i$ has some significant match then
        $j \leftarrow$ smallest block in $E(Y)$ that significantly matches $i$;
        Remove all matches incident with blocks $i$ and $j$ from $\mathcal{A}'$, and add a perfect
          match between the two blocks;
    end
end
$i \leftarrow 1$ ;                                               /* Stage II: */
while $i <= |X|$ do
    if $i$ has a partial and not perfect match then
        if $i$ is partially matched to more than a single block then
            Delete from $\mathcal{A}'$ all matches of characters from the $i$-th block of $E(X)$;
            $i++$;
        end
        else
            $j \leftarrow$ smallest $E(Y)$-block that partially matches $i$;
            $i' \leftarrow$ smallest $E(X)$ that does not match with the $j$-th block;
            Delete from $\mathcal{A}'$ all matches of characters from the $j$-th block of $E(Y)$;
            $i \leftarrow i'$;
        end
    end
end
```

Proof of Lemma 11 (using Claim 14 and Claim 15). Let OPT represent the normalized cost of the optimal alignment between X and Y, and $\widetilde{OPT}$ denote the normalized cost of the optimal alignment between $E(X)$ and $E(Y)$. Notice that any alignment between X and Y can be paired with an alignment between $E(X)$ and $E(Y)$ having the same cost. Consequently, we have: $\widetilde{OPT} \leq OPT$. To complete the argument, it remains to establish that: $(1 - 48\varepsilon)OPT \leq \widetilde{OPT}$, which can be achieved by demonstrating: $OPT \leq (1 + 24\varepsilon)\widetilde{OPT}$.

Consider the optimal alignment $\mathcal{A}$ that transforms X into Y, and let $\mathcal{A}'$ be the alignment generated by Algorithm 1 when applied to $\mathcal{A}$. According to Claims 14 and 15, we obtain:

$$\frac{1}{2|E(X)|} \cdot cost(\mathcal{A}') \leq \frac{1}{2|E(X)|} \cdot (1+4\varepsilon)^2 cost(\mathcal{A}) \leq \frac{1}{2|E(X)|} \cdot (1+24\varepsilon) cost(\mathcal{A}) = (1+24\varepsilon)OPT.$$

We conclude the proof by noting that: $\widetilde{OPT} \leq \frac{1}{2|E(X)|} \cdot cost(\mathcal{A}')$. ◀

3.3 Lower Bounds

Proof of Claim 8. For any value of $n \in \mathbb{N}$, define A_n as the set of length n strings composed of a single character from Γ. Clearly, $|A_n| = |\Gamma|$ and moreover, for every distinct pair of strings in A_n, their Indel distance is $2n$.

Now consider any embedding $E : \Gamma^* \to \Sigma^*$. Since $|A_n| = |\Gamma| > |\Sigma|$, by the pigeonhole principle there exist $X \neq Y \in A_n$ satisfying: $E(X)_1 = E(Y)_1$ [2]. Hence, $\Delta_{indel}(E(X), E(Y)) < 2\ell(n)$ while $\Delta_{edit}(X, Y) = 2n$. ◁

Proof of Claim 9.

1. Fix $n \in \mathbb{N}$ and consider any embedding $E : \Gamma^* \to \Sigma^*$. For any $X \in \Gamma^n$, define the value $p(E(X)) \in \Sigma$ as the plurality value among $\{E(X)_i\}_{i\in\mathbb{N}}$ (ties are broken arbitrarily). Observe that the character $p(E(X))$ appears at least $\frac{\ell(n)}{|\Sigma|}$ times in the string $E(X)$. Furthermore, for any $X, Y \in \Sigma^n$ if: $p(E(X)) = p(E(Y))$, then we get: $LCS(E(X), E(Y)) \geq \frac{\ell(n)}{|\Sigma|}$ and hence: $\Delta_{indel}(E(X), E(Y)) \leq \left(1 - \frac{1}{|\Sigma|}\right) 2\ell(n)$.
   As in the proof of Claim 8, define A_n as the set of strings composed of a single character from Γ. Recall that $|A_n| = |\Gamma|$ and moreover, for every distinct pair of points in A_n, their LCS distance is $2n$.
   Since $|A_n| = |\Gamma| > |\Sigma|$, by the pigeonhole principle there exist $X \neq Y \in A_n$ satisfying: $p(E(X)) = p(E(Y))$, yielding: $\Delta_{indel}(E(X), E(Y)) \leq \left(1 - \frac{1}{|\Sigma|}\right) 2\ell(n)$, whereas $\Delta_{indel}(X, Y) = 2n$, as claimed.
2. Let $k = |E(\Lambda)|$. For $Z \in \Gamma^n$, we have: $\Delta_{indel}(E(Z), E(\Lambda)) \geq \ell(n) - k$ so $\widetilde{\Delta}_{indel}(E(Z), E(\Lambda)) \geq 1 - \frac{k}{\ell(n)} \geq 1 - \frac{k}{n}$. On the other hand, $\Delta_{indel}(Z, \Lambda) = n$ so $\widetilde{\Delta}_{indel}(Z, \Lambda) = 1$. ◁

4 Alphabet Reduction – Binary Alphabets

In this section we show a reduction of Δ_{edit} and Δ_{indel} over an arbitrary alphabet to the binary alphabet. The reduction expands the strings super-polynomially, but one can think of it as a proof of concept that more efficient reduction might exist. The main theorem of this section is the following statement which is a formal statement of Theorem 5. For ease of presentation it is beneficial to think about Longest Common Subsequence instead of Δ_{indel}. That is how we state the theorem here.

▶ **Theorem 16.** *For any integer $n \geq 1$, any alphabet Σ of size at most n^3, there exist integers S, R, N where $N = n^{O(\log n)}$ and functions $G, H, G', H' : \Sigma^n \to \{0,1\}^N$ such that for any $X, Y \in \Sigma^n$,*

$$\begin{aligned} LCS(X, Y) &= \frac{LCS(G(X), H(Y)) - R}{S} \\ \Delta_{edit}(X, Y) &= \frac{LCS(G'(X), H'(Y)) - R}{S}. \end{aligned}$$

Hence, for any pair of strings X, Y one can recover $\Delta_{edit}(X, Y)$ from $\Delta_{indel}(G'(X), H'(Y))$ over a binary alphabet. Both mappings G, H and G', H' can be computed efficiently in the length of their output. Indeed, they will be defined explicitly below. We remark that the bound n^3 on the size of Σ is essentially arbitrary and could be replaced for example by a bound 2^n without change in the other parameters (except for multiplicative constants). However, the n^3 bound allows for hashing any large alphabet by a random pair-wise independent hash function to an alphabet of size n^3 without affecting the distance of any given pair of strings except with probability $< 1/n$.

[2] $E(X)_i$ is the i^{th} character of the string $E(X)$.

In order to prove the theorem we will need several auxiliary functions. We say that a 0-1 string is *balanced* if it contains the same number of 0's and 1's. We say a formula ϕ is *normalized* if it consists of alternating layers of binary AND and OR and all of its literals are at the same depth; each literal is either a constant, a variable or its negation.

We define two functions $g, h : \{0,1\}^* \times \{\text{normalized formulas}\} \to \{0,1\}^*$ and two threshold functions $f, t : \{\text{normalized formulas}\} \to \mathbb{N}$ as follows: Let us consider sets of variables $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$, and let $A = \{a_1, \ldots, a_p\}$ where a_i is the assignment to the variable u_i for all $1 \leq i \leq p$, and $B = \{b_1, \ldots, b_q\}$ where b_i is the assignment to the variable v_i for all $1 \leq i \leq q$.

Let $\phi(U, V)$ be a normalized formula which is defined over two disjoint sets of variables $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$. Let $A \in \{0,1\}^p, B \in \{0,1\}^q$ where A and B are interpreted as assignments for U and V respectively. We define two functions g, h, such that g gets as an input a pair (ϕ, A) and outputs a string in $\{0,1\}^*$, similarly h takes a pair (ϕ, B) as its input and outputs a string in $\{0,1\}^*$. We also define threshold functions f, t which take such a formula as input and output a natural number. The crux of the construction is that for any assignment A for U and B for V we have that if ϕ is satisfied by the assignment pair A, B then $LCS(g(\phi, A), h(\phi, B)) = t(\phi)$, otherwise $LCS(g(\phi, A), h(\phi, B)) = f(\phi)$

We establish the recursive definitions of g, h, f and t based on the depth of the formula. The base case is when ϕ is either a constant $0, 1$ or single literals $u_i, \neg u_i, v_j, \neg v_j$, where, $u_i \in U, v_j \in V$. Here by $\neg 0$ we understand symbol 1, and similarly by $\neg 1$ we understand symbol 0.

	$\phi = u_i$	$\phi = \neg u_i$	$\phi = v_j$	$\phi = \neg v_j$	$\phi = 1$	$\phi = 0$
$g(A, \phi)$	$\neg a_i a_i$	$a_i \neg a_i$	0 1	0 1	0 1	1 0
$h(B, \phi)$	0 1	0 1	$\neg b_j b_j$	$b_j \neg b_j$	0 1	0 1
$t(\phi)$	2	2	2	2	2	2
$f(\phi)$	1	1	1	1	1	1

and further inductively:

	$\phi = \phi_0\ OR\ \phi_1$	$\phi = \phi_0\ AND\ \phi_1$
$g(A, \phi)$	$1^{k/2}\ 1^{4k}\ g(A, \phi_0)\ 1^{4k}\ 0^{4k}\ g(A, \phi_1)\ 0^{4k}\ 0^{k/2}$	$0^{T+F}\ 1^{11k+T+F}\ 0^{5k}\ g(A, \phi_0)\ 0^k\ 1^k\ 0^k\ g(A, \phi_1)\ 0^{5k}$
$h(B, \phi)$	$0^{k/2}\ 0^{4k}\ h(B, \phi_0)\ 0^{4k}\ 1^{4k}\ h(B, \phi_1)\ 1^{4k}\ 1^{k/2}$	$0^{T+F}\ 0^{5k}\ h(B, \phi_0)\ 0^k\ 1^k\ 0^k\ h(B, \phi_1)\ 0^{5k}\ 1^{11k+T+F}$
$t(\phi)$	$9k + T$	$13k + 3T + F$
$f(\phi)$	$9k + F$	$13k + 2T + 2F.$

where $k = |g(x, \phi_0)|$, $T = t(\phi_0)$, and $F = f(\phi_0)$.

Key properties of our functions are summarized in the next lemma.

▶ **Lemma 17.** *Let $\phi(U, V)$ be a balanced formula of depth d with set of variables $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$. For every two assignments $A, A' \in \{0,1\}^p$ to variables U, we have $|g(A, \phi)| = |g(A', \phi)|$. Similarly, for every two assignments $B, B' \in \{0,1\}^q$ to variables V, $|h(B, \phi)| = |h(B', \phi)|$. Additionally, $|g(A, \phi)| = |h(B, \phi)| \leq 30^d$.*

Furthermore, the following holds:

- *If* $\phi(A, B)$ *is true then* $LCS(g(A, \phi), h(B, \phi)) = t(\phi)$.
- *If* $\phi(A, B)$ *is false then* $LCS(g(A, \phi), h(B, \phi)) = f(\phi)$.

Finally, $f(\phi) < t(\phi)$.

In order to prove the above lemma we also need two gadgets which we call the AND-gadget and the OR-gadget. We need the lemmas on these gadgets (statement and proofs included in the full version of the paper) which analyze the composition of AND and OR.

The proofs of theorem 16 and lemma 17 can also be found in the full version of the paper.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 59–78, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. 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 8:1–8:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.8.

3 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 990–1001. IEEE, 2020. doi:10.1109/FOCS46700.2020.00096.

4 Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 199–204, New York, NY, USA, 2009. ACM. doi:10.1145/1536414.1536444.

5 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746612.

6 Ziv Bar-Yossef, TS Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 550–559. IEEE, 2004.

7 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 316–324, New York, NY, USA, 2003. ACM. doi:10.1145/780542.780590.

8 Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 792–801, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

9 Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. *Journal of the ACM (JACM)*, 68(3):1–41, 2021. doi:10.1145/3456807.

10 Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 685–698. ACM, 2020. doi:10.1145/3357713.3384282.

11 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Kouckỳ, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. *Journal of the ACM (JACM)*, 67(6):1–22, 2020. doi:10.1145/3422823.

12 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 712–725, 2016. doi:10.1145/2897518.2897577.

13 E. N. Gilbert. A comparison of signalling alphabets. *The Bell System Technical Journal*, 31(3):504–522, 1952. doi:10.1002/j.1538-7305.1952.tb01393.x.

14 Elazar Goldenberg, Aviad Rubinstein, and Barna Saha. Does preprocessing help in fast sequence comparisons? In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 657–670, 2020. doi:10.1145/3357713.3384300.

15 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016. doi:10.1016/J.DAM.2015.10.040.

16 Michal Koucký and Michael E. Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 699–712. ACM, 2020. doi:10.1145/3357713.3384307.

17 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM J. Comput.*, 27(2):557–582, April 1998. doi:10.1137/S0097539794264810.

18 Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10(8), pages 707–710. Soviet Union, 1966.

19 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

20 Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5):23, 2007. doi:10.1145/1284320.1284322.

21 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1:571–603, 2008. doi:10.1007/S11786-007-0033-3.

22 Rom Rubenovich Varshamov. Estimate of the number of signals in error correcting codes. *Docklady Akad. Nauk, SSSR*, 117:739–741, 1957.

23 Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974. doi:10.1145/321796.321811.

Parallel Complexity of Geometric Bipartite Matching

Sujoy Bhore
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Sarfaraz Equbal
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Rohit Gurjar
Department of Computer Science & Engineering, Indian Institute of Technology Bombay, India

Abstract

In this work, we study the parallel complexity of the geometric minimum-weight bipartite perfect matching (GWBPM) problem in $\mathbb{R}^2$. Here our graph is the complete bipartite graph G on two sets of points A and B in $\mathbb{R}^2$ ($|A| = |B| = n$) and the weight of each edge $(a, b) \in A \times B$ is the ℓ_p distance (for some integer $p \geq 2$) between the corresponding points, i.e., $||a - b||_p$. The objective is to find a minimum weight perfect matching of $A \cup B$. In their seminal work, Mulmuley, Vazirani, and Vazirani (STOC 1987) showed that the weighted perfect matching problem on general bipartite graphs is in RNC. Almost three decades later, Fenner, Gurjar, and Thierauf (STOC 2016) showed that the problem is in Quasi-NC. Both of these results work only when the weights are of $O(\log n)$ bits. It is a long-standing open question to show the problem to be in NC.

First, we show that in a geometric bipartite graph under the ℓ_p metric for any $p \geq 2$, unless we take $\Omega(n)$ bits of approximation for weights, we cannot distinguish the minimum-weight perfect matching from other perfect matchings. This means that we cannot hope for an MVV-like NC/RNC algorithm for solving GWBPM exactly (even when vertex coordinates are small integers).

Next, we give an NC algorithm (assuming vertex coordinates are small integers) that solves GWBPM up to $1/\text{poly}(n)$ additive error, under the l_p metric for any p ≥ 2.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization; Theory of computation → Computational geometry

Keywords and phrases Parallel algorithms, Geometric matching, Derandomization, Isolation Lemma

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.12

Funding *Rohit Gurjar*: supported by SERB MATRICS grant MTR/2022/001009.

1 Introduction

The perfect matching problem is one of the well-studied problems in Complexity theory, especially, in the context of derandomization and parallelization. Given a graph $G = (V, E)$, the problem asks, whether the graph contains a matching that matches every vertex of G. Due to Edmonds [12], the problem is known to be solvable in polynomial time. However, the parallel complexity of the problem has not been completely resolved till today. In 1979, Lovász [19] showed that perfect matching can be solved by efficient randomized parallel algorithms, i.e., the problem is in RNC. Hence, the main question, with respect to its parallel complexity, is whether this randomness is necessary, i.e., whether the problem is in NC[1].

[1] The class NC represents the problems that have efficient parallel algorithms, i.e., they have uniform circuits of polynomial size and polylog depth

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 12; pp. 12:1–12:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The search version of the problem asks to explicitly construct a perfect matching in a graph if one exists. Note that in the parallel setting, there is no obvious reduction from search to decision. This version is also known to be in RNC [18, 21]. The Mulmuley-Vazirani-Vazirani (MVV) algorithm [21], in fact, also works for the weighted version of the problem, where there is a polynomially bounded weight assignment given on the edges of the graph.

The MVV algorithm [21] introduced the celebrated *Isolation lemma.* A weight assignment is called *isolating* for a graph G, if the minimum weight perfect matching in G is unique if one exists. Mulmuley, Vazirani, and Vazirani [21] showed that given an isolating weight assignment with polynomially bounded integer weights for a graph G, a perfect matching in G can be constructed in NC. The only place where they use randomization is to get an isolating weight assignment. Their Isolation lemma states that a random weight assignment is isolating!

Derandomizing the Isolation lemma means to construct such a weight assignment deterministically in NC. A line of work derandomized the Isolation Lemma for special families of graphs, e.g., planar bipartite graphs [11, 27], strongly chordal graphs [10], graphs with a small number of perfect matchings [16]. In 2016, Fenner, Gurjar, and Thierauf [14] showed that the bipartite perfect matching problem is in quasi-NC, by an almost complete derandomization of the Isolation Lemma. Later, Svensson and Tarnawski [26] showed that the problem in general graphs is also in Quasi-NC. Subsequently, Anari and Vazirani [5] gave an NC algorithm for finding a perfect matching in general planar graphs. All of these algorithms work for the weighted version (poly-bounded) of the problem as well.

What remains a challenging open question is to find an NC algorithm for any versions (decision/search/weighted) of the perfect matching problem, even for bipartite graphs. Inspired by the positive results on planar bipartite graphs, we investigate the weighted version of the perfect matching problem in the geometric setting (2 dimensional).

Geometric Bipartite Matching

Let A and B be two point sets in $\mathbb{R}^2$ of size n each. Consider the **complete** bipartite graph $G(A, B, E)$ with the following cost function on the edges: for any edge $e = (a, b)$, define $\mathcal{C}(e) = ||a - b||_p$, where $|| \cdot ||_p$ denotes the ℓ_p norm for some integer $p \geq 2$. In other words, we consider the ℓ_p distance between the endpoints as the cost of an edge. The cost of a perfect matching M is the sum of its edge costs $\mathcal{C}(M) = \Sigma_{e \in M} \mathcal{C}(e)$. The Geometric Minimum-Weight Bipartite Perfect Matching (GWBPM) problem is to find $M_{opt} = \text{argmin}_{|M|=n} \mathcal{C}(M)$, that is, the optimal perfect matching with respect to function $\mathcal{C}$. GWBPM is a fundamental problem in Computational Geometry and has been studied extensively over the years. See Section 1.2 for an overview of the results. In this work, we focus on the parallel complexity of the GWBPM problem, and ask the following question –

▶ **Question 1.** *Is GWBPM in NC?*

Optimization problems in computational geometry are usually studied in real arithmetic computational model, where comparing two distances or sums of distances is assumed to be a unit cost operation. However, in the bit complexity model, it is not clear if distances, which can be irrational numbers (under ℓ_p metric for $p \geq 2$), can be efficiently added or compared. In fact, the problem of comparing two sums of square roots (or other pth roots) is not known to be in P (see, for example, [22, 1]). See [13] for some recent progress on the sum of square roots problem.

Our path towards showing an NC algorithm for GWBPM naturally goes via the MVV algorithm. Recall that the MVV algorithm works only when the given weights/costs are polynomially bounded integers, because in intermediate steps, it needs to put weights in the exponent. Hence, inevitably we need to consider the bit complexity of the weights. Note that there are other parallel algorithms for the weighted perfect matching problem (e.g., [15]), but there too it is important that the weights are polynomially bounded integers.

It is not clear if the GWBPM problem (for $p \geq 2$) is in P (or even in NP) in the bit-complexity model. To the best of our knowledge, the existing algorithms for GWBPM require comparisons between two sums of pth roots. For $p \geq 2$, this naturally leads us to consider an approximate version of the problem. Let us define the δ-GWBPM problem, which asks for a perfect matching whose weight is at most δ more than the minimum-weight perfect matching. We aim to get an NC algorithm for the problem whenever $1/\delta$ is $\operatorname{poly}(n)$.

1.1 Our Contribution

In this work, we study the parallel complexity of δ-GWBPM problem. First, it is natural to ask whether solving δ-GWBPM for some $\delta = 1/\operatorname{poly}(n)$ will already solve the GWBPM problem. In other words, by considering only $O(\log n)$ bit approximations of ℓ_p distances, can we hope to find the geometric minimum weight perfect matching? Our first result rules out this possibility. We show that for GWBPM (under ℓ_p metric for any $p \geq 2$), a super-linear number of bit approximations is required to distinguish the minimum-weight perfect matching from others.

▶ **Theorem 1.1.** *There is a set of $2n$ points in the $O(n^7) \times O(n^7)$ integer grid such that in the corresponding complete bipartite graph, the difference between the weights of the minimum weight perfect matching and another perfect matching is at most $1/(n-1)!$ (under any ℓ_p metric with $p \geq 2$).*

This theorem is proved in Section 2. The first part of the proof goes via a known counting technique [8], where we construct a geometric bipartite graph and argue that there must be two perfect matchings whose weights are distinct but very close. In the second part of the proof, we construct another geometric bipartite graph based on these two matchings, where one of the two is the minimum weight perfect matching.

Next, we come to our positive result. We affirmatively answer Question 1, by showing that the geometric minimum weight perfect matching problem that allows up to $\frac{1}{\operatorname{poly}(n)}$ error, is in NC.

▶ **Theorem 1.2.** *The δ-GWBPM under ℓ_p metric ($p \geq 2$) is in NC, assuming the points are on a polynomially bounded integer grid, where δ is $1/\operatorname{poly}(n)$.*

This theorem is proved in Section 3. The main idea is to reduce the problem to bipartite planar matching and then use known techniques for the planar case [27].

1.2 Related Work

The classical Hopcroft-Karp algorithm computes a maximum-cardinality matching in a bipartite graph with n vertices and m edges in $O(m\sqrt{n})$ time [17]. After almost three decades, Madry [20] improved the running time to $O(m^{10/7} \operatorname{polylog} n)$ time, which was further improved to $O(m + n^{3/2} \operatorname{polylog} n)$ by Brand et al. [29]. The Hungarian algorithm computes the minimum-weight maximum cardinality matching in $O(mn + n^2 \log n)$ time [23]. In some recent breakthrough results [28, 9], they have shown that maximum-cardinality matching in bipartite graphs can be solved in near-linear time.

For two sets of points A and B in $\mathbb{R}^2$, the best known algorithm for computing GWBPM runs in $O(n^2 \operatorname{polylog} n)$ time [3, 4]. Moreover, if points have integer coordinates bounded by Δ, the running time can be improved to $O(n^{3/2} \operatorname{polylog} n \log \Delta)$ [24]. If coordinates of input points have real values, it is not known whether a subquadratic algorithm exists. However, for the non-bipartite case, Varadarajan [30] presented an $O(n^{3/2} \operatorname{polylog} n)$-time algorithm under any ℓ_p-norm. For bipartite matching, a large body of literature focused on obtaining approximate matching for points in $\mathbb{R}^d$. Varadarajan and Agarwal [31] presented an $O(n^{3/2}\varepsilon^{-d} \log^d n)$-time ε-approximation algorithm for geometric matching in $\mathbb{R}^d$. Later, Agarwal and Raghavendra [25] improved the running time. Recently, Agarwal et al. [2] presented a deterministic algorithm with running time $n \cdot (\varepsilon^{-1} \log n)^{O(d)}$ time, and computes a perfect matching whose cost is within a $(1 + \varepsilon)$ factor of the optimal matching under any ℓ_p-norm.

2 Lower Bound

In this section, we want to show that for a geometric bipartite graph with $n + n$ vertices, we need at least $\Omega(n \log n)$ bits of precision to distinguish the minimum weight perfect matching from others (under ℓ_p metric for any integer $p \geq 2$). We will show this by constructing a bipartite set of $2n$ points in the integer grid of size $O(n^7) \times O(n^7)$ such that the difference between the weights of the minimum weight perfect matching and the one with the next higher weight will be $1/(n-1)!$. Towards this, the first step is to construct a geometric graph where there are two perfect matchings whose weights differ by at most $1/(n-1)!$ (Claim 2.2). Here we use an argument based on the pigeonhole principle. A similar argument was used to show such a bound on the difference of two sums of square roots [8].

In the above construction, it is not necessary that one of the two perfect matchings is of minimum weight. In the second step, we show that the above geometric graph can be modified to construct another one where the same two perfect matchings appear, but now one of them is of minimum weight (Claim 2.4).

▶ **Construction 1.** *Consider the left hand side vertices $u_0, u_1, \ldots, u_{n-1}$ at points*

$$\{(0, 0), (0, 1), (0, 2), \ldots, (0, n-1)\}.$$

Similarly, consider the right hand side vertices $v_0, v_1, \ldots, v_{n-1}$ at points

$$\{(q, n), (q, 2n), (q, 3n), \ldots, (q, n^2)\},$$

where $q = n^6$.

▷ Claim 2.1. The geometric bipartite graph in Construction 1 has all its perfect matchings with distinct weights (under ℓ_p metric for any integer $p > 1$).

Proof. Recall that edge weights are pth roots of integers. We will argue that the edge weights are linearly independent over rationals, which immediately implies that any two different subsets of edges cannot have equal weights. It is known that to show linear independence of a set of pth roots of integers, it suffices to show that they are *pairwise* linearly independent (see, for example, [7]). So, now we just argue that the edge weights are pairwise linearly independent.

First we observe that none of the edge weights is an integer. This is because from our construction, we have $n^6 < \sqrt[p]{n^{6p}+1} < w(e) \leq \sqrt[p]{n^{6p}+n^{2p}} < n^6+1$.

For the sake of contradiction, suppose we have two edges e and e', whose weights are linearly dependent. Then we have $aw(e) = bw(e')$ for some integers a and b. From here we get that $w(e)^p w(e')^p = (a/b)^p w(e)^{2p}$. That is, the product $w(e)^p w(e')^p$ is pth power of a rational number. Since it is an integer, it must be pth power of an integer. From our construction, for any edge e, we have $q < w(e) \leq \sqrt[p]{q^p + n^{2p}}$. Moreover, note that only one of the edges e or e' can match the upper bound. Hence,

$$(q^2)^p < w(e)^p w(e')^p < (q^p + n^{2p})^2. \tag{1}$$

Now, we consider two cases $p \geq 3$ and $p = 2$.

Case I ($p \geq 3$). As $w(e)^p w(e')^p$ is pth power of an integer, from Equation (1) we have

$$(q^2+1)^p \leq w(e)^p w(e')^p < (q^p + n^{2p})^2.$$

Comparing the first and the last terms, we get

$$pq^{2p-2} + \binom{p}{2} q^{2p-4} + \cdots + 1 < 2q^p n^{2p} + n^{4p}.$$

Putting $q = n^6$, we see that the above inequality is false. Hence, we get a contradiction.

Case II ($p = 2$). From Equation 1, we have

$$q^4 < w(e)^2 w(e')^2 < (q^2 + n^4)^2.$$

Since $w(e)^2 w(e')^2$ is square of an integer, we can write $w(e)^p w(e')^p = (q^2+\alpha)^p$ for some integer $0 < \alpha < n^4$.

For any edge e, let us denote by Δ_e, the difference in the y coordinates of the two endpoints of the edge. Then, the weight of an edge e can be written as $w(e) = \sqrt{q^2 + \Delta_e^2}$. Now, we have

$$w(e)^2 w(e')^2 = (q^2 + \Delta_e^2)(q^2 + \Delta_{e'}^2) = (q^2 + \alpha)^2.$$

Equivalently,

$$q^2(\Delta_e^2 + \Delta_{e'}^2) + \Delta_e^2 \Delta_{e'}^2 = 2q^2\alpha + \alpha^2.$$

Observe that $\Delta_e^2 \Delta_{e'}^2 \leq n^8 < q^2$ (from construction) and also $\alpha^2 < n^8 < q^2$. Hence, we conclude from above that

$$\Delta_e^2 + \Delta_{e'}^2 = 2\alpha \text{ and } \Delta_e^2 \Delta_{e'}^2 = \alpha^2.$$

This implies that $\Delta_e = \Delta_{e'}$.

Now, we will argue that for any two distinct edges, we have $\Delta_e \neq \Delta_{e'}$, which will give us a contradiction. Indeed for the edge (u_i, v_j), we have $\Delta_e = jn - i$, which comes from a unique choice of $0 \leq i \leq n-1$ and $1 \leq j \leq n$. ◁

▷ Claim 2.2. In the geometric bipartite graph from Construction 1, there are two perfect matchings whose weights are different and differ by at most $1/(n-1)!$.

Proof. From Claim 2.1, all $n!$ perfect matchings have distinct weights. From the construction, any perfect matching has its weight between n^7 and $n\sqrt[p]{n^{6p}+n^{2p}} \leq n(n^6+1)$. The bound follows from the pigeonhole principle. ◁

Now, consider the two perfect matchings from Claim 2.2, say M_1 and M_2, whose weights differ by at most $1/(n-1)!$. Let M_1 be the one with a smaller weight. The union of two perfect matchings $M_1 \cup M_2$ is a set of vertex-disjoint cycles and edges. We are going to ignore the common edges between M_1 and M_2. Let $(e_1, e_2, \ldots, e_{2\ell})$ be the sequence of edges generated from the cycles in $M_1 \cup M_2$ as follows: arrange the cycles in an arbitrary order. For each cycle, start from that edge in M_1 which has its left endpoint with minimum y-coordinate, and traverse along the cycle till we hit the starting vertex. Note that the sequence $(e_1, e_2, \ldots, e_{2\ell})$ has edges alternating from M_1 and M_2. The new graph will be constructed by "unrolling" these cycles. The construction will be such that edges outside these cycles will be long, and hence, will not be a part of any minimum weight perfect matching. Recall that for any edge $e = (u_i, v_j)$ (Construction 1), we denote by Δ_e the difference in the y-coordinates of the endpoints, i.e., $jn - i$.

▶ **Construction 2.** *Consider the vertex t_0 at $(0,0)$. Let $y_0 = 0$. For $1 \leq k \leq 2\ell$, we place the vertex t_k at (kq, y_k), where*

- $y_k = y_{k-1} + \Delta_{e_k}$ *if k is odd*
- $y_k = y_{k-1} - \Delta_{e_k}$ *if k is even*

We add three more vertices: s_0 at $(0, -2\ell q)$, s_1 at $(\ell q, -2\ell q)$, and s_2 at $(2\ell q, -2\ell q)$. See Figure 1.

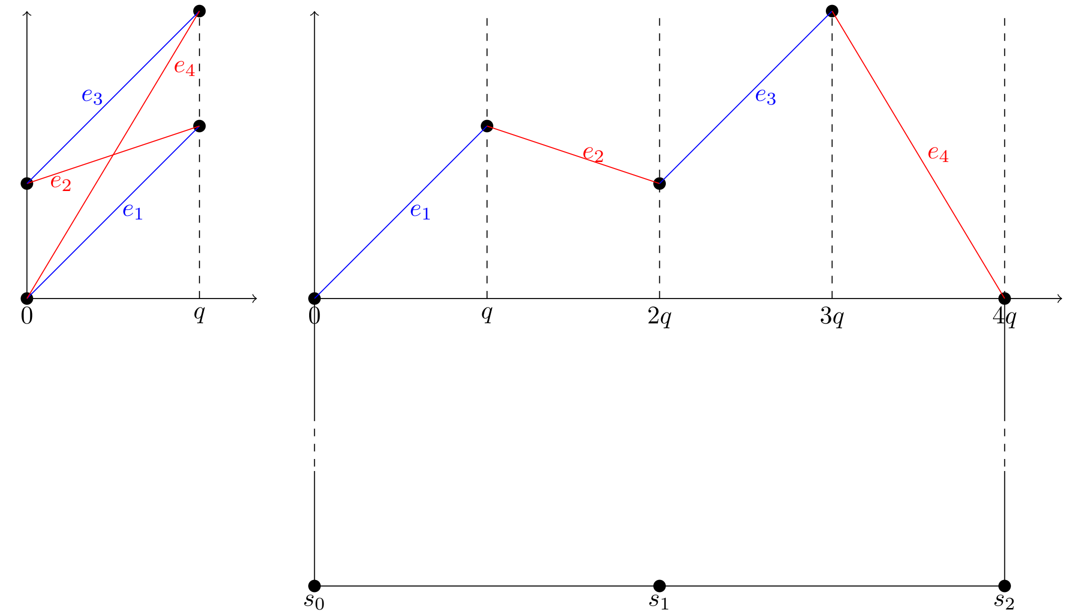


Figure 1 The left-hand side figure shows a cycle in the union of two perfect matchings. The right-hand side figure shows how we "unroll" this cycle.

Corresponding to perfect matchings M_1 and M_2, here we will have perfect matchings M_1' and M_2' as

$$M_1' = \{e_1, e_3, \ldots, e_{2\ell-1}, (t_{2\ell}, s_2), (s_0, s_1)\}$$

$$M_2' = \{e_2, e_4, \ldots, e_{2\ell}, (t_0, s_0), (s_1, s_2)\}.$$

The following are easy observations about Construction 2.

1. The edge lengths of $e_1, e_2, \ldots, e_{2\ell}$ are exactly the same as their lengths in Construction 1.
2. $y_k \geq 0$ for each $1 \leq k \leq 2\ell$, because for each cycle, the cycle traversal starts from the lowest y coordinate on the left. Moreover, $y_{2\ell}$ must be zero, because any cycle traversal ends at the starting vertex.
3. Any pair of vertices are at least distance q apart.
4. $w(M_1') = w(M_1) + 3\ell q$ and $w(M_2') = w(M_2) + 3\ell q$.

▷ Claim 2.3. The minimum weight perfect matching in Construction 2 is M_1', with weight $w(M_1) + 3\ell q$.

Proof. Recall that weight of any edge e_k is at most $\sqrt[p]{q^p + n^{2p}} = \sqrt[p]{n^{6p} + n^{2p}} < n^6 + 1/(pn^{4p-6}) < q + 1/(pn^{4p-6})$. Hence, $w(M_1') \leq \ell(q + 1/(pn^{4p-6})) + 3\ell q = \ell(4q + 1/(pn^{4p-6}))$. We have already assumed that M_2' has weight higher than M_1'. Now, consider any perfect matching M other than M_1' and M_2'. We will consider different cases and argue that in each case M has a larger weight.

- If M matches s_1 with one of the t_k vertices, the weight of that edge will be at least $2\ell q$. The vertices s_0 and s_2 will either match with each other or to some t_k vertices. In either case, they will contribute at least $2\ell q$ to the weight. The remaining vertices must have at least $\ell - 3$ edges, each with weight at least q. Hence, the total weight will be at least $5\ell q - 3q$, which is larger than $w(M_1')$.
- Consider the case when M has (s_1, s_2) (the other case is similar) and s_0 is matched with one of the t_k vertices, other than t_0. Recall that $y_k \geq 0$ and $s_0 = (0, -2\ell q)$. Then the weight of (s_0, t_k) (for $k > 0$) is at least $\sqrt[p]{(2\ell q)^p + q^p} \geq 2\ell q + 2\ell q/(4\ell)^p$. The remaining 2ℓ vertices will have ℓ matching edges, each with weight at least q. Hence, the weight of the matching M will be at least $\ell q + 2\ell q + 2\ell q/(4\ell)^p + q\ell$. This is larger than $w(M_1') \leq 4\ell q + \ell/(pn^{4p-6})$ (as $q = n^6$ and $\ell \leq n$).
- Consider the case when M has (s_1, s_2) and (s_0, t_0). These two edges will add up to weight $3\ell q$. Since the matching M is different from M_1' and M_2', it must match a vertex t_k with another vertex t_j such that $j \neq \{k-1, k+1\}$. Then $|j - k|$ must be at least 3, because the graph is bipartite. The edge (t_k, t_j) will have weight at least $3q$. The other $\ell - 1$ edges will have weight at least q. Hence, the total weight is at least $4\ell q + 2q$, which is again larger than $w(M_1')$.
- The other cases when M has (s_0, s_1) matched are similar to the above two cases. ◁

Now, we finally come to our main claim.

▷ Claim 2.4. In the geometric bipartite graph from Construction 2, the difference between the minimum weight perfect matching and the perfect matching with the next higher weight is at most $1/(n-1)!$.

Proof. From Claim 2.3, we know that M_1' is the minimum weight perfect matching. We had observed that $w(M_1') - w(M_2') = w(M_1) - w(M_2)$. From Claim 2.2, this difference is at most $1/(n-1)!$. ◁

3 Geometric Bipartite Matching

In this section, we study the parallel complexity of δ-GWBPM (under ℓ_p metric for $p \geq 2$), and show that the problem lies in the class NC, for $\delta = 1/\operatorname{poly}(n)$.

First of all, we assume that no three vertex points are colinear. There is a simple fix to break colinearity by way of small perturbations in coordinates. Specifically, for the ith vertex at point (x_i, y_i), let us assign its new coordinates to be $(x_i + i/K, y_i + i^2/K)$, where

K is a large enough number. This specific perturbation guarantees that no three points are colinear. To see this, consider ith, jth, and kth vertices after the perturbation. They will be colinear if and only if the following matrix has zero determinant.

$$\begin{pmatrix} 1 & 1 & 1 \\ x_i + i/K & x_j + j/K & x_k + k/K \\ y_i + i^2/K & y_j + j^2/K & y_k + k^2/K \end{pmatrix}$$

Consider the coefficient of the term $1/K^2$ in the determinant, which is $(i-j)(j-k)(k-i) \neq 0$. Other terms in the determinant will be an integer multiple of $1/K$ and hence, cannot cancel this term, when K is large enough $(\operatorname{poly}(n))$. This perturbation can cause additive error in the weights of perfect matchings, but the error will remain bounded by $O(n^3/K)$. Thus, the minimum weight perfect matching with respect to perturbed coordinates will be an GWBPM up to a $1/\operatorname{poly}(n)$ additive error. To make the coordinate integral, we can multiply them by K. Now, give a brief overview of our ideas.

Our main idea is to design an isolating weight assignment for the given graph and then use the MVV algorithm. Let G be a complete bipartite graph of two sets of points A and B in $\mathbb{R}^2$. The MVV theorem asserts that if a graph has an isolating weight assignment, then the task of finding the minimum weight perfect matching in G can be accomplished in NC.

To construct an isolating weight assignment, we adopt the weight scheme introduced by Tewari and Vinodchandran [27], which was designed specifically for planar bipartite graphs. However, note that our graph is a complete bipartite graph and hence, far from planar. Our first key observation is that the union of minimum weight perfect matchings (with respect to ℓ_p distances or even approximate ℓ_p distances) forms a planar subgraph. Then one can hope to use the Tewari and Vinodchandran [27] weight scheme on this planar subgraph. However, we cannot compute this planar subgraph (i.e., the union of minimum weight perfect matchings). What proves to be useful is the fact that the Tewari-Vinodchandran weight scheme is black-box, i.e., it does not care what the underlying planar graph is, it only needs to know the points in the plane where vertices are situated. Finally, we combine the approximate distance function with the Tewari-Vinodchandran weight function on a smaller scale and apply it to the complete bipartite graph. We show that this combined weight function is indeed isolating.

Towards showing the planarity of the union of minimum weight perfect matchings, first, we establish the simple fact that for any convex quadrilateral, there is a significant difference between the sum of diagonals and the sum of any opposite sides.

▶ **Lemma 3.1.** *Consider a convex quadrilateral formed by a quadruple in an integer grid of size $N \times N$. The sum of lengths of its diagonals is larger than the sum of any two opposite sides. And the gap between the two sums in ℓ_p metric is at least $\frac{1}{4\sqrt[p]{2}N^{p^4-1}}$ $(p \geq 2)$.*

Proof. Intuitively, the sum of the diagonals will be larger than the sum of any two opposite sides because of triangle inequality (the diagonals combined with any two opposite edges form two triangles). The significance of this gap arises from the fact that if the points are from a grid and are not collinear, then the angle between any side and the diagonal cannot be arbitrarily small. Formally, let the four corners of the quadrilateral be A, B, C, D (in cyclic order). See Figure 2. Let O be the intersection point of the diagonals AC and BD (diagonals always intersect in a convex quadrilateral). By triangle inequality, we have $|AO| + |OB| \geq AB$ and $|CO| + |OD| \geq CD$. Adding the two we get,

$$|AC| + |BD| \geq |AB| + |CD|.$$

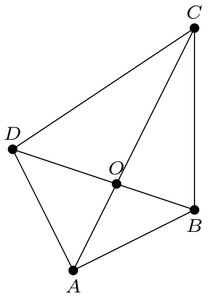


Figure 2 A convex quadrilateral with its two diagonals.

Now, we lower bound the gap. To calculate the lower bound of the gap we directly use the result of [6]. They gave an easy way to calculate the lower bound for an arithmetic expression over operators $+, -, *, /$ and $\sqrt[p]{}$, with integer operands. Our aim to lower bound the gap between, $|AC| + |BD|$ and $|AB| + |CD|$. Let us find the expression for the same where A, B, C, and D are the points on the $N \times N$ grid from $\mathbb{R}^2$. Let the co-ordinates of the points are $(i, j), (k, \ell), (m, n)$, and (o, p) respectively. Then the expression we want to lower bound is,

$$E = ||AC||_p + ||BD||_p - ||AB||_p - ||CD||_p.$$

$$E = \sqrt[p]{(m-i)^p + (n-j)^p} + \sqrt[p]{(o-k)^p + (p-l)^p} - \sqrt[p]{(k-i)^p + (l-j)^p} - \sqrt[p]{(o-m)^p + (p-n)^p}.$$

Our expression also uses only $+, -, *$ and $\sqrt[p]{}$ operators over the integer operands, we can use the Corollary 2 from [6]. It says that for any division-free expression E whose value ξ is nonzero, we have

$$(u(E)^{D(E)-1})^{-1} \leq |\xi| \leq u(E).$$

Here $u(E)$ represents the upper bound on the absolute value of E and $D(E)$ represents the product of indices of all the radicals involved in E. The detailed methodology for calculating $u(E)$ and $D(E)$ can be found in [6]. The values of $u(E)$ and $D(E)$ for our specific case turn out to be as follows:

$$u(E) = 4\sqrt[p]{2}N,$$

$$D(E) = p^4.$$

So the value of the expression E i.e ξ is bounded by,

$$\frac{1}{4\sqrt[p]{2}N^{p^4-1}} \leq |\xi| \leq 4\sqrt[p]{2}N$$

The statement of our Lemma 3.1 easily follows from this. ◀

3.1 Union of Near-Minimum Weight Perfect Matchings

In this subsection, we establish our main lemma that in a geometric bipartite graph G, the union of near-minimum weight perfect matchings forms a planar subgraph of G. This allows us to use the Tewari and Vinodchandran [27] isolating weight scheme for planar bipartite graphs. We first define a near-minimum weight perfect matching.

▶ **Definition 3.2.** *Let the vertices of the geometric bipartite graph lie in the $N \times N$ integer grid. A perfect matching is said to be of near-minimum weight under the ℓ_p metric if its weight is less than $w^* + 1/(8\sqrt[p]{2}N^{p^4-1})$, where w^* is the minimum weight of a perfect matching.*

▶ **Lemma 3.3.** *For a geometric bipartite graph G with vertices in the $N \times N$ integer grid, the union of near-minimum weight perfect matchings forms a planar graph (under ℓ_p metric for any $p \geq 2$).*

Proof. Let $A \cup B$ be the bipartition of the vertices. We will first show that no two edges in a near-minimum weight perfect matching M cross each other. For the sake of contradiction, let there be two edges $\{a_1, b_1\}$ and $\{a_2, b_2\}$ in M that cross each other, where $a_1, a_2 \in A$ and $b_1, b_2 \in B$. Since G is a complete bipartite graph, every vertex of set A must have an edge to every vertex of set B in G. We can construct another matching M' from M by replacing the crossing edges $\{a_1, b_1\}$ and $\{a_2, b_2\}$ with $\{a_1, b_2\}$ and $\{a_2, b_1\}$, respectively.

Note that (a_1, a_2, b_1, b_2) form a convex quadrilateral, since its diagonals a_1b_1 and a_2b_2 cross each other. From Lemma 3.1, we know that

$$|a_1b_2| + |a_2b_1| \leq |a_1b_1| + |a_2b_2| - 1/(4\sqrt[p]{2}N^{p^4-1}).$$

From here, we can conclude that $w(M') \leq w(M) - 1/(4\sqrt[p]{2}N^{p^4-1})$, where $w(M')$ and $w(M)$ are the weights of M' and M, respectively. This contradicts the fact that M is a near-minimum weight perfect matching.

Now, we will show that two edges belonging to two different near-minimum weight perfect matchings cannot cross. Consider two such near-minimum weight perfect matchings M_1 and M_2, where the edges $\{a_1, b_1\} \in M_1$ and $\{a_2, b_2\} \in M_2$ cross each other. Observe that the union of these two perfect matchings forms a set of vertex-disjoint cycles and a set of disjoint edges (which are common to both). There are two cases: (i) the edges $\{a_1, b_1\}$ and $\{a_2, b_2\}$ are part of one of these cycles and (ii) they are part of two different cycles. In each of the cases, we will create two new perfect matchings with significantly smaller weight, which will contradict the near-minimumness of M_1 and M_2.

Case (i): $\{a_1, b_1\}$ and $\{a_2, b_2\}$ are part of one cycle C. See Figure 3. Note that the edges of this cycle come alternatingly from M_1 and M_2 (shown in the figure in red and blue colors).

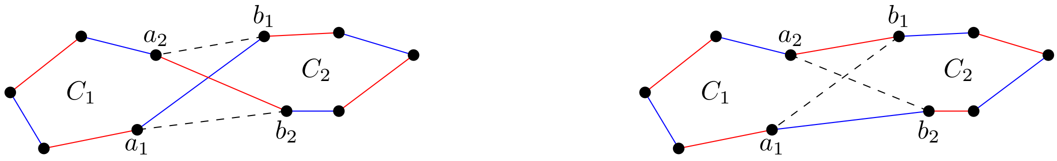


Figure 3 Construction of M_1' and M_2', when the crossing edges are part of one cycle.

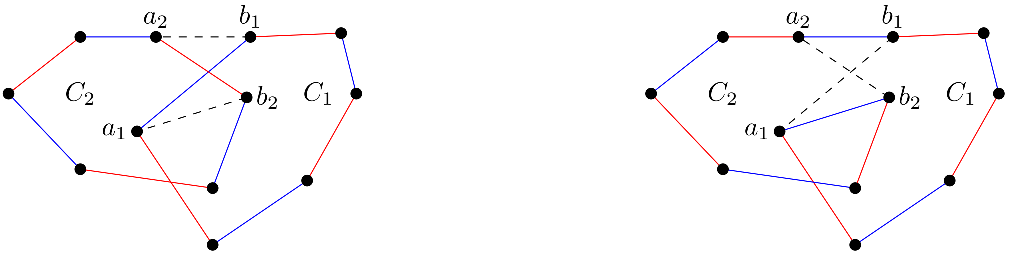


Figure 4 Construction of M_1' and M_2', when the crossing edges are part of two different cycles.

We construct two distinct perfect matchings, M_1' and M_2', using M_1 and M_2. Removing the edges $\{a_1, b_1\}$ and $\{a_2, b_2\}$ from cycle C divides it into two parts. Note that both parts must have an even number of edges since the edges are alternating between M_1 and M_2. It follows that one of these parts is a path from a_1 to a_2, let us call it C_1. And the other one is a path from b_1 to b_2, let us call it C_2 (as shown in Figure 3).

Let us put $\{a_1, b_2\}$ into M_1' and $\{a_2, b_1\}$ into M_2'. For the edges in C_1, we put the M_1 edges into M_1' and the M_2 edges into M_2'. For the edges in C_2 we do the opposite, put the M_1 edges into M_2' and the M_2 edges into M_1'. For edges outside of the cycle C, we put edges from M_1 into M_1' and edges from M_2 into M_2'.

Case (ii): $\{a_1, b_1\}$ and $\{a_2, b_2\}$ are part of two different cycles. Let C_1 and C_2 be the paths obtained from removing $\{a_1, b_1\}$ and $\{a_2, b_2\}$ from the two cycles, respectively. See Figure 4. Here again we construct two distinct perfect matchings, M_1' and M_2', using a similar uncrossing of edges. Let us put both $\{a_1, b_2\}$ and $\{a_2, b_1\}$ into M_1'. For the edges in C_1, we put the M_1 edges into M_1' and the M_2 edges into M_2'. For the edges in C_2 we do the opposite, put the M_1 edges into M_2' and the M_2 edges into M_1'. For edges outside the two cycles, we put edges from M_1 into M_1' and edges from M_2 into M_2'.

Note that in both Case (i) and Case (ii), the newly constructed perfect matchings M_1' and M_2' together have the same edges as $M_1 \cup M_2$, except for $\{a_1, b_1\}$ and $\{a_2, b_2\}$ being replaced with $\{a_2, b_1\}$ and $\{a_1, b_2\}$.

Let w_1, w_2, w_1', w_2' be the weights of matchings M_1, M_2, M_1', M_2', respectively. Then,

$$w_1' + w_2' = w_1 + w_2 - |a_1 b_1| - |a_2 b_2| + |a_1 b_2| + |a_2 b_1|.$$

From Lemma 3.1, we have that

$$|a_1 b_1| + |a_2 b_2| - |a_1 b_2| - |a_2 b_1| \geq 1/(4\sqrt[p]{2}N^{p^4-1}).$$

Thus,

$$w_1' + w_2' \leq w_1 + w_2 - 1/(4\sqrt[p]{2}N^{p^4-1}).$$

Let w^* be the weight of the minimum weight perfect matching. Since M_1 and M_2 are of near-minimum weight, we have $w_1, w_2 < w^* + 1/(8\sqrt[p]{2}N^{p^4-1})$. Using this with the above inequality, we get $w_1' + w_2' < 2w^*$. This implies that at least one of the two matchings M_1' and M_2' have weight smaller than w^*, which is a contradiction. ◀

3.2 Weight scheme

Now, we come to the design of an isolating weight assignment for the graph and the proof of our main theorem. One of the components of our weight scheme is the isolating weight assignment W_{TV} constructed by Tewari and Vinodchandran [27] for planar bipartite graph. We will use the same weight scheme, but for any graph (not necessarily planar) embedded in the plane.

Consider a bipartite graph $G = (A, B, E)$ (not necessarily planar) with a straight-line embedding in $\mathbb{R}^2$. For any vertex u, let (x_u, y_u) be the associated point in $\mathbb{R}^2$. For an edge $e = (u, v)$, where $u \in A$ and $v \in B$, we define the weight function W_{TV} as follows:

$$W_{TV}(e) = (y_v - y_u) \times (x_v + x_u)$$

Then, the theorem below says that W_{TV} is isolating for bipartite planar graphs.

▶ **Theorem 3.4** ([27]). *Let G be a planar bipartite graph. Then with respect to weight function W_{TV} (defined using any planar embedding), the minimum weight perfect matching in G, if one exists, is unique.*

For a geometric bipartite graph, our main idea is to combine W_{TV} with the approximate distance function (up to a certain number of bits of precision) The purpose of combining W_{TV} is to break ties among minimum weight perfect matchings according to the approximate distance function.

Let $G(A, B, E)$ be a geometric bipartite graph on the $N \times N$ integer grid. Let $d(\cdot)$ be the weight function on the edges defined using the ℓ_p distance and let it naturally extend to subsets of edges. For any positive integer ℓ, let us define the approximate distance function $d_\ell \colon E \to \mathbb{Z}$ as

$$d_\ell(e) = \lfloor d(e) \times 2^\ell \rfloor.$$

First, let us show that the minimum weight perfect matchings with respect to approximate distance function remains near-minimum with respect to the exact distance function.

▷ Claim 3.5. For any positive integer ℓ, let M and M^* be minimum weight perfect matchings with respect to functions d_ℓ and d, respectively. Then,

$$d(M) < d(M^*) + n/2^\ell.$$

Proof. Observe that for any edge e, $2^\ell d(e) - 1 < d_\ell(e) \leq 2^\ell d(e)$. Hence, for perfect matching M,

$$2^\ell d(M) - n < d_\ell(M) \leq 2^\ell d(M).$$

Then, we can write

$$2^\ell d(M) < d_\ell(M) + n \leq d_\ell(M^*) + n \leq 2^\ell d(M^*) + n.$$

This implies that $d(M) < d(M^*) + n/2^\ell$. ◁

3.2.1 Weight scheme

For any integer ℓ, now let us define the combined weight function W_ℓ on the edges as follows:

$$W_\ell := (2nN^2 + 1) \times d_\ell + W_{TV}$$

Here, the scaling d_ℓ with a large number ensures that W_ℓ has the same ordering of perfect matchings as d_ℓ, and the W_{TV} function plays the role of tie breaking. Our next lemma says that when to take enough number of bits from the distance function and then combine it with W_{TV} as above, the resulting weight function is isolating.

▶ **Lemma 3.6.** *For any integer $\ell \geq (p^4 - 1)\log N + \log n + 3 + 1/p$, the minimum weight perfect matching in G with respect to the weight function W_ℓ is unique.*

Proof. First, observe that for any two perfect matchings M_1 and M_2,

$$d_\ell(M_1) > d_\ell(M_2) \implies W_\ell(M_1) > W_\ell(M_2).$$

This is because the maximum contribution of W_{TV} to the weight of a matching can be at most $n \times 2N^2$. Thus, we can write

$$\begin{aligned} W_\ell(M_1) - W_\ell(M_2) &= (2nN^2+1)(d_\ell(M_1) - d_\ell(M_2)) + W_{TV}(M_1) - W_{TV}(M_2) \\ &\geq (2nN^2+1)\cdot 1 + 0 - 2nN^2. \\ &\geq 1 \end{aligned}$$

It follows that the set of minimum weight perfect matchings with respect to W_ℓ is a subset of that with respect to d_ℓ. Now, we argue that these sets form a planar subgraph.

▷ Claim 3.7. The union of minimum weight perfect matchings with respect to d_ℓ forms a planar subgraph.

Proof. Let M and M^* be the minimum weight perfect matchings with respect to functions d_ℓ and d, respectively. From Claim 3.5 we have that $d(M) < d(M^*) + n/2^\ell$. By substituting $\ell \geq (p^4-1)\log N + \log n + 3 + 1/p$, we get that the gap is less than $1/(8\sqrt[p]{2}N^{p^4-1})$. Hence, M is a near-minimum weight perfect matching with respect to the $d(\cdot)$. Then, the claim follows from Lemma 3.3. ◁

To finish the proof of the lemma, let H be the subgraph formed by the union of minimum weight perfect matchings with respect to d_ℓ. Clearly, d_ℓ gives equal weights to all the perfect matchings in H. Thus, the function W_ℓ is the same as W_{TV} on H (up to an additive constant). From Theorem 3.4, we know that W_{TV} ensures a unique minimum weight perfect matching in the planar graph H. Hence, so does W_ℓ. ◀

3.2.2 Proof of the main theorem (Theorem 1.2)

Once we have shown how to construct an isolating weight assignment, we just need to use the algorithm of Mulmuley, Vazirani and Vazirani [21] to construct the minimum weight perfect matching.

▶ **Theorem 3.8** ([21]). *Given a graph $G = (V, E)$ with an isolating weight assignment on the edges that uses $O(\log n)$ bits, there is an* NC *algorithm to find the minimum-weight perfect matching.*

Now, we are ready to prove the main theorem. Suppose we are given a bipartite set of $2n$ points in $N \times N$ integer grid. Recall that the weight of an edge is defined to be the Euclidean distance between the endpoints. Our goal is to construct a perfect matching whose weight is at most $w^* + \delta$, where δ is the given error parameter and w^* is the minimum weight of a perfect matching. If we choose $\ell \geq \log(n/\delta)$, then from Claim 3.5, we know that a minimum weight perfect matching with respect to function $d_\ell(\cdot)$ will have the desired property.

We choose $\ell = \max\{\log(n/\delta), (p^4-1)\log N + \log n + 4\}$. Then we use the weight scheme W_ℓ with the MVV algorithm (Theorem 3.8). Recall that from Lemma 3.6, we have the isolation property required in Theorem 3.8. Finally, let us analyse the number of bits used by weight function W_ℓ. The maximum weight given to any edge by function $d(\cdot)$ is at most $\sqrt[p]{2}N$ and by function W_{TV}, it is at most $2N^2$. Thus, the maximum weight given to any edge by function W_ℓ will be at most $2^\ell \times \sqrt[p]{2}N \times (2nN^2+1) + 2N^2$. The number of bits in weight of any edge comes out to be $O(\log(Nn/\delta))$. Hence, we have an NC algorithm, whenever N and $1/\delta$ are polynomial in n.

4 Conclusion

In this work, we explored the parallel complexity of GWBPM problem. We established a lower bound which shows that for GWBPM, a linear number of bits is required to distinguish the minimum-weight perfect matching from others. Next, we showed that GWBPM problem (under ℓ_p metric for $p \geq 2$) that allows up to $\frac{1}{\text{poly}(n)}$ additive error, is in NC. The main question that arises from our work is whether the non-bipartite version of GWBPM is also in NC. Another possible extension is to consider the bipartite version in 3 or higher dimensions.

References

1 The open problems project: problem 33 sum of square roots. https://topp.openproblem.net/p33. Accessed: 2010-09-30.

2 Pankaj K Agarwal, Hsien-Chih Chang, Sharath Raghvendra, and Allen Xiao. Deterministic, near-linear ε-approximation algorithm for geometric bipartite matching. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1052–1065, 2022. doi:10.1145/3519935.3519977.

3 Pankaj K Agarwal, Hsien-Chih Chang, and Allen Xiao. Efficient algorithms for geometric partial matching. In *35th International Symposium on Computational Geometry (SoCG 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

4 Pankaj K Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 39–50, 1995. doi:10.1145/220279.220284.

5 Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in NC. *J. ACM*, 67(4):21:1–21:34, 2020. doi:10.1145/3397504.

6 Christoph Burnikel, Rudolf Fleischer, Kurt Mehlhorn, and Stefan Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27(1):87–99, 2000. doi:10.1007/S004530010005.

7 Richard Carr and Cormac O'Sullivan. On the linear independence of roots. *International Journal of Number Theory*, 05:161–171, 2007.

8 Qi Cheng and Yu-Hsin Li. Finding the smallest gap between sums of square roots. In Alejandro López-Ortiz, editor, *LATIN 2010: Theoretical Informatics*, pages 446–455, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-12200-2_39.

9 Julia Chuzhoy and Sanjeev Khanna. Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, pages 83–94, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3618260.3649725.

10 Elias Dahlhaus and Marek Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1-3):79–91, 1998. doi:10.1016/S0166-218X(98)00006-7.

11 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47(3):737–757, 2010. doi:10.1007/S00224-009-9204-8.

12 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

13 Friedrich Eisenbrand, Matthieu Haeberle, and Neta Singer. An improved bound on sums of square roots via the subspace theorem. *CoRR*, abs/2312.02057, 2023. To appear in SoCG 2024. doi:10.48550/arXiv.2312.02057.

14 Stephen Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-NC. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 754–763, 2016. doi:10.1145/2897518.2897564.

15 Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Using interior-point methods for fast parallel algorithms for bipartite matching and related problems. *SIAM J. Comput.*, 21(1):140–150, February 1992. doi:10.1137/0221011.

16 Dima Yu Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 166–172. IEEE, 1987.

17 John E Hopcroft and Richard M Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973. doi:10.1137/0202019.

18 Richard M Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 22–32, 1985. doi:10.1145/22145.22148.

19 László Lovász. On determinants, matchings, and random algorithms. In *FCT*, volume 79, pages 565–574, 1979.

20 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013. doi:10.1109/FOCS.2013.35.

21 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, 1987. doi:10.1145/28395.383347.

22 Joseph O'Rourke. Advanced problem 6369. *Amer. Math. Monthly*, 88(10):769, 1981.

23 Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

24 R Sharathkumar. A sub-quadratic algorithm for bipartite matching of planar points with bounded integer coordinates. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 9–16, 2013. doi:10.1145/2462356.2480283.

25 R Sharathkumar and Pankaj K Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 306–317. SIAM, 2012. doi:10.1137/1.9781611973099.29.

26 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-NC. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. Ieee, 2017. doi:10.1109/FOCS.2017.70.

27 Raghunath Tewari and NV Vinodchandran. Green's theorem and isolation in planar graphs. *Information and Computation*, 215:1–7, 2012. doi:10.1016/J.IC.2012.03.002.

28 Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023. doi:10.1109/FOCS57990.2023.00037.

29 Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020. doi:10.1109/FOCS46700.2020.00090.

30 Kasturi R Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 320–329. IEEE, 1998.

31 Kasturi R Varadarajan and Pankaj K Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *SODA*, volume 99, pages 805–814, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.314918.

PosSLP and Sum of Squares

Markus Bläser
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Julian Dörfler
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Gorav Jindal
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

The problem PosSLP is the problem of determining whether a given straight-line program (SLP) computes a positive integer. PosSLP was introduced by Allender et al. to study the complexity of numerical analysis (Allender et al., 2009). PosSLP can also be reformulated as the problem of deciding whether the integer computed by a given SLP can be expressed as the sum of squares of four integers, based on the well-known result by Lagrange in 1770, which demonstrated that every natural number can be represented as the sum of four non-negative integer squares.

In this paper, we explore several natural extensions of this problem by investigating whether the positive integer computed by a given SLP can be written as the sum of squares of two or three integers. We delve into the complexity of these variations and demonstrate relations between the complexity of the original PosSLP problem and the complexity of these related problems. Additionally, we introduce a new intriguing problem called Div2SLP and illustrate how Div2SLP is connected to DegSLP and the problem of whether an SLP computes an integer expressible as the sum of three squares.

By comprehending the connections between these problems, our results offer a deeper understanding of decision problems associated with SLPs and open avenues for further exciting research.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Algebraic complexity theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases PosSLP, Straight-line program, Polynomial identity testing, Sum of squares

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.13

Related Version *arXiv Version*: `https://arxiv.org/abs/2403.00115` [6]

Acknowledgements We would like to thank Robert Andrews for providing a simpler proof of Lemma 2.6. We had a proof of it that was a bit longer. Robert Andrews simplified the proof after reviewing our proof in a personal communication.

1 Introduction

1.1 Straight Line Programs and PosSLP

The problem PosSLP was introduced in [1] to study the complexity of numerical analysis and relate the computations over the reals (in the so-called Blum-Shub-Smale model, see [5]) to classical computational complexity. PosSLP asks whether a given integer is positive or not. The problem may seem trivial at first glance but becomes highly non-trivial when the given integer is not explicitly provided but rather represented by an implicit expression which computes it. One way to model the implicit computations of integers and polynomials is through arithmetic circuits and straight line programs (SLPs).

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 13; pp. 13:1–13:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An arithmetic circuit takes the form of a directed acyclic graph where input nodes are designated with constants 0, 1, or variables $x_1, x_2, \ldots, x_m$. Internal nodes are labeled with mathematical operations such as addition $(+)$, subtraction $(-)$, multiplication $(\times)$, or division $(\div)$. Such arithmetic circuits are said to be *constant-free*. In the algebraic complexity theory literature, usually, one studies arithmetic circuits where constants are arbitrary scalars from the underlying field. But in this paper, we are only concerned with arithmetic circuits that are constant-free.

On the other hand, a straight-line program is a series of instructions corresponding to a sequential evaluation of an arithmetic circuit. If this program does not contain any division operations, it is called "division-free". Unless explicitly specified otherwise, we will exclusively consider division-free straight-line programs. Consequently, straight-line programs can be viewed as a compact representation of polynomials or integers. In many instances, we will be concerned with division-free straight-line programs that do not incorporate variables, representing an integer. Arithmetic circuits and SLPs are used interchangeably in this paper. Now we define the central object of study in this paper.

▶ **Problem 1.1** (PosSLP). *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether $N > 0$.*

An SLP P computing an integer is a sequence $(b_0, b_1, b_2, \ldots, b_m)$ of integers such that $b_0 = 1$ and $b_i = b_j \circ_i b_k$ for all $i > 0$, where $j, k < i$ and $\circ_i \in \{+, -, \times\}$. Given such an SLP P, PosSLP is the problem of determining the sign of the integer computed by P, *i.e.*, the sign of b_m. Note that we cannot simply compute b_m from a description of P because the absolute value of b_m can be as large as 2^{2^m}. Therefore computing b_m exactly might require exponential time. Hence this brute-force approach of determining the sign of b_m is too computationally inefficient. [1] also show some evidence that PosSLP might be a hard problem computationally. They achieve this by proving that PosSLP is polynomial-time Turing equivalent to the Boolean component of problems that are solvable in polynomial time in the Blum-Shub-Smale (BSS) model, as well as to the general problem of numerical computation. We briefly survey this relevance of PosSLP to emphasize its importance in numerical analysis. For a more detailed discussion, the interested reader is referred to [1, Section 1].

The Blum-Shub-Smale (BSS) computational model deals with computations using real numbers. It is a well-explored area where complexity theory and numerical analysis meet. For a detailed understanding, see [5]. Here we only describe the constant-free BSS model. BSS machines handle inputs from $\mathbb{R}^\infty := \cup_{k\in\mathbb{N}}\mathbb{R}^k$, allowing polynomial-time computations over $\mathbb{R}$ to solve "decision problems" $L \subseteq \mathbb{R}^\infty$. The set of problems solvable by polynomial-time BSS machines is denoted by $\mathsf{P}^0_\mathbb{R}$, see e.g., [8]. To relate the complexity class $\mathsf{P}^0_\mathbb{R}$ to classical complexity classes, one considers the *boolean part* of $\mathsf{P}^0_\mathbb{R}$, defined as: $\mathrm{BP}(\mathsf{P}^0_\mathbb{R}) := \{L \cap \{0,1\}^\infty \mid L \in \mathsf{P}^0_\mathbb{R}\}$. To highlight the importance of PosSLP as a bridge between the BSS model and Turing machine model, [1] proved the following Theorem 1.2.

▶ **Theorem 1.2** (Proposition 1.1 in [1]). *We have* $\mathsf{P}^{\mathrm{PosSLP}} = \mathrm{BP}(\mathsf{P}^0_\mathbb{R})$.

Another motivation for the complexity of PosSLP comes from its connection to the task of numerical computation. Here we recall this connection from [1]. [1] defined the following problem to formalize the task of numerical computation:

▶ **Problem 1.3** (Generic Task of Numerical Computation (GTNC) [1]). *Given a straight-line program P with n variables, and given inputs $a_1, a_2, \ldots, a_n$ for P (as floating-point numbers) and an integer k in unary, compute a floating-point approximation of $P(a_1, a_2, \ldots, a_n)$ with k significant bits.*

The following result was also demonstrated in [1].

▶ **Theorem 1.4** (Proposition 1.2 in [1]). GTNC *is polynomial-time Turing equivalent to* PosSLP.

1.2 How Hard is PosSLP?

GTNC can be viewed as the task that formalizes what is computationally efficient when we are allowed to compute with arbitrary precision arithmetic. Conversely, the BSS model can be viewed as formalizing computational efficiency, where we have infinite precision arithmetic at no cost. Theorem 1.2 and Theorem 1.4 show that both these models are equivalent to PosSLP under polynomial-time Turing reductions. One can also view these results as an indication that PosSLP is computationally intractable. Despite this, no unconditional non-trivial hardness results for PosSLP beyond P-hardness (which holds due to a reduction from EquSLP) are known. Still, a lot of important computational problems reduce to PosSLP. We briefly survey some of these problems now. By the n-bit binary representation of an integer N with the condition $|N| < 2^n$, we mean a binary string with a length of $n+1$. This string consists of a sign bit followed by n bits encoding the absolute value of N, with leading zeros added if necessary. A very important problem in complexity theory is the EquSLP problem defined as:

▶ **Problem 1.5** (EquSLP, [1]). *Given a straight-line program representing an integer N, decide whether $N = 0$.*

EquSLP is also known to be equivalent to arithmetic circuit identity testing (ACIT) or polynomial identity testing [1]. It is easy to see that EquSLP reduces to PosSLP: $N \in \mathbb{Z}$ is zero if and only if $1 - N^2 > 0$. Recently, a conditional hardness result was proved for PosSLP in [9], formalized below.

▶ **Theorem 1.6** (Theorem 1.2 in [9]). *If a constructive variant of the radical conjecture of [13] is true and* PosSLP $\in$ BPP *then* NP $\subseteq$ BPP*.*

As for upper bounds on PosSLP, PosSLP was shown to be in the counting hierarchy CH in [1]. This is still the best-known upper bound on the complexity of PosSLP. Another important problem is the sum of square roots, defined as follows:

▶ **Problem 1.7** (Sum of Square Roots (SoSRoot)). *Given a list $(a_1, a_2, \ldots, a_n)$ of positive integers and a list $(\delta_1, \delta_2, \ldots, \delta_n) \in \{\pm 1\}^n$ of signs, decide if $\sum_{i=1}^{n} \delta_i \sqrt{a_i}$ is positive.*

SoSRoot is well-known and has applications in computational geometry, as well as in several other fields. The Euclidean traveling salesman problem, whose inclusion in NP is not known, is easily seen to be in NP relative to SoSRoot. SoSRoot is conjectured to be in P in [24] but this is far from clear. Still, one can show that SoSRoot reduces to PosSLP [31, 1]. There are several other problems related to straight line program which are intimately related to PosSLP. For instance, the following problems were also introduced in [1]. These problems will be useful in our discussion later.

▶ **Problem 1.8** (BitSLP). *Given a straight-line program representing N, and given $n, i \in \mathbb{N}$ in binary, decide whether the i^{th} bit of the n-bit binary representation of N is 1.*

It was also shown in [1] that PosSLP reduces to BitSLP. Although we do not know any unconditional hardness results for PosSLP, BitSLP was shown to be #P-hard in [1]. Another important problem related to PosSLP is the following DegSLP problem, which was shown to be reducible to PosSLP in [1].

▶ **Problem 1.9** (DegSLP). *Given a straight-line program representing a polynomial $f \in \mathbb{Z}[x]$ and a natural number d in binary, decide whether* $\deg(f) \leq d$*.*

The problem DegSLP was posed in [1] for multivariate polynomials, here we have considered its univariate version. But these are seen to be equivalent under polynomial time many-one reductions [1, Proof of Proposition 2.3], we recall this reduction in Section C. We also recall the following new problem from [12] related to straight line programs, which is important to results in this paper.

▶ **Problem 1.10** (OrdSLP). *Given a straight-line program representing a polynomial $f \in \mathbb{Z}[x]$ and a natural number ℓ in binary, decide whether* $\operatorname{ord}(f) \geq \ell$*. Here, the order of f, denoted as* $\operatorname{ord}(f)$*, is defined to be the largest k such that $x^k \mid f$.*

1.3 Our Results

Lagrange proved in 1770 that every natural number can be represented as a sum of four non-negative integer squares [27, Theorem 6.26]. Therefore PosSLP can be reformulated as: Given a straight-line program representing $N \in \mathbb{Z}$, decide if there exist $a, b, c, d \in \mathbb{N}$ (not all zero) such that $N = a^2 + b^2 + c^2 + d^2$. In light of this rephrasing of PosSLP, we study the various sum of squares variants of PosSLP in Section 2 and Section 3. To formally state our results, we define these problems now. For convenience, we say that $n \in \mathbb{N}$ is 3SoS if it can be expressed as the sum of three squares (of integers). We study the following problem.

▶ **Problem 1.11** (3SoSSLP). *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether N is a* 3SoS*.*

One might expect that 3SoSSLP is easier than PosSLP, but we show that PosSLP reduces to 3SoSSLP under polynomial-time Turing reductions. More precisely, we prove the following Theorem 1.12 in Section 2.

▶ **Theorem 1.12.** $\text{PosSLP} \in \mathsf{P}^{\text{3SoSSLP}}$.

Similarly, we say that $n \in \mathbb{N}$ is 2SoS if it can be expressed as the sum of two squares (of integers). We also study the following problem.

▶ **Problem 1.13** (2SoSSLP). *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether N is a* 2SoS*.*

These problems 3SoSSLP and 2SoSSLP can also be seen as special cases of the renowned Waring problem. The Waring problem has an intriguing history in number theory. It asks whether for each $k \in \mathbb{N}$ there exists a positive integer $g(k)$ such that any natural number can be written as the sum of at most $g(k)$ many k^{th} powers of natural numbers. Lagrange's four-square theorem can be seen as the equality $g(2) = 4$. Later, Hilbert settled the Waring problem for integers by proving that $g(k)$ is finite for every k [16]. Therefore, problems 2SoSSLP and 3SoSSLP can be seen as computational variants of the Waring problem. These computational variants of the Waring problems are extensively studied in computer algebra and algebraic complexity theory, and Shitov actually proved that computing the Waring rank of multivariate polynomials is $\exists\mathbb{R}$-hard [30]. For 2SoSSLP, we prove the following conditional hardness result in Section 3.

▶ **Theorem 1.14.** *If the generalized Cramér conjecture A (Conjecture 3.3) is true, then* $\text{PosSLP} \in \mathsf{NP}^{\text{2SoSSLP}}$.

We also study whether 3SoSSLP can be reduced to PosSLP. Unfortunately, we cannot show this reduction unconditionally. Hence we study and rely on the following problem Div2SLP, which might be of independent interest. One can view Div2SLP as the variant of OrdSLP for numbers in binary.

▶ **Problem 1.15** (Div2SLP). *Given a straight-line program representing $N \in \mathbb{Z}$, and a natural number ℓ in binary, decide if 2^ℓ divides $|N|$, i.e., the ℓ least significant bits of $|N|$ are zero.*

We show that if we are allowed oracle access to both PosSLP and Div2SLP oracles then 3SoSSLP can be decided in polynomial time, formalized below in Theorem 1.16. A proof can be found in Section 2.

▶ **Theorem 1.16.** 3SoSSLP $\in \mathsf{P}^{\{\text{Div2SLP},\text{PosSLP}\}}$.

We also study how Div2SLP is related to other problems related to straight line programs. To this end, we prove the following Theorem 1.17 in Section 2.

▶ **Theorem 1.17.** OrdSLP $\equiv_{\mathsf{P}}$ DegSLP $\leq_{\mathsf{P}}$ Div2SLP.

As for the hardness results for 3SoSSLP and 2SoSSLP, we also show that similar to PosSLP, EquSLP reduces to both 3SoSSLP and 2SoSSLP. Analogous to integers, we also study the complexity of deciding the positivity of univariate polynomials computed by a given SLP. In this context, we study the following problem.

▶ **Problem 1.18** (PosPolySLP). *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if f is positive, i.e., $f(x) \geq 0$ for all $x \in \mathbb{R}$.*

We prove that in contrast to PosSLP, hardness of PosPolySLP can be proved unconditionally, formalized below in Theorem 1.19.

▶ **Theorem 1.19.** PosPolySLP *is* coNP*-hard under polynomial-time many-one reductions.*

In contrast to numbers, every positive polynomial can be written as the sum of two squares (but only over the reals, see Section 4 for a detailed discussion). So PosPolySLP is equivalent to the question whether f is the sum of two squares. To conclude, we motivate and study the following problem (see Section 4 for more details).

▶ **Problem 1.20** (SqPolySLP). *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if $\exists g \in \mathbb{Z}[x]$ such that $f = g^2$.*

We show in Section 4 that SqPolySLP is in coRP.

2 SLPs as Sums of Three Squares

This section is concerned with studying the complexity of 3SoSSLP and related problems.

2.1 Lower Bound for 3SoSSLP

In this section, we prove Theorem 1.12. We use the following characterization of integers which can be expressed as the sum of three squares.

▶ **Theorem 2.1** ([23, 15, 3, 25]). *An integer n is* 3SoS *if and only if it is not of the form $4^a(8k+7)$, with $a, k \in \mathbb{N}$.*

Theorem 2.1 informally implies that 3SoS integers are "dense" in $\mathbb{N}$ and hence occur very frequently. A useful application of this intuitive high density of 3SoS integers is demonstrated below in Lemma 2.2. More formally, Landau showed that the asymptotic density of 3SoS integers in $\mathbb{N}$ is 5/6 [21]. To reduce PosSLP to 3SoSSLP, we shift the given integer (represented by a given SLP) by a positive number to convert into 3SoS. To this end we prove the following Lemma 2.2.

▶ **Lemma 2.2.** *For every $n \in \mathbb{N}$, at least one element in the set $\{n, n+2\}$ is* 3SoS*.*

Proof. If n is 3SoS then we are done. Suppose n is not 3SoS, by using Theorem 2.1 we know that $n = 4^a(8k+7)$ for some $a, k \in \mathbb{N}$. If $a = 0$ then $n = 8k+7$ and hence $n+2 = 8k+9$ is clearly not of the form $4^b(8c+7)$ for any $b, c \in \mathbb{N}$. If $a > 0$ then $n+2 = 4^a(8k+7)+2$ is not divisible by 4. Hence for $n+2$ of the form $4^b(8c+7)$, we have to have $4^a(8k+7)+2 = 8c+7$. This is clearly impossible because the LHS is even whereas RHS is odd. ◀

▶ **Lemma 2.3.** *If $M \in \mathbb{Z}_+$ then $7M^4$ not a* 3SoS*.*

Proof. Suppose $M = 4^a(4b+c)$ where a is the largest power of 4 dividing M, $c = \frac{M}{4^a} \pmod 4$ and $b = \lfloor \frac{M}{4^{a+1}} \rfloor$. We prove the claim by analyzing the following cases.

If $c = 0$ then $M = 4^a b$ for some $a, b \in \mathbb{N}$ and 4 does not divide b. Note that here $a > 0$, otherwise c cannot be zero by its definition. Therefore $7M^4 = 4^{4a} \cdot 7b^4$. Now we can apply this Lemma recursively on b (which is smaller than M) to infer that $7b^4$ is of the form $4^\alpha(8\beta+7)$ for some $\alpha, \beta \in \mathbb{N}$. Hence $7M^4$ is also of this form and thus not a 3SoS by using Theorem 2.1.

If $c = 1$ then $7M^4 = 4^{4a} \cdot 7 \cdot (256b^4 + 256b^3 + 96b^2 + 16b + 1) = 4^\alpha(8\beta+7)$ for some $\alpha, \beta \in \mathbb{N}$, hence $7M^4$ is not a 3SoS by using Theorem 2.1.

If $c = 2$ then $7M^4 = 4^{4a+2} \cdot 7 \cdot (16b^4 + 32b^3 + 24b^2 + 8b + 1) = 4^\alpha(8\beta+7)$ for some $\alpha, \beta \in \mathbb{N}$, hence $7M^4$ is not a 3SoS by using Theorem 2.1.

If $c = 3$ then $7M^4 = 4^{4a} \cdot 7 \cdot (256b^4 + 768b^3 + 964b^2 + 12496b + 81) = 4^\alpha(8\beta+7)$ for some $\alpha, \beta \in \mathbb{N}$, hence $7M^4$ is not a 3SoS by using Theorem 2.1. ◀

Lemma 2.3 implies the following EquSLP hardness of 3SoSSLP.

▶ **Lemma 2.4.** EquSLP $\leq_{\mathsf{P}}$ 3SoSSLP*.*

Proof. Given a straight-line program representing an integer N, we want to decide whether $N = 0$. Suppose $M = N^2$. We have $M \geq 0$ and $M = 0$ iff $N = 0$. By using Lemma 2.3, we know that $7M^4$ is a 3SoS iff $M = 0$. ◀

▶ Remark 2.5. Lemma 2.4 illustrates that 3SoSSLP is at least as hard as EquSLP under deterministic polynomial time Turing reductions. This may not appear as a very strong result, since EquSLP can be decided in randomized polynomial time anyway. However, unconditionally, even PosSLP is known to be only EquSLP-hard. Moreover, we rely on Lemma 2.4 in the proof of Theorem 1.12 below.

▶ **Theorem 1.12.** PosSLP $\in \mathsf{P}^{\text{3SoSSLP}}$.

Proof of Theorem 1.12. Given a straight-line program representing an integer N, we want to decide whether $N > 0$. Using an EquSLP oracle, we first check if $N \in \{0, -1, -2\}$. By using Lemma 2.4, these oracle calls to EquSLP can also be simulated by oracle calls to the 3SoSSLP oracle. Hence this task belongs to $\mathsf{P}^{\text{3SoSSLP}}$. If $N \in \{0, -1, -2\}$, then clearly $N > 0$ is false, and we answer "No". Otherwise we check if N is a 3SoS, if it is then clearly $N > 0$ and we answer "Yes". If it is not a 3SoS then we check if $N+2$ is a 3SoS. If $N+2$ is a 3SoS then clearly $N > 0$ because $N \notin \{0, -1, -2\}$. If $N+2$ is not a 3SoS, then by Lemma 2.2 we can conclude that $N < -2$ and hence we answer "No". ◀

2.2 Upper Bound for 3SoSSLP

Now we prove the upper bound for 3SoSSLP, claimed in Theorem 1.16.

▶ **Theorem 1.16.** 3SoSSLP $\in \mathsf{P}^{\{\text{Div2SLP},\text{PosSLP}\}}$.

Proof of Theorem 1.16. Given an $N \in \mathbb{Z}$ represented by a given SLP, we want to decide if N is a 3SoS. By using the PosSLP oracle, we first check if $N \geq 0$. If $N < 0$ then we answer "No". By invoking the PosSLP oracle again on $1 - N^2$, we can determine whether $N = 0$, in which case we answer "Yes". Hence we can now assume that $N > 0$. By using Theorem 2.1, it is easy to see that N is not a 3SoS iff the binary representation $\text{Bin}(N)$ of N looks like below:

$$N \text{ is not a } 3\text{SoS} \iff \text{Bin}(N) = S1110^t \text{ where } t \text{ is even and } S \in \{0,1\}^*.$$

By using the Div2SLP oracle, we compute the number of trailing zeroes (call it again t) in the binary representation of N. This can be achieved by doing a binary search and repeatedly using the Div2SLP oracle. If t is not even, then N is a 3SoS. Next we construct an SLP which computes 2^t, *i.e.*, the number 10^t in the binary representation. Such an SLP can be constructed in time $\mathsf{poly}(\log t)$ and is of size $O(\log^2 t)$. This can be seen by looking at the binary representation of t and then using repeated squaring. We have:

$$\text{Bin}\left(N + 2^t\right) = S'0^{t+3} \iff \text{Bin}(N) = S1110^t \text{ for some } S, S' \in \{0,1\}^*.$$

Hence N is not a 3SoS iff $N + 2^t$ has $t + 3$ trailing zeroes, which again can be decided using the Div2SLP oracle. ◀

2.3 Complexity of Div2SLP

In this section, we show a DegSLP lower bound for Div2SLP. To this end, we first prove the following equivalence of DegSLP and OrdSLP.

▶ **Lemma 2.6.** *Given a straight-line program P of length s computing a polynomial $f \in \mathbb{Z}[x]$, we can compute in $\mathsf{poly}(s)$ time:*

1. *A number $m \in \mathbb{N}$ such that $\deg(f) \leq m \leq 2^s$.*
2. *A straight line program Q of length $O(s)$ such that Q computes the polynomial $x^m f\left(\frac{1}{x}\right)$.*

Proof. We generate the desired straight line program Q in an inductive manner. Namely, if a gate g in P computes a polynomial R_g then the corresponding gate in Q computes a number $m_g \geq \deg(R_g)$ (the gate itself does not compute a number, to be precise, but our reduction algorithm does) and the polynomial $x^{m_g} R_g\left(\frac{1}{x}\right) \in \mathbb{Z}[x]$. It is clear how to do it for leaf nodes. Suppose $g = g_1 + g_2$ is a $+$ gate in P. So we have already computed integers m_{g_1}, m_{g_2} and polynomials $x^{m_{g_1}} R_{g_1}\left(\frac{1}{x}\right), x^{m_{g_2}} R_{g_2}\left(\frac{1}{x}\right)$. We consider $m_g := m_{g_1} + m_{g_2}$. We then have:

$$x^{m_g} R_g\left(\frac{1}{x}\right) = x^{m_{g_2}} x^{m_{g_1}} R_{g_1}\left(\frac{1}{x}\right) + x^{m_{g_1}} x^{m_{g_2}} R_{g_2}\left(\frac{1}{x}\right).$$

We also construct a straight-line program of length $O(s)$ that simultaneously computes x^{m_h} for all gates h in P. With this, we can compute $x^{m_g} R_g\left(\frac{1}{x}\right)$ using 3 additional gates. This implies the straight-line program for Q can be implemented using only $O(s)$ gates. Similarly, for a × gate $g = g_1 \times g_2$, we can simply use $x^{m_g} R_g\left(\frac{1}{x}\right) = x^{m_{g_1}+m_{g_2}} R_{g_1}\left(\frac{1}{x}\right) R_{g_2}\left(\frac{1}{x}\right)$ with $m_g = m_{g_1} + m_{g_2}$. By induction, it is also clear that at the top gate g, we have $m_g \leq 2^s$.

It remains to describe a straight-line program of length $O(s)$ which computes x^{m_h} for all gates h in P. Consider the straight-line program P' obtained from P by changing every addition gate into a multiplication gate. If g' is a gate in P' corresponding to the gate g in P, then one can show via induction that $R_{g'}(x) = x^{m_g}$. This gives the desired straight-line program. ◀

▶ **Lemma 2.7.** DegSLP $\leq_{\mathsf{P}}$ OrdSLP.

Proof. Suppose we are given a straight line program P of length s computing a polynomial $f \in \mathbb{Z}[x]$. By using Lemma 2.6, we compute:

1. A number $m \in \mathbb{N}$ such that $\deg(f) \leq m \leq 2^s$.
2. A straight line program Q of length $O(s)$ such that Q computes the polynomial $x^m f\left(\frac{1}{x}\right) \in \mathbb{Z}[x]$.

Now it is clear that:

$$\deg(f) \leq d \Longleftrightarrow \operatorname{ord}\left(x^m f\left(\frac{1}{x}\right)\right) \geq (m-d).$$

Hence the claim follows. ◀

The proof of the following Lemma 2.8 is almost the same to that of Lemma 2.7, hence we omit it.

▶ **Lemma 2.8.** OrdSLP $\leq_{\mathsf{P}}$ DegSLP.

▶ **Theorem 2.9.** OrdSLP $\equiv_{\mathsf{P}}$ DegSLP.

Proof. Follows immediately from Lemma 2.7 and Lemma 2.8. ◀

▶ **Theorem 2.10.** OrdSLP $\equiv_{\mathsf{P}}$ DegSLP $\leq_{\mathsf{P}}$ Div2SLP.

Proof. We only need to show that OrdSLP $\leq_{\mathsf{P}}$ Div2SLP. Suppose we are given a straight line program P of length s computing a polynomial $f \in \mathbb{Z}[x]$ and $\ell \in \mathbb{N}$ in binary, we want to decide if $\operatorname{ord}(f) \geq \ell$. We know that $\|f\|_\infty \leq 2^{2^s}$, where $\|f\|_\infty$ is the maximum absolute value of coefficients of $f(x)$. We now construct an SLP which computes $f(B)$ where B is a suitably chosen large integer, which we will specify in a moment. If $\operatorname{ord}(f) \geq \ell$ then clearly B^ℓ divides $f(B)$. Now consider the case when $\operatorname{ord}(f) = m < \ell$. So we have $f = x^m(f_0 + xg)$ for some $f_0 \in \mathbb{Z}, g \in \mathbb{Z}[x]$ and $m < \ell$. Here $f_0 \neq 0$. In this case we have:

$$f(B) = B^m(f_0 + Bg(B)).$$

If B is chosen large enough then B does not divide $f_0 + Bg(B)$ and hence B^ℓ does not divide $f(B)$. It can be verified that choosing $B = 2^{2^{3s}}$ suffices for this argument. It is also not hard to see that a SLP for $f(B)$ can be constructed in polynomial time. Hence we conclude:

$$\operatorname{ord}(f) \geq \ell \iff 2^{\ell 2^{3s}} \text{ divides } f(2^{2^{3s}}).$$

This completes the reduction. ◀

▶ **Problem 2.11.** *What is the exact complexity of* Div2SLP*?*

Now we show that Div2SLP is in CH, this claim follows by employing ideas from [1].

▶ **Lemma 2.12.** Div2SLP *is in* CH.

Proof. Given a straight-line program representing $N \in \mathbb{Z}$, and a natural number ℓ in binary, we want to decide if 2^ℓ divides $|N|$, *i.e.*, if the ℓ least significant bits of $|N|$ are zero. We show that this can be done in $\mathsf{coNP}^{\text{BitSLP}}$. The condition $2^\ell \nmid |N|$ is equivalent to the statement that at least one bit in ℓ least significant bits of $|N|$ is one. Hence there is a witness of this statement, *i.e.*, the index $i \leq \ell$ such that i^{th} bit of $|N|$ is one. By using the BitSLP oracle, we can verify the existence of such a witness in polynomial time. Therefore Div2SLP $\in \mathsf{coNP}^{\text{BitSLP}}$. By using [1, Theorem 4.1], we get that Div2SLP $\in \mathsf{coNP}^{\mathsf{CH}} \subseteq \mathsf{CH}$. ◀

In Section B, we provide a more general proof showing that "SLP versions" of problems in dlogtime uniform TC_0 are in CH, although it is not required for the main results.

3 SLPs as Sum of Two and Fewer Squares

This section is primarily concerned with studying the complexity of 2SoSSLP. To this end, we first recall the following renowned Theorem 3.1 which characterizes when a natural number is a sum of two squares.

▶ **Theorem 3.1** ([11, Section 18]). *An integer $n > 1$ is not* 2SoS *if and only if the prime-power decomposition of n contains a prime of the form $4k + 3$ with an odd power.*

When the input integer n is given explicitly as a bit string, Theorem 3.1 illustrates that a factorization oracle suffices to determine whether n is a 2SoS. In fact, we are not aware of any algorithm that bypasses the need for factorization. For $x \in \mathbb{Z}_+$, let $B(x)$ denote the number of 2SoS integers in $[x]$. Landau's Theorem [22] gives the following asymptotic formula for $B(x)$.

▶ **Theorem 3.2** ([22]). $B(x) = K\frac{x}{\sqrt{\ln x}} + O\left(\frac{x}{\ln^{3/2} x}\right)$ *as $x \to \infty$, where K is the Landau-Ramanujan constant with $K \approx 0.764$.*

Ideally, we want to use the above Theorem 3.2 on the density of 2SoS to show that PosSLP reduces to 2SoSSLP, as we did for 3SoSSLP. There are two issues with this approach:

1. The density of 2SoS integers is not as high as 3SoS integers, hence to find the next 2SoS integer after a given $N \in \mathbb{N}$ might require a larger shift (as compared to the shift of 2 for 3SoS). This issue is overcome below by using NP oracle reductions instead of P reductions.
2. A more serious issue is that Theorem 3.2 says something about the density of 2SoS integers only asymptotically, as $x \to \infty$. But this idea of finding the next 2SoS integer after a given integer only works if this density bound is true for all intervals of naturals. This issue is side stepped by relying on the Conjecture 3.3 below.

Let q and r be positive integers such that $1 \leq r < q$ and $\gcd(q, r) = 1$. We use $G_{q,r}(x)$ to denote the maximum gap between primes in the arithmetic progression $\{qn + r \mid n \in \mathbb{N}, qn + r \leq x\}$. We use $\varphi(n)$ to denote the Euler's totient function, *i.e.*, the number of positive $m \leq n$ with $\gcd(m, n) = 1$.

▶ **Conjecture 3.3** (Generalized Cramér conjecture A, [20]). *For any $q > r \geq 1$ with $\gcd(q, r) = 1$, we have*

$$G_{q,r}(p) = O(\varphi(q) \log^2 p).$$

3.1 Lower Bounds for 2SoSSLP

▶ **Lemma 3.4.** EquSLP $\leq_{\mathsf{P}}$ 2SoSSLP.

Proof. Given a straight-line program representing an integer N, we want to decide whether $N = 0$. Suppose $M = N^2$. We have $M \geq 0$ and $M = 0$ iff $N = 0$. If $M \neq 0$ then by employing Theorem 3.1, $3M^2$ cannot be a 2SoS. Hence $3M^2$ is a 2SoS iff $M = 0$. ◀

▶ **Lemma 3.5** (Zweiter Teil in [7]). *For $x \geq 7$, there exists at least one prime number in the interval $(x, 2x]$ that belongs to the arithmetic progression $4n + 1$.*

▶ **Theorem 1.14.** *If the generalized Cramér conjecture A (Conjecture 3.3) is true, then* PosSLP $\in \mathsf{NP}^{\text{2SoSSLP}}$.

Proof of Theorem 1.14. Given a straight-line program (SLP) of size s representing an integer N, we aim to decide whether $N > 0$.

To proceed, choose $M := 2^{3s}$. Our first step is to compute $N \bmod T$, where $T := 2M + 1$. Specifically, we compute an integer K such that $K \in [-M, M]$ and $K = N \bmod T$.

This computation can be done in $\mathsf{poly}(s)$-time by simulating the SLP that computes N, modulo T. If $|N| \leq M$, then we know that $N = K$. Using the EquSLP oracle (which can be simulated by the 2SoSSLP oracle via Lemma 3.4), we check whether $N = K$ holds. If $N = K$, we can immediately determine the sign of N. Otherwise, our assumption $|N| \leq M$ is false, meaning we can conclude that $|N| > M$.

Now, suppose $p \geq |N|$ is the smallest prime of the form $4k + 1$. By Lemma 3.5, we know that $p \leq 2|N|$ for $|N| \geq 7$. Moreover, by using Conjecture 3.3 with $q = 4, r = 1$, we obtain $p \leq |N| + O(\varphi(4)\log^2 p) \leq |N| + c\log^2|N|$ for some absolute constant c. For sufficiently large $|N|$, this implies $p \leq |N| + \log^3|N| \leq |N| + 2^{3s}$. Consequently, $p - |N| \leq M$. Since p is a prime of the form $4k + 1$, we know, by Theorem 3.1, that p is a sum of two squares (2SoS).

To establish that PosSLP $\in \mathsf{NP}^{\text{2SoSSLP}}$, we must provide a witness for the positivity of N, which can be verified in polynomial time using the 2SoSSLP oracle. The desired witness is $S := p - |N| \leq M$, which has a binary description of size at most $O(s)$. We then use the 2SoSSLP oracle to check whether $N + S$ is a sum of two squares.

If $N > 0$, such a witness exists. On the other hand, if $N < 0$, we know that $N < -M$, which implies that $N + S < 0$, and thus $N + S$ cannot be a sum of two squares. Therefore if Conjecture 3.3 holds, we conclude that PosSLP $\in \mathsf{NP}^{\text{2SoSSLP}}$. ◀

Similarly to 2SoSSLP and 3SoSSLP, one can also study the complexity of the following problem SquSLP.

▶ **Problem 3.6** (SquSLP, Problem 7 in [17]). *Given a straight-line program representing $N \in \mathbb{Z}$, decide whether $N = a^2$ for some $a \in \mathbb{Z}$.*

SquSLP was shown to be decidable in randomized polynomial time in [17, Sec 4.2], assuming GRH. The complexity of 2SoSSLP remains an intriguing open problem. If 2SoSSLP were to be in P then this would disprove Conjecture 3.3 or prove that PosSLP $\in \mathsf{NP}$, neither of which is currently known.

4 Polynomials as Sum of Squares

4.1 Positivity of Polynomials

Analogous to PosSLP, we also study the positivity problem for polynomials represented by straight line programs. In particular, we study the following problem, called PosPolySLP.

▶ **Problem 4.1** (PosPolySLP). *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if f is positive, i.e., $f(x) \geq 0$ for all $x \in \mathbb{R}$.*

It is known that every positive univariate polynomial f can be written as sum of two squares. The formal statement (Lemma A.1) and its folklore proof can be found in the appendix.

Now we look at the rational variant of the Lemma A.1. Suppose $f \in \mathbb{Z}[x] \subset \mathbb{Q}[x]$ is a positive polynomial. We know that it can be written as a sum of squares of two real polynomials. Can it also be written as sum of squares of rational polynomials? In this direction, Landau proved that each positive polynomial in $\mathbb{Q}[x]$ can be expressed as a sum of at most eight polynomial squares in $\mathbb{Q}[x]$ [14]. Pourchet improved this result and proved that only five or fewer squares are needed [33].

We now show that PosPolySLP is coNP-hard, this result follows from an application of results proved in [28]. Suppose W is a 3-SAT formula on n literals $x_1, x_2, \ldots, x_n$ with $W = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$, here C_i is a clause composed of 3 literals. We choose any n distinct odd primes $p_1 < p_2 < \cdots < p_n$. So x_i is associated with the prime p_i. Thereafter, we define $M := \prod_{i\in[n]} p_i$. The following Theorem 4.2 was proved in [28].

▶ **Theorem 4.2** ([28]). *One can construct a SLP C of size $\mathsf{poly}(p_n, \ell)$ which computes a polynomial $P_M(W)$ of the form:*

$$P_M(W) := \sum_{i\in[\ell]} (F_M(C_i))^2$$

such that $P_M(W)$ has a real root iff W is satisfiable. Here, $F_M(C_i)$ is a univariate polynomial that depends on C_i (see [28] for more details).

▶ **Theorem 4.3** (Theorem 1.2 in [4]). *Let $f \in \mathbb{Z}[x]$ be a univariate polynomial of degree d taking only positive values on the interval $[0, 1]$. Let τ be an upper bound on the bit size of the coefficients of f. Let m denote the minimum of f over $[0, 1]$. Then*

$$m > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

The theorem above proves the lower bound for the interval $[0, 1]$. Next, we extend it to the whole real line.

▶ **Lemma 4.4.** *Let $f \in \mathbb{Z}[x]$ be a positive univariate polynomial of degree d. Let τ be an upper bound on the bit size of the coefficients of f. Let m denote the minimum of f over $\mathbb{R}$. If $m \neq 0$ then*

$$m > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

Proof. We assume $m \neq 0$. Consider the reverse polynomial $f_{\text{rev}} := x^d f\left(\frac{1}{x}\right)$. It is clear that f_{rev} is positive on $[0, \infty)$. Moreover, f_{rev} has degree d and τ is an upper bound on the bit size of its coefficients. By employing Theorem 4.3 on f_{rev}, we infer that

$$\min_{a\in[0,1]} f_{\text{rev}}(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

Theorem 4.3 implies that:

$$\min_{a\in[0,1]} f(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}. \tag{1}$$

Now consider a $\lambda \in [0,1]$, we have:

$$f\left(\frac{1}{\lambda}\right) = \frac{f_{\text{rev}}(\lambda)}{\lambda^d} \geq f_{\text{rev}}(\lambda) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}. \tag{2}$$

By combining Equation (1) and Equation (2), we obtain that:

$$\min_{a\in[0,\infty)} f(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}.$$

By repeating the above argument on $f(-x)$ instead of $f(x)$, we obtain:

$$m = \min_{a\in(-\infty,\infty)} f(a) > \frac{3^{d/2}}{2^{(2d-1)\tau}(d+1)^{2d-1/2}}. \qquad \blacktriangleleft$$

▶ **Theorem 1.19.** PosPolySLP *is* coNP*-hard under polynomial-time many-one reductions.*

Proof of Theorem 1.19. Suppose W is 3-SAT formula on n literals $x_1, x_2, \ldots, x_n$ with $W = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$, here C_i is a clause composed of 3 literals. By using Theorem 4.2, we can construct a SLP of size $\mathsf{poly}(p_n, \ell)$ which computes a polynomial $P(W) \in \mathbb{Z}[x]$ such that $P(W)$ has a real root iff W is satisfiable. (Recall that $p_1 < \cdots < p_n$ was a sequence of odd primes.) Since $P(W)$ is a sum of squares, $P(W)$ is positive. Suppose m is the minimum value of $P(W)$ over $\mathbb{R}$. We know that $m \geq 0$.

By the prime number theorem, we can assume $p_n = O(n \log n)$. Moreover, it is easy to see that $\ell \leq 8n^3$. Hence the constructed SLP is of size $s = \mathsf{poly}(n)$. Suppose τ is an upper bound on the bit size of the coefficients of $P(W)$. It is easy to see that $\deg(P(W)) \leq 2^s$ and $\tau \leq 2^s$. If W is not satisfiable then we know that $m \neq 0$ and therefore Lemma 4.4 implies that

$$\log(m) > 2^{s-1}\log 3 - (2^{s+1}-1)2^s - (2^{s+1}-1/2)\log(2^s+1) > -2^{2s+2}.$$

Hence

$$m > \frac{1}{2^{2^{2s+2}}}.$$

Suppose $B = 2^{2^{2s+2}}$. Then $B \cdot P(W) - 1$ is positive iff $m > 0$. Hence we have:

$B \cdot P(W) - 1$ is positive iff W is unsatisfiable.

Moreover $B \cdot P(W) - 1$ has a SLP of size $O(s) = \mathsf{poly}(n)$ and this SLP can be constructed in time $\mathsf{poly}(n)$. Since determining the unsatisfiability of W is coNP-complete, it follows that PosPolySLP is coNP-hard. ◀

4.2 Checking if a Polynomial is a Square

In light of the results in [33] and Theorem 1.19, we also study the following related problem SqPolySLP. Another motivation to study this problem also comes from the quest for studying the complexity of factors of polynomials. In this context, one wants to prove that if a polynomial can be computed by a small arithmetic circuit, then so can be its factors. In this direction, Kaltofen showed that if a polynomial $f = g^e h$ can be computed an arithmetic circuit of size s and g, h are coprime, then g can also be computed by a circuit of size $\mathsf{poly}(e, \deg(g), s)$ [18]. When $f = g^e$, Kaltofen also showed that g can be computed by an arithmetic circuit of size $\mathsf{poly}(\deg(g), s)$ [18]. This question for finite fields is posed as an open question in [19]. What if we do not want to find a small circuit for polynomial g in case $f = g^e$ but only want to determine if f is e^{th} power of some polynomial. And in this decision problem, we want to avoid the dependency on $\deg(g)$ in running time, which can be exponential in s. We study this problem for $e = 2$ in SqPolySLP, but our results work for any arbitrary constant e.

▶ **Problem 4.5** (SqPolySLP). *Given a straight-line program representing a univariate polynomial $f \in \mathbb{Z}[x]$, decide if $\exists g \in \mathbb{Z}[x]$ such that $f = g^2$.*

One can also study the complexity of determining if the given univariate polynomial can be written as a sum of two, three or four squares, but in this section, we only focus on the problem SqPolySLP. The following Theorem 4.6 hints to an approach that SqPolySLP can be reduced to SquSLP.

▶ **Theorem 4.6** (Theorem 4 in [26]). *For $f \in \mathbb{Z}[x]$, $\exists g \in \mathbb{Z}[x]$ with $f = g^2$ iff $\forall t \in \mathbb{Z}$, $f(t)$ is a perfect square.*

We shall use an effective variant of Theorem 4.6 which follows from the following effective variant of the Hilbert's irreducibility theorem. For an integer polynomial f, $H(f)$ is the height of f, *i.e.*, the maximum of the absolute values of the coefficients of f.

▶ **Theorem 4.7** ([32, 10]). *Suppose $P(T, Y)$ is an irreducible polynomial in $\mathbb{Q}[T, Y]$ with $\deg_Y(P) \geq 2$ and with coefficients in $\mathbb{Z}$ assumed to be relatively prime. Suppose B is a positive integer such that $B \geq 2$. We define:*

$$\begin{aligned} m &:= \deg_T(P) \\ n &:= \deg_Y(P) \\ H &:= \max(H(P), e^e) \\ S(P, B) &:= |\{1 \leq t \leq B \mid P(t, Y) \textit{ is reducible in } \mathbb{Q}[Y]\}| \end{aligned}$$

Then we have:

$$S(P, B) \leq 2^{165} m^{64} 2^{296n} \log^{19}(H) B^{\frac{1}{2}} \log^5(B).$$

▶ **Corollary 4.8.** *Suppose $f(x) \in \mathbb{Z}[x]$ is an integer polynomial computed by a SLP of size s. Define $S(f) := \left|\{1 \leq t \leq 2^{200s} \mid f(t) \textit{ is a square }\}\right|$. If f is not a square, then we have:*

$$S(f) < 2^{800} s^5 2^{183s}.$$

Proof. Consider the polynomial $P(T, Y) := Y^2 - f(T)$. Since $f(x)$ is not a square, we infer that $P(T, Y)$ is an irreducible polynomial in $\mathbb{Q}[T, Y]$. Now we employ Theorem 4.7 on $P(T, Y)$ with $B = 2^{200s}$, we have $m \leq 2^s, n = 2$ and $H \leq 2^{2^s}$. In this case, we have $S(P, B) = S(f)$. By using Theorem 4.7, we have:

$$S(f) \leq 2^{165} 2^{64s} 2^{592} 2^{19s} 2^{100s} (200s)^5 < 2^{800} s^5 2^{183s}. \qquad ◀$$

Corollary 4.8 implies a randomized polynomial time algorithm for SqPolySLP, as demonstrated below in Theorem 4.9.

▶ **Theorem 4.9.** SqPolySLP *is in* coRP.

Proof. Given an integer polynomial $f(x)$ computed by a SLP of size s, we want to decide if $f = g^2$ for some $g \in \mathbb{Z}[x]$. We sample a positive integer uniformly at random from the set $\{1 \leq t \leq 2^{200s} \mid t \in \mathbb{N}\}$. Using the algorithm in [17, Sec 4.2], we test if $f(t)$ is a square. We output "Yes" if $f(t)$ is a square. If $f = g^2$ for some $g \in \mathbb{Z}[x]$, then we always output "Yes". Suppose $f \neq g^2$ for any $g \in \mathbb{Z}[x]$. By using Corollary 4.8, we obtain that:

$$\Pr[f(t) \text{ is a square}] < \frac{2^{800}s^5 2^{183s}}{2^{200s}} < \frac{1}{100} \text{ for } s > 100.$$

Hence with probability at least 0.99 we sample a t such that $f(t)$ is not a square. The algorithm for SquSLP verifies that $f(t)$ is not a square with probability at least $\frac{1}{3}$ [17, Sec 4.2]. Hence we output "No" with probability at least 0.33. This implies SqPolySLP $\in$ coRP. ◀

5 Conclusion and Open Problems

We studied the connection between PosSLP and problems related to the representation of integers as sums of squares, drawing on Lagrange's four-square theorem from 1770. We investigated variants of the problem, considering whether the positive integer computed by a given SLP can be represented as the sum of squares of two or three integers. We analyzed the complexity of these variations and established relationships between them and the original PosSLP problem. Additionally, we introduced the Div2SLP problem, which involves determining if a given SLP computes an integer divisible by a given power of 2. We showed that Div2SLP is at least as hard as DegSLP. We also showed the relevance of Div2SLP in connecting the 3SoSSLP to PosSLP. In contrast to PosSLP, we also showed that the polynomial variant of the PosSLP problem is unconditionally coNP-hard. Overall, this paper contributes to a deeper understanding of decision problems associated with SLPs and provides insights into the computational complexity of problems related to the representation of integers as sums of squares. A visual representation illustrating the problems discussed in this paper and their interrelations is available in Figure 1. Our results open avenues for further research in this area; in particular, we highlight the following research avenues:

1. What is the complexity of Div2SLP? We showed it is DegSLP hard. Is it NP-hard too? How does it relate to PosSLP?
2. Can we prove Theorem 1.14 without relying on Conjecture 3.3?
3. One can also study the problems of deciding whether a given SLP computes an integer univariate polynomial, which can be written as the sum of two, three, or four squares. We studied these questions for integers in this paper. But it makes for an interesting research to study these questions for polynomials.
4. And finally, can we prove unconditional hardness results for PosSLP?

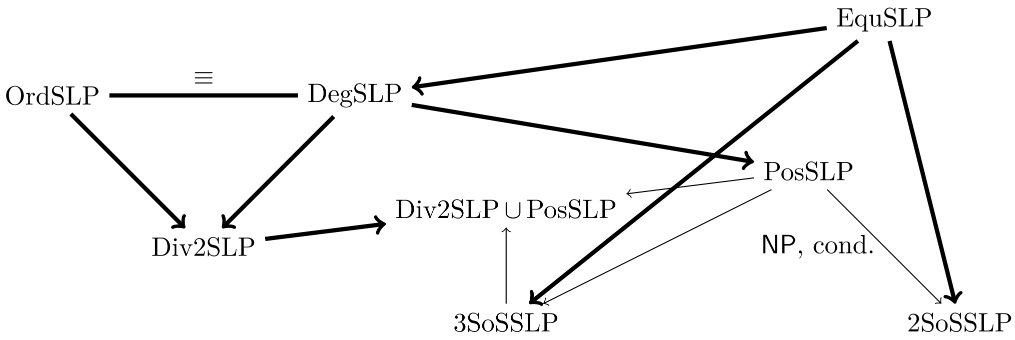


Figure 1 A visualization of the relations between the problems studied in this work. An arrow means that there is a Turing reduction. A thicker arrow indicates a polynomial time many-one reduction. The reduction from PosSLP to 2SoSSLP is nondeterministic and depends on Conjecture 3.3.

References

1 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM Journal on Computing*, 38(5):1987–2006, 2009. doi:10.1137/070697926.

2 Eric Allender, Michal Koucký, Detlef Ronneburger, Sambuddha Roy, and V. Vinay. Time-space tradeoffs in the counting hierarchy. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 295–302. IEEE Computer Society, 2001. doi:10.1109/CCC.2001.933896.

3 N. C. Ankeny. Sums of three squares. *Proceedings of the American Mathematical Society*, 8(2):316–319, 1957. doi:10.1090/s0002-9939-1957-0085275-8.

4 Saugata Basu, Richard Leroy, and Marie-Francoise Roy. A bound on the minimum of a real positive polynomial over the standard simplex, 2009. arXiv:0902.3304.

5 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation.* Springer-Verlag, Berlin, Heidelberg, 1997.

6 Markus Bläser, Julian Dörfler, and Gorav Jindal. Posslp and sum of squares, 2024. doi:10.48550/arXiv.2403.00115.

7 R. Breusch. Zur verallgemeinerung des bertrandschen postulates, dass zwischen x und 2x stets primzahlen liegen. *Mathematische Zeitschrift*, 34:505–526, 1932. URL: http://eudml.org/doc/168326.

8 Peter Bürgisser and Felipe Cucker. Counting complexity classes for numeric computations ii: Algebraic and semialgebraic sets. *Journal of Complexity*, 22(2):147–191, 2006. doi:10.1016/j.jco.2005.11.001.

9 Peter Bürgisser and Gorav Jindal. *On the Hardness of PosSLP*, pages 1872–1886. Society for Industrial and Applied Mathematics, 2024. doi:10.1137/1.9781611977912.75.

10 Pierre Debes and Yann Walkowiak. Bounds for hilbert's irreducibility theorem. *Pure and Applied Mathematics Quarterly*, 4(4):1059–1083, 2008. doi:10.4310/pamq.2008.v4.n4.a4.

11 U. Dudley. *Elementary Number Theory: Second Edition.* Dover Books on Mathematics. Dover Publications, 2012.

12 Pranjal Dutta, Gorav Jindal, Anurag Pandey, and Amit Sinhababu. Arithmetic Circuit Complexity of Division and Truncation. In Valentine Kabanets, editor, *36th Computational Complexity Conference (CCC 2021)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:36, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CCC.2021.25.

13 Pranjal Dutta, Nitin Saxena, and Amit Sinhababu. Discovering the roots: Uniform closure results for algebraic classes under factoring. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 1152–1165, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3188745.3188760.

14 Landau Edmund. Über die darstellung definiter funktionen durch quadrate. *Mathematische Annalen*, 62:272–285, 1906. URL: http://eudml.org/doc/158257.

15 C.F. Gauss. *Disquisitiones arithmeticae.* Apud G. Fleischer, 1801. URL: https://books.google.de/books?id=OwX6GwAACAAJ.

16 David R. Hilbert. Beweis für die darstellbarkeit der ganzen zahlen durch eine feste anzahlnter potenzen (waringsches problem). *Mathematische Annalen*, 67:281–300, 1909.

17 Gorav Jindal and Louis Gaillard. On the order of power series and the sum of square roots problem. In *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*, ISSAC '23, pages 354–362, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3597066.3597079.

18 E. Kaltofen. Single-factor hensel lifting and its application to the straight-line complexity of certain polynomials. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 443–452, New York, NY, USA, 1987. Association for Computing Machinery. doi:10.1145/28395.28443.

19 Swastik Kopparty, Shubhangi Saraf, and Amir Shpilka. Equivalence of polynomial identity testing and deterministic multivariate polynomial factorization. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 169–180. IEEE, 2014. doi:10.1109/CCC.2014.25.

20 Alexei Kourbatov. On the distribution of maximal gaps between primes in residue classes, 2018. arXiv:1610.03340.

21 E. Landau. Über die einteilung der positiven ganzen zahlen in vier klassen nach der mindestzahl der zu ihrer additiven zusammensetzung erforderlichen quadrate. *Arch. Math. und Physik (3)*, 13, 1908.

22 Edmund Landau. Über die Einteilung der positiven ganzen Zahlen in vier Klassen nach der Mindestzahl der zu ihrer additiven Zusammensetzung erforderliche Quadrate, 1908. URL: https://books.google.de/books?id=e3XBnQAACAAJ.

23 Adrien Marie Legendre. *Essai Sur La Théorie Des Nombres.* Duprat, 1797. URL: http://eudml.org/doc/204253.

24 Gregorio Malajovich. An effective version of kronecker's theorem on simultaneous diophantine approximation. Technical report, Citeseer, 1996.

25 LJ Mordell. On the representation of a number as a sum of three squares. *Rev. Math. Pures Appl*, 3:25–27, 1958.

26 M Ram Murty. Polynomials assuming square values. *Number theory and discrete geometry*, pages 155–163, 2008.

27 Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. *An Introduction to the Theory of Numbers.* Wiley, hardcover edition, January 1991. URL: https://lead.to/amazon/com/?op=bt&la=en&cu=usd&key=0471625469.

28 Daniel Perrucci and Juan Sabia. Real roots of univariate polynomials and straight line programs. *Journal of Discrete Algorithms*, 5(3):471–478, 2007. Selected papers from Ad Hoc Now 2005. doi:10.1016/j.jda.2006.10.002.

29 Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

30 Yaroslav Shitov. How hard is the tensor rank?, 2016. arXiv:1611.01559.

31 Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic ram. *Journal of Complexity*, 8(4):393–397, 1992. doi:10.1016/0885-064X(92)90003-T.

32 Yann Walkowiak. Théorème d'irréductibilité de hilbert effectif. *Acta Arithmetica*, 116(4):343–362, 2005. URL: http://eudml.org/doc/277977.

33 Pourchet Y. Sur la représentation en somme de carrés des polynômes à une indéterminée sur un corps de nombres algébriques. *Acta Arithmetica*, 19(1):89–104, 1971. URL: http://eudml.org/doc/205020.

A Missing Proofs

▶ **Lemma A.1.** *For every positive polynomial $f \in \mathbb{R}[x]$, there exist polynomials $g, h \in \mathbb{R}[x]$ such that $f = g^2 + h^2$.*

Proof. Let $f(x) \in \mathbb{R}[x]$ be a polynomial such that $f(x) \geq 0$ for all $x \in \mathbb{R}$. We aim to express f as a sum of two squares of real polynomials.

If $\alpha \in \mathbb{R}$ is a real root of f, it must have even multiplicity, as $f(x)$ is non-negative. Specifically, if α is a root of multiplicity $2k$, we can write:

$$(x-\alpha)^{2k} = \left((x-\alpha)^k\right)^2 + 0^2,$$

which is already in the form of a sum of two squares.

If f has complex-conjugate roots, say $\beta = s + \imath t$ and $\overline{\beta} = s - \imath t$ with $t \neq 0$, the quadratic factor associated with these roots is:

$$(x-\beta)(x-\overline{\beta}) = (x-s)^2 + t^2,$$

which is clearly a sum of two squares. Since $f(x)$ is the product of factors corresponding to its real roots and complex-conjugate pairs of roots, we now combine these factors by using the following identity:

$$(a^2+b^2)(c^2+d^2)=(ac-bd)^2+(ad+bc)^2.$$

By applying this identity iteratively to the factors of $f(x)$, we can express $f(x)$ as a sum of two squares of real polynomials. ◀

B Alternative Proof of Lemma 2.12

We prove a general theorem on how to show that problems involving SLPs are in CH. It is similar to the proof of [2, Lemma 5]. Let C be a Boolean circuit in TC_0. C consists of unbounded AND, unbounded OR, and unbounded majority gates (MAJ). According to [29], when a family (C_n) is in dlogtime-uniform TC_0, this means that there is a deterministic Turing machine (DTM) that decides in time $O(\log n)$ whether given (n,f,g) the gate f is connected to the gate g and whether given (n,f,t) the gate f has type t. All numbers are given in binary. For a language $B\subseteq\{0,1\}^*$, let $\mathrm{SLP}(B)$ be the language:

$$\mathrm{SLP}(B):=\{P\mid P\text{ is an SLP computing a number }N\text{ such that }\mathrm{Bin}\,(N)\in B\}$$

This can be viewed as the “SLP-version” of B.

▶ **Lemma B.1.** *Let B be in dlogtime-uniform TC_0. Then $\mathrm{SLP}(B)\in\mathsf{CH}$.*

Proof. The proof is by induction on the depth. We prove the more general statement: Let M be a DTM from the definition of dlogtime and (C_n) be the sequence of circuits for B. Let P be the given SLP encoding a number N. Given (P,g,b) we can decide in CH_{t+c} whether the value of the gate g on input N given by P is b. t is the depth of g. If $t=0$, then g is an input gate. Thus this problem is BitSLP which is in CH_c for some c. If $t>0$, then we have to decide whether the majority of the gates that are children of g are 1. This can be done using a PP-machine with oracle to CH_{c+t-1}. We guess a gate f and check using the DTM M whether f is a predecessor of g. If not, we add an accepting and rejecting path. If yes, we use the oracle to check whether f has value 1. If yes, we accept and otherwise, we reject. ◀

It is easy to see that checking whether the ℓ least significant bits of a number given in binary are 0 can be done in dlogtime-uniform TC_0. Thus Div2SLP is in CH by Lemma B.1 above.

C Reduction from multivariate DegSLP to univariate DegSLP

We use mDegSLP to denote the multivariate variant of the DegSLP problem, which we define formally below.

▶ **Problem C.1** (mDegSLP). *Given a straight line program representing a polynomial $f\in\mathbb{Z}[x_1,x_2,\ldots,x_n]$, and given a natural number d in binary, decide whether $\deg(f)\le d$.*

mDegSLP was simply called DegSLP in [1]. Now we recall the proof in [1], to show that to study the hardness of mDegSLP, it is enough to focus on its univariate variant DegSLP. To this end, we note the following Observation C.2.

▶ **Observation C.2** ([1]). mDegSLP *is equivalent to* DegSLP *under deterministic polynomial time many-one reductions.*

Proof. We only need to show that mDegSLP reduces o DegSLP under deterministic polynomial time many-one reductions, other direction is trivial. Suppose we are given an SLP of size s which computes $f \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$, and we want to decide whether $\deg(f) \leq d$ for a given $d \in \mathbb{N}$. Suppose $D = \deg(f)$. For all $i \in \{0, 1, \ldots, D\}$, we use f_i to denote the homogeneous degree i part of f. Now notice that for any $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{Z}^d$, we have:

$$f(y\boldsymbol{\alpha}) = f(y\alpha_1, y\alpha_2, \ldots, y\alpha_n) = \sum_{i=0}^{D} y^i f_i(\boldsymbol{\alpha}),$$

where y is a fresh variable. So if $\boldsymbol{\alpha}$ is chosen such that $f_D(\boldsymbol{\alpha})$ is non-zero, then $\deg(f(y\boldsymbol{\alpha})) = \deg(f) = D$. If we choose $\alpha_i = 2^{2^{is^2}}$ then it can be seen that $f_D(\boldsymbol{\alpha})$ is non-zero, see e.g. [1, Proof of Proposition 2.2]. SLPs computing α_i can be constructed using iterated squaring in polynomial time. Hence we can construct an SLP for $f(y\boldsymbol{\alpha})$ in polynomial time. By this argument, we know that $\deg(f(y\boldsymbol{\alpha})) \leq d$ if and only if $\deg(f) \leq d$. Therefore mDegSLP reduces to DegSLP under polynomial time many-one reductions. ◀

Unifying Asynchronous Logics for Hyperproperties

Alberto Bombardelli
Fondazione Bruno Kessler, Trento, Italy

Laura Bozzelli
University of Napoli “Federico II”, Italy

César Sánchez
IMDEA Software Institute, Madrid, Spain

Stefano Tonetta
Fondazione Bruno Kessler, Trento, Italy

Abstract

We introduce and investigate a powerful hyper logical framework in the linear-time setting that we call *generalized* HyperLTL *with stuttering and contexts* ($\mathsf{GHyperLTL}_{\mathsf{S+C}}$ for short). $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ unifies the asynchronous extensions of HyperLTL called $\mathsf{HyperLTL}_{\mathsf{S}}$ and $\mathsf{HyperLTL}_{\mathsf{C}}$, and the well-known extension KLTL of LTL with knowledge modalities under both the synchronous and asynchronous perfect recall semantics. As a main contribution, we identify a meaningful fragment of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, that we call *simple* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, with a decidable model-checking problem, which is more expressive than HyperLTL and known fragments of asynchronous extensions of HyperLTL with a decidable model-checking problem. Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ subsumes KLTL under the synchronous semantics and the one-agent fragment of KLTL under the asynchronous semantics and to the best of our knowledge, it represents the unique hyper logic with a decidable model-checking problem which can express powerful non-regular trace properties when interpreted on singleton sets of traces. We justify the relevance of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ by showing that it can express diagnosability properties, interesting classes of information-flow security policies, both in the synchronous and asynchronous settings, and bounded termination (more in general, global promptness in the style of Prompt LTL).

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Asynchronous hyperproperties, Temporal logics for hyperproperties, Expressiveness, Decidability, Model checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.14

Related Version *Full Version*: `https://arxiv.org/abs/2404.16778` [4]

Funding *César Sánchez*: funded in part by PRODIGY Project (TED2021-132464B-I00) – funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU/PRTR – by DECO Project (PID2022-138072OB-I00) – funded by MCIN/AEI/10.13039/501100011033 and by the ESF+ – and by a research grant from Nomadic Labs and the Tezos Foundation.

1 Introduction

Temporal logics [27] play a fundamental role in the formal verification of the dynamic behaviour of complex reactive systems. Classic *regular* temporal logics such as LTL, CTL, and CTL* [29, 13] are suited for the specification of *trace properties* which describe the ordering of events along individual execution traces of a system. In the last 15 years, a novel specification paradigm has been introduced that generalizes traditional regular trace properties by properties of sets of traces, the so called *hyperproperties* [10]. Hyperproperties relate distinct traces and are useful to formalize a wide range of properties of prime interest which go, in general, beyond regular properties and cannot be expressed in standard regular temporal logics. A relevant example concerns information-flow security policies like noninterference [18, 28] and observational determinism [36] which compare observations made by an external

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 14; pp. 14:1–14:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

low-security agent along traces resulting from different values of not directly observable inputs. Other examples include bounded termination of programs, diagnosability of critical systems (which amounts to checking whether the available sensor information is sufficient to infer the presence of faults on the hidden behaviour of the system) [31, 5, 3], and epistemic properties describing the knowledge of agents in distributed systems [23, 33, 22].

In the context of model checking of finite-state reactive systems, many temporal logics for hyperproperties have been proposed [12, 9, 6, 30, 15, 11, 20] for which model checking is decidable, including HyperLTL [9], HyperCTL* [9], HyperQPTL [30, 11], and HyperPDL$-\Delta$ [20] which extend LTL, CTL*, QPTL [32], and PDL [17], respectively, by explicit first-order quantification over traces and trace variables to refer to multiple traces at the same time. The semantics of all these logics is *synchronous*: the temporal modalities are evaluated by a lockstepwise traversal of all the traces assigned to the quantified trace variables. Other approaches for the formalization of synchronous hyper logics are either based on hyper variants of monadic second-order logic over traces or trees [11], or the adoption of a *team semantics* for standard temporal logics, in particular, LTL [24, 26, 35]. For the first approach in the linear-time setting, we recall the logic S1S[E] [11] (and its first-order fragment FO[<,E] [16]) which syntactically extends monadic second-order logic of one successor S1S with the *equal-level predicate* E, which relates the same time point on different traces. More recently, an extension of HyperLTL with second-order quantification over traces has been introduced [2] which allows to express common knowledge in multi-agent distributed systems. Like S1S[E], model checking of this extension of HyperLTL is highly undecidable [2].

Hyper logics supporting asynchronous features have been introduced recently [21, 1, 7]. These logics allow to relate traces at distinct time points which can be arbitrarily far from each other. Asynchronicity is ubiquitous in many real-world systems, for example, in multithreaded environments in which threads are not scheduled lockstepwise, and traces associated with distinct threads progress with different speed. Asynchronous hyperproperties are also useful in information-flow security and diagnosability settings where an observer cannot distinguish consecutive time points along an execution having the same observations. This requires to match asynchronously sequences of observations along distinct execution traces. The first systematic study of asynchronous hyperproperties was done by Gutsfeld et al. [21], who introduced the temporal fixpoint calculus H_μ and its automata-theoretic counterpart for expressing such properties in the linear-time setting.

More recently, three temporal logics [1, 7] which syntactically extend HyperLTL have been introduced for expressing asynchronous hyperproperties: *Asynchronous* HyperLTL (A-HyperLTL) [1] and *Stuttering* HyperLTL (HyperLTL$_\mathsf{S}$) [7], both useful for asynchronous security analysis, and *Context* HyperLTL (HyperLTL$_\mathsf{C}$) [7], useful for expressing hyper-bounded-time response requirements. The logic A-HyperLTL, which is expressively incomparable with both HyperLTL and HyperLTL$_\mathsf{S}$ [8], models asynchronicity by means of an additional quantification layer over the so called *trajectories* which control the relative speed at which traces progress by choosing at each instant which traces move and which traces stutter. On the other hand, the logic HyperLTL$_\mathsf{S}$ exploits relativized versions of the temporal modalities with respect to finite sets Γ of LTL formulas: these modalities are evaluated by a lockstepwise traversal of the sub-traces of the given traces which are obtained by removing "redundant" positions with respect to the pointwise evaluation of the LTL formulas in Γ. Finally, the logic HyperLTL$_\mathsf{C}$ is more expressive than HyperLTL and is not expressively subsumed by either A-HyperLTL or HyperLTL$_\mathsf{S}$ [8]. HyperLTL$_\mathsf{C}$ extends HyperLTL by unary modalities $\langle C \rangle$ parameterized by a non-empty subset C of trace variables – called the *context* – which restrict the evaluation of the temporal modalities to the traces associated with the variables in C.

Note that the temporal modalities in $\mathsf{HyperLTL_C}$ are evaluated by a lockstepwise traversal of the traces assigned to the variables in the current context, and unlike $\mathsf{HyperLTL}$, the current time points of these traces from which the evaluation starts are in general different. It is known that these three syntactical extensions of $\mathsf{HyperLTL}$ are less expressive than H_μ [8] and like H_μ, model checking the respective quantifier alternation-free fragments are already undecidable [1, 7]. The works [1, 7] identify practical fragments of the logics $\mathsf{A\text{-}HyperLTL}$ and $\mathsf{HyperLTL_S}$ with a decidable model checking problem. In particular, we recall the so called *simple fragment* of $\mathsf{HyperLTL_S}$ [7], which is more expressive than $\mathsf{HyperLTL}$ [8] and can specify interesting security policies in both the asynchronous and synchronous settings.

Formalization of asynchronous hyperproperties in the *team semantics setting* following an approach similar to the *trajectory construct* of $\mathsf{A\text{-}HyperLTL}$ has been investigated in [19]. It is worth noting that unlike other hyper logics (including logics with team semantics) which only capture regular trace properties when interpreted on singleton sets of traces, the logics $\mathsf{HyperLTL_C}$, $\mathsf{A\text{-}HyperLTL}$, and H_μ can express non-regular trace properties [8].

Our contribution. Specifications in $\mathsf{HyperLTL}$ and in the known asynchronous extensions of $\mathsf{HyperLTL}$, whose most expressive representative is H_μ [21], consist of a prefix of trace quantifiers followed by a quantifier-free formula which expresses temporal requirements on a fixed number of traces. Thus, these hyper logics lack mechanisms to relate directly an unbounded number of traces, which are required for example to express bounded termination or diagnosability properties [31, 5, 3]. This ability is partially supported by temporal logics with team semantics [24, 26, 35] and extensions of temporal logics with the knowledge modalities of epistemic logic [14], which relate computations whose histories are observationally equivalent for a given agent. In this paper, we introduce and investigate a hyper logical framework in the linear-time setting which unifies two known asynchronous extensions of $\mathsf{HyperLTL}$ and the well-known extension KLTL [23] of LTL with knowledge modalities under both the synchronous and asynchronous perfect recall semantics (where an agent remembers the whole sequence of its observations). The novel logic, that we call *generalized* $\mathsf{HyperLTL}$ *with stuttering and contexts* ($\mathsf{GHyperLTL_{S+C}}$ for short), merges $\mathsf{HyperLTL_S}$ and $\mathsf{HyperLTL_C}$ and adds two new natural modeling facilities: past temporal modalities for asynchronous hyperproperties and general trace quantification where trace quantifiers can occur in the scope of temporal modalities. Past temporal modalities used in combination with context modalities provide a powerful mechanism to compare histories of computations at distinct time points. Moreover, unrestricted trace quantification allows to relate an unbounded number of traces.

As a main contribution, we identify a meaningful fragment of $\mathsf{GHyperLTL_{S+C}}$ with a decidable model-checking problem, that we call *simple* $\mathsf{GHyperLTL_{S+C}}$. This fragment is obtained from $\mathsf{GHyperLTL_{S+C}}$ by carefully imposing restrictions on the use of the stuttering and context modalities. Simple $\mathsf{GHyperLTL_{S+C}}$ allows quantification over arbitrary *pointed* traces (i.e., traces plus time points) in the style of $\mathsf{FO[<,E]}$ [16], it is more expressive than the simple fragment of $\mathsf{HyperLTL_S}$ [7], and it is expressively incomparable with full $\mathsf{HyperLTL_S}$ and $\mathsf{S1S[E]}$. Moreover, this fragment subsumes both KLTL under the synchronous semantics and the one-agent fragment of KLTL under the asynchronous semantics. In fact, simple $\mathsf{GHyperLTL_{S+C}}$ can be seen as a very large fragment of $\mathsf{GHyperLTL_{S+C}}$ with a decidable model checking problem which (1) strictly subsumes $\mathsf{HyperLTL}$ and the simple fragment of $\mathsf{HyperLTL_S}$, (2) is closed under Boolean connectives, and (3) allows an unrestricted nesting of temporal modalities. We justify the relevance of simple $\mathsf{GHyperLTL_{S+C}}$ by showing that it can express diagnosability properties, interesting classes of information-flow security policies, both in the synchronous and asynchronous settings, and bounded termination (more in general, global promptness in the style of Prompt LTL [25]). To the best of our knowledge, simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ represents the unique hyper logic with a decidable model-checking problem which can express powerful non-regular trace properties when interpreted over singleton sets of traces.

2 Background

We denote by $\mathbb{N}$ the set of natural numbers. Given $i, j \in \mathbb{N}$, we write $[i, j]$ for the set of natural numbers h such that $i \leq h \leq j$, we use $[i, j)$ for the set $[i, j] \setminus \{j\}$, we use $(i, j]$ for the set $[i, j] \setminus \{i\}$, and $[i, \infty)$ for the set of natural numbers h such that $h \geq i$. Given a word w over some alphabet Σ, $|w|$ is the length of w ($|w| = \infty$ if w is infinite). For each $0 \leq i < |w|$, $w(i)$ is the $(i+1)^{th}$ symbol of w, w^i is the suffix of w from position i, that is, the word $w(i)w(i+1)\ldots$, and $w[0, i]$ is the prefix of w that ends at position i.

We fix a finite set AP of atomic propositions. A *trace* is an infinite word over 2^{AP}, while a *finite trace* is a nonempty finite word over 2^{AP}. A *pointed trace* is a pair (σ, i) consisting of a trace σ and a position (timestamp) $i \in \mathbb{N}$ along σ.

Kripke structures. We define the dynamic behaviour of reactive systems by *Kripke structures* $\mathcal{K} = \langle S, S_0, E, Lab \rangle$ over a finite set AP of atomic propositions, where S is a set of states, $S_0 \subseteq S$ is the set of initial states, $E \subseteq S \times S$ is a transition relation which is total in the first argument (i.e., for each $s \in S$ there is $s' \in S$ with $(s, s') \in E$), and $Lab : S \to 2^{\mathsf{AP}}$ is a labeling map assigning to each state s the set of propositions holding at s. The Kripke structure $\mathcal{K}$ is finite if S is finite. A *path* π of $\mathcal{K}$ is an infinite word $\pi = s_0, s_1, \ldots$ over S such that $s_0 \in S_0$ and for all $i \geq 0$, $(s_i, s_{i+1}) \in E$. The path $\pi = s_0, s_1, \ldots$ induces the trace $Lab(s_0)Lab(s_1)\ldots$. A *trace of* $\mathcal{K}$ is a trace induced by some path of $\mathcal{K}$. We denote by $\mathcal{L}(\mathcal{K})$ the set of traces of $\mathcal{K}$. A *finite path* of $\mathcal{K}$ is a non-empty infix of some path of $\mathcal{K}$. We also consider *fair finite Kripke structures* $(\mathcal{K}, F)$, that is, finite Kripke structures $\mathcal{K}$ equipped with a subset F of $\mathcal{K}$-states. A path π of $\mathcal{K}$ is F*-fair* if π visits infinitely many times some state in F. We denote by $\mathcal{L}(\mathcal{K}, F)$ the set of traces of $\mathcal{K}$ associated with the F-fair paths of $\mathcal{K}$.

Standard LTL with past (PLTL for short) [29]. Formulas ψ of PLTL over the given finite set AP of atomic propositions are defined by the following grammar:

$$\psi ::= \top \mid p \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \mathbf{Y}\psi \mid \psi\,\mathbf{U}\,\psi \mid \psi\,\mathbf{S}\,\psi$$

where $p \in \mathsf{AP}$, $\mathbf{X}$ and $\mathbf{U}$ are the *next* and *until* temporal modalities respectively, and $\mathbf{Y}$ (*previous* or *yesterday*) and $\mathbf{S}$ (*since*) are their past counterparts. LTL is the fragment of PLTL that does not contain the past temporal modalities $\mathbf{Y}$ and $\mathbf{S}$. We also use the following abbreviations: $\mathbf{F}\psi := \top\,\mathbf{U}\,\psi$ (*eventually*), $\mathbf{O}\psi := \top\,\mathbf{S}\,\psi$ (*past eventually* or *once*), and their duals $\mathbf{G}\psi := \neg\,\mathbf{F}\,\neg\psi$ (*always*) and $\mathbf{H}\psi := \neg\,\mathbf{O}\,\neg\psi$ (*past always* or *historically*).

The semantics of PLTL is defined over pointed traces (σ, i). The satisfaction relation $(\sigma, i) \models \psi$, that defines whether formula ψ holds at position i along σ, is inductively defined as follows (we omit the semantics for the Boolean connectives which is standard):

$$\begin{array}{ll}
(\sigma, i) \models p & \Leftrightarrow p \in \sigma(i) \\
(\sigma, i) \models \mathbf{X}\psi & \Leftrightarrow (\sigma, i+1) \models \psi \\
(\sigma, i) \models \mathbf{Y}\psi & \Leftrightarrow i > 0 \text{ and } (\sigma, i-1) \models \psi \\
(\sigma, i) \models \psi_1\,\mathbf{U}\,\psi_2 & \Leftrightarrow \text{for some } j \geq i : (\sigma, j) \models \psi_2 \text{ and } (\sigma, k) \models \psi_1 \text{ for all } i \leq k < j \\
(\sigma, i) \models \psi_1\,\mathbf{S}\,\psi_2 & \Leftrightarrow \text{for some } j \leq i : (\sigma, j) \models \psi_2 \text{ and } (\sigma, k) \models \psi_1 \text{ for all } j < k \leq i
\end{array}$$

A trace σ is a *model* of ψ, written $\sigma \models \psi$, whenever $(\sigma, 0) \models \psi$.

The logic HyperLTL [9]. The syntax of HyperLTL formulas φ over the given finite set AP of atomic propositions and a finite set VAR of trace variables is as follows:

$$\varphi := \exists x.\, \varphi \mid \forall x.\, \varphi \mid \psi \qquad \psi := \top \mid p[x] \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi\, \mathbf{U}\, \psi$$

where $p \in \mathsf{AP}$, $x \in \mathsf{VAR}$, and $\exists x$ and $\forall x$ are the *hyper* existential and universal trace quantifiers for variable x, respectively, which allow relating different traces of the given set of traces. Note that a HyperLTL formula consists of a prefix of traces quantifiers followed by a quantifier-free formula, where the latter corresponds to an LTL formula whose atomic propositions p are replaced with x-relativized versions $p[x]$. Intuitively, $p[x]$ asserts that p holds at the pointed trace assigned to variable x. A *sentence* is a formula where each relativized proposition $p[x]$ occurs in the scope of trace quantifier $\exists x$ or $\forall x$.

In order to define the semantics of HyperLTL, we need additional definitions. The *successor* $succ(\sigma, i)$ of a pointed trace (σ, i) is the pointed trace $(\sigma, i+1)$, which captures the standard local successor of a position along a trace.

Given a set of traces $\mathcal{L}$, a *(pointed) trace assignment* over $\mathcal{L}$ is a partial mapping $\Pi : \mathsf{VAR} \to \mathcal{L} \times \mathbb{N}$ assigning to each trace variable x – where Π is defined – a pointed trace (σ, i) such that $\sigma \in \mathcal{L}$. We use $Dom(\Pi)$ to refer to the trace variables for which Π is defined. The *successor* $succ(\Pi)$ *of* Π is the trace assignment over $\mathcal{L}$ having domain $Dom(\Pi)$ such that $succ(\Pi)(x) = succ(\Pi(x))$ for each $x \in Dom(\Pi)$. For each $i \geq 0$, we use $succ^i$ for the function obtained by i applications of the function $succ$: $succ^0(\Pi) := \Pi$ and $succ^{i+1}(\Pi) := succ(succ^i(\Pi))$.

Given $x \in \mathsf{VAR}$ and a pointed trace (σ, i) with $\sigma \in \mathcal{L}$, we denote by $\Pi[x \mapsto (\sigma, i)]$ the trace assignment that is identical to Π besides for x, which is mapped to (σ, i).

Given a formula φ, a set of traces $\mathcal{L}$, and a trace assignment Π over $\mathcal{L}$ such that $Dom(\Pi)$ contains all the trace variables occurring free in φ, the satisfaction relation $\Pi \models_{\mathcal{L}} \varphi$ is inductively defined as follows (we again omit the semantics of the Boolean connectives):

$$\begin{array}{ll}
\Pi \models_{\mathcal{L}} p[x] & \Leftrightarrow \Pi(x) = (\sigma, i) \text{ and } p \in \sigma(i) \\
\Pi \models_{\mathcal{L}} \exists x.\, \varphi & \Leftrightarrow \text{ for some } \sigma \in \mathcal{L},\ \Pi[x \mapsto (\sigma, 0)] \models_{\mathcal{L}} \varphi \\
\Pi \models_{\mathcal{L}} \forall x.\, \varphi & \Leftrightarrow \text{ for all } \sigma \in \mathcal{L},\ \Pi[x \mapsto (\sigma, 0)] \models_{\mathcal{L}} \varphi \\
\Pi \models_{\mathcal{L}} \mathbf{X}\psi & \Leftrightarrow succ(\Pi) \models \psi \\
\Pi \models_{\mathcal{L}} \psi_1\, \mathbf{U}\, \psi_2 & \Leftrightarrow \text{for some } i \geq 0 :\ succ^i(\Pi) \models_{\mathcal{L}} \psi_2 \text{ and } succ^j(\Pi) \models_{\mathcal{L}} \psi_1 \text{ for all } 0 \leq j < i
\end{array}$$

Note that trace quantification ranges over *initial* pointed traces $(\sigma, 0)$ over $\mathcal{L}$ (the timestamp is 0). As an example, the sentence $\forall x_1.\, \forall x_2.\, \bigwedge_{p \in \mathsf{AP}} \mathbf{G}(p[x_1] \leftrightarrow p[x_2])$ captures the sets of traces which are singletons.

For a sentence φ and a set of traces $\mathcal{L}$, $\mathcal{L}$ is a *model* of φ, written $\mathcal{L} \models \varphi$, if $\Pi_\emptyset \models_{\mathcal{L}} \varphi$ where $\Pi_\emptyset$ is the trace assignment with empty domain.

3 Unifying Framework for Asynchronous Extensions of HyperLTL

In this section, we introduce a novel logical framework for specifying both asynchronous and synchronous linear-time hyperproperties which unifies two known more expressive extensions of HyperLTL [9], namely *Stuttering* HyperLTL ($\mathsf{HyperLTL_S}$ for short) [7] and *Context* HyperLTL ($\mathsf{HyperLTL_C}$ for short) [7]. The proposed hyper logic, that we call *generalized* HyperLTL *with stuttering and contexts* ($\mathsf{GHyperLTL_{S+C}}$ for short), merges $\mathsf{HyperLTL_S}$ and $\mathsf{HyperLTL_C}$ and adds two new features: past temporal modalities for asynchronous/synchronous hyperproperties and general trace quantification where trace quantifiers can occur in the scope of temporal modalities. Since model checking of the logics $\mathsf{HyperLTL_S}$ and $\mathsf{HyperLTL_C}$ is already undecidable [7], we also consider a meaningful fragment of $\mathsf{GHyperLTL_{S+C}}$ which is strictly more expressive than the known *simple fragment* of $\mathsf{HyperLTL_S}$ [7]. Our fragment is able to express relevant classes of hyperproperties and, as we show in Section 4, its model checking problem is decidable.

3.1 PLTL-Relativized Stuttering and Context Modalities

Classically, a trace is stutter-free if there are no consecutive positions having the same propositional valuation unless the valuation is repeated ad-infinitum. We can associate to each trace a unique stutter-free trace by removing "redundant" positions. The logic $\mathsf{HyperLTL_S}$ [7] generalizes these notions with respect to the pointwise evaluation of a finite set of LTL formulas. Here, we consider LTL with past (PLTL).

▶ **Definition 3.1** (PLTL stutter factorization [7]). *Let Γ be a finite set of* PLTL *formulas and σ a trace. The* Γ-stutter factorization of σ *is the unique increasing sequence of positions $\{i_k\}_{k\in[0,m_\infty]}$ for some $m_\infty \in \mathbb{N}\cup\{\infty\}$ such that the following holds for all $j < m_\infty$:*

- *$i_0 = 0$ and $i_j < i_{j+1}$;*
- *for each $\theta \in \Gamma$, the truth value of θ along the segment $[i_j, i_{j+1})$ does not change, that is, for all $h, k \in [i_j, i_{j+1})$, $(\sigma, h) \models \theta$ iff $(\sigma, k) \models \theta$, and the same holds for the infinite segment $[m_\infty, \infty]$ in case $m_\infty \neq \infty$;*
- *the truth value of some formula in Γ changes along adjacent segments, that is, for some $\theta \in \Gamma$ (depending on j), $(\sigma, i_j) \models \theta$ iff $(\sigma, i_{j+1}) \not\models \theta$.*

Thus, the Γ-stutter factorization $\{i_k\}_{k\in[0,m_\infty]}$ of σ partitions the trace in adjacent non-empty segments such that the valuation of formulas in Γ does not change within a segment, and changes in moving from a segment to the adjacent ones. This factorization induces in a natural way a trace obtained by selecting the first positions of the finite segments and all the positions of the unique tail infinite segment, if any. These positions form an infinite increasing sequence $\{\ell_k\}_{k\in\mathbb{N}}$ called (Γ,ω)-*stutter factorization* of σ, where:

$$\ell_0, \ell_1, \ldots := \begin{cases} i_0, i_1, \ldots & \text{if } m_\infty = \infty \\ i_0, i_1, \ldots, i_{m_\infty}, i_{m_\infty}+1, \ldots & \text{otherwise} \end{cases}$$

The Γ-*stutter trace $stfr_\Gamma(\sigma)$ of σ* (see [7]) is defined as follows: $stfr_\Gamma(\sigma) := \sigma(\ell_0)\sigma(\ell_1)\ldots$. Note that for $\Gamma = \emptyset$, $stfr_\Gamma(\sigma) = \sigma$. A trace σ is Γ-*stutter free* if it coincides with its Γ-stutter trace, i.e. $stfr_\Gamma(\sigma) = \sigma$.

As an example, assume that $\mathsf{AP} = \{p, q, r\}$ and let $\Gamma = \{p\,\mathbf{U}\,q\}$. Given $h, k \geq 1$, let $\sigma_{h,k}$ be the trace $\sigma_{h,k} = p^h q^k r^\omega$. These traces have the same Γ-stutter trace given by pr^ω.

The semantics of the Γ-relativized temporal modalities in $\mathsf{HyperLTL_S}$ is based on the notion of Γ-*successor $succ_\Gamma(\sigma, i)$* of a pointed trace (σ, i) [7]: $succ_\Gamma(\sigma, i)$ is the pointed trace (σ, ℓ) where ℓ is the smallest position ℓ_j in the (Γ,ω)-stutter factorization $\{\ell_k\}_{k\in\mathbb{N}}$ of σ which is greater than i. Note that for $\Gamma = \emptyset$, $succ_\emptyset(\sigma, i) = succ(\sigma, i) = (\sigma, i+1)$. Hence, $\emptyset$-relativized temporal modalities in $\mathsf{HyperLTL_S}$ correspond to the temporal modalities of HyperLTL.

In this paper we extend $\mathsf{HyperLTL_S}$ with past temporal modalities, so that we introduce the past counterpart of the successor function. The Γ-*predecessor $pred_\Gamma(\sigma, i)$* of a pointed trace (σ, i) is undefined if $i = 0$ (written $pred_\Gamma(\sigma, i) = \mathtt{und}$); otherwise, $pred_\Gamma(\sigma, i)$ is the pointed trace (σ, ℓ) where ℓ is the greatest position ℓ_j in the (Γ,ω)-stutter factorization $\{\ell_k\}_{k\in\mathbb{N}}$ of σ which is smaller than i (since $\ell_0 = 0$ such an ℓ_j exists). Note that for $\Gamma = \emptyset$, $pred_\emptyset(\sigma, i)$ captures the standard local predecessor of a position along a trace.

Successors and predecessors of trace assignments. We now define a generalization of the successor $\mathit{succ}(\Pi)$ of a trace assignment Π in HyperLTL. This generalization is based on the notion of Γ-successor $\mathit{succ}_\Gamma(\sigma, i)$ of a pointed trace (σ, i) and also takes into account the context modalities $\langle C \rangle$ of HyperLTL$_\mathsf{C}$ [7], where a *context* C is a non-empty subset of VAR. Intuitively, modality $\langle C \rangle$ allows reasoning over a subset of the traces assigned to the variables in the formula, by restricting the temporal progress to those traces.

Formally, let Π be a trace assignment over some set of traces $\mathcal{L}$, Γ be a finite set of PLTL formulas, and C be a context. The (Γ, C)*-successor of* Π, denoted by $\mathit{succ}_{(\Gamma,C)}(\Pi)$, is the trace assignment over $\mathcal{L}$ having domain $\mathit{Dom}(\Pi)$, and defined as follows for each $x \in \mathit{Dom}(\Pi)$:

$$\mathit{succ}_{(\Gamma,C)}(\Pi)(x) := \begin{cases} \mathit{succ}_\Gamma(\Pi(x)) & \text{if } x \in C \\ \Pi(x) & \text{otherwise} \end{cases}$$

Note that $\mathit{succ}_{(\emptyset,\mathsf{VAR})}(\Pi) = \mathit{succ}(\Pi)$. Moreover, we define the (Γ, C)*-predecessor* $\mathit{pred}_{(\Gamma,C)}(\Pi)$ *of* Π as follows: $\mathit{pred}_{(\Gamma,C)}(\Pi)$ is *undefined*, written $\mathit{pred}_{(\Gamma,C)}(\Pi) = \mathtt{und}$, if there is $x \in \mathit{Dom}(\Pi)$ such that $\mathit{pred}_\Gamma(\Pi(x)) = \mathtt{und}$. Otherwise, $\mathit{pred}_{(\Gamma,C)}(\Pi)$ is the trace assignment over $\mathcal{L}$ having domain $\mathit{Dom}(\Pi)$, and defined as follows for each $x \in \mathit{Dom}(\Pi)$:

$$\mathit{pred}_{(\Gamma,C)}(\Pi)(x) := \begin{cases} \mathit{pred}_\Gamma(\Pi(x)) & \text{if } x \in C \\ \Pi(x) & \text{otherwise} \end{cases}$$

Finally, for each $i \geq 0$, we define the i^{th} application $\mathit{succ}^i_{(\Gamma,C)}$ of $\mathit{succ}_{(\Gamma,C)}$ and the i^{th} application $\mathit{pred}^i_{(\Gamma,C)}$ of $\mathit{pred}_{(\Gamma,C)}$ as follows, where $\mathit{pred}_{(\Gamma,C)}(\mathtt{und}) := \mathtt{und}$:

- $\mathit{succ}^0_{(\Gamma,C)}(\Pi) := \Pi$ and $\mathit{succ}^{i+1}_{(\Gamma,C)}(\Pi) := \mathit{succ}_{(\Gamma,C)}(\mathit{succ}^i_{(\Gamma,C)}(\Pi))$.
- $\mathit{pred}^0_{(\Gamma,C)}(\Pi) := \Pi$ and $\mathit{pred}^{i+1}_{(\Gamma,C)}(\Pi) := \mathit{pred}_{(\Gamma,C)}(\mathit{pred}^i_{(\Gamma,C)}(\Pi))$.

3.2 Generalized HyperLTL with Stuttering and Contexts

We introduce now the novel logic GHyperLTL$_{\mathsf{S+C}}$. GHyperLTL$_{\mathsf{S+C}}$ formulas φ over AP and a finite set VAR of trace variables are defined by the following syntax:

$$\varphi := \top \mid p[x] \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\, \varphi \mid \langle C \rangle \varphi \mid \mathbf{X}_\Gamma \varphi \mid \mathbf{Y}_\Gamma \varphi \mid \varphi\, \mathbf{U}_\Gamma\, \varphi \mid \varphi\, \mathbf{S}_\Gamma\, \varphi$$

where $p \in \mathsf{AP}$, $x \in \mathsf{VAR}$, $\langle C \rangle$ is the context modality with $\emptyset \neq C \subseteq \mathsf{VAR}$, Γ is a finite set of PLTL formulas, and $\mathbf{X}_\Gamma$, $\mathbf{Y}_\Gamma$, $\mathbf{U}_\Gamma$ and $\mathbf{S}_\Gamma$ are the stutter-relativized versions of the PLTL temporal modalities. Intuitively, the context modality $\langle C \rangle$ restricts the evaluation of the temporal modalities to the traces associated with the variables in C, while the temporal modalities $\mathbf{X}_\Gamma$, $\mathbf{Y}_\Gamma$, $\mathbf{U}_\Gamma$ and $\mathbf{S}_\Gamma$ are evaluated by a lockstepwise traversal of the Γ-stutter traces associated to the traces assigned to the variables in the current context C. Note that the hyper universal quantifier $\forall x$ can be introduced as an abbreviation: $\forall x.\, \varphi \equiv \neg \exists x.\, \neg\varphi$. For a variable x, we write $\langle x \rangle$ instead of $\langle \{x\} \rangle$. Moreover, we write $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{U}$ and $\mathbf{S}$ instead of $\mathbf{X}_\emptyset$, $\mathbf{Y}_\emptyset$, $\mathbf{U}_\emptyset$ and $\mathbf{S}_\emptyset$, respectively. Furthermore, for a PLTL formula ψ and a variable x, $\psi[x]$ is the formula obtained from ψ by replacing each proposition p with its x-version $p[x]$. A *sentence* is a formula where each relativized proposition $p[x]$ occurs in the scope of trace quantifier $\exists x$ or $\forall x$, and each temporal modality occurs in the scope of a trace quantifier.

The known logics HyperLTL$_\mathsf{S}$, HyperLTL$_\mathsf{C}$, and simple HyperLTL$_\mathsf{S}$. A formula φ of GHyperLTL$_{\mathsf{S+C}}$ is in *prenex* form if it is of the form $\mathsf{Q}_1 x_1. \ldots \mathsf{Q}_n x_n. \psi$ where ψ is quantifier-free and $\mathsf{Q}_i \in \{\exists, \forall\}$ for all $i \in [1, n]$. The logics HyperLTL$_\mathsf{S}$ and HyperLTL$_\mathsf{C}$ introduced in [7] correspond to syntactical fragments of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ where the formulas are in prenex form and past temporal modalities are not used. Moreover, in $\mathsf{HyperLTL}_{\mathsf{S}}$, the context modalities are not allowed, while in $\mathsf{HyperLTL}_{\mathsf{C}}$, the subscript Γ of every temporal modality must be the empty set. Note that in HyperLTL [9], both the context modalities and the temporal modalities where the subscript Γ is not empty are disallowed. Finally, we recall the *simple* fragment of $\mathsf{HyperLTL}_{\mathsf{S}}$ [7], which is more expressive than HyperLTL and is parameterized by a finite set Γ of LTL formulas. The *quantifier-free* formulas of simple $\mathsf{HyperLTL}_{\mathsf{S}}$ for the parameter Γ are defined as Boolean combinations of formulas of the form $\psi[x]$, where ψ is an LTL formula, and formulas ψ_Γ defined by the following grammar:

$$\psi_\Gamma := \top \mid p[x] \mid \neg\psi_\Gamma \mid \psi_\Gamma \vee \psi_\Gamma \mid \mathbf{X}_\Gamma \psi_\Gamma \mid \psi_\Gamma \,\mathbf{U}_\Gamma\, \psi_\Gamma$$

Semantics of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$. Given a formula φ, a set of traces $\mathcal{L}$, a trace assignment Π over $\mathcal{L}$ such that $\mathit{Dom}(\Pi)$ contains all the trace variables occurring free in φ, and a context $C \subseteq \mathsf{VAR}$, the satisfaction relation $(\Pi, C) \models_{\mathcal{L}} \varphi$, meaning that the assignment Π over $\mathcal{L}$ satisfies φ under the context C, is inductively defined as follows (we again omit the semantics of the Boolean connectives):

$$\begin{array}{ll}
(\Pi, C) \models_{\mathcal{L}} p[x] & \Leftrightarrow \Pi(x) = (\sigma, i) \text{ and } p \in \sigma(i) \\
(\Pi, C) \models_{\mathcal{L}} \exists x.\, \varphi & \Leftrightarrow \text{ for some } \sigma \in \mathcal{L},\ (\Pi[x \mapsto (\sigma, 0)], C) \models_{\mathcal{L}} \varphi \\
(\Pi, C) \models_{\mathcal{L}} \langle C' \rangle \varphi & \Leftrightarrow (\Pi, C') \models_{\mathcal{L}} \varphi \\
(\Pi, C) \models_{\mathcal{L}} \mathbf{X}_\Gamma \varphi & \Leftrightarrow (\mathit{succ}_{(\Gamma,C)}(\Pi), C) \models \varphi \\
(\Pi, C) \models_{\mathcal{L}} \mathbf{Y}_\Gamma \varphi & \Leftrightarrow \mathit{pred}_{(\Gamma,C)}(\Pi) \neq \mathtt{und} \text{ and } (\mathit{pred}_{(\Gamma,C)}(\Pi), C) \models_{\mathcal{L}} \varphi \\
(\Pi, C) \models_{\mathcal{L}} \varphi_1 \,\mathbf{U}_\Gamma\, \varphi_2 & \Leftrightarrow \text{for some } i \geq 0\text{: } (\mathit{succ}^i_{(\Gamma,C)}(\Pi), C) \models_{\mathcal{L}} \varphi_2 \text{ and} \\
 & \quad (\mathit{succ}^j_{(\Gamma,C)}(\Pi), C) \models_{\mathcal{L}} \varphi_1 \text{ for all } 0 \leq j < i, \\
(\Pi, C) \models_{\mathcal{L}} \varphi_1 \,\mathbf{S}_\Gamma\, \varphi_2 & \Leftrightarrow \text{for some } i \geq 0 \text{ such that } \mathit{pred}^i_{(\Gamma,C)}(\Pi) \neq \mathtt{und}\text{: } (\mathit{pred}^i_{(\Gamma,C)}(\Pi), C) \models_{\mathcal{L}} \varphi_2 \\
 & \quad \text{and } (\mathit{pred}^j_{(\Gamma,C)}(\Pi), C) \models_{\mathcal{L}} \varphi_1 \text{ for all } 0 \leq j < i
\end{array}$$

For a sentence φ and a set of traces $\mathcal{L}$, $\mathcal{L}$ is a *model* of φ, written $\mathcal{L} \models \varphi$, if $(\Pi_\emptyset, \mathsf{VAR}) \models_{\mathcal{L}} \varphi$ where $\Pi_\emptyset$ is the trace assignment with empty domain.

Fair model checking and standard model checking. For a fragment $\mathcal{F}$ of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, the *fair model checking problem* for $\mathcal{F}$ consists on deciding, given a fair finite Kripke structure $(\mathcal{K}, F)$ and a sentence φ of $\mathcal{F}$, whether $\mathcal{L}(\mathcal{K}, F) \models \varphi$. The previous problem is simply called *model checking problem* whenever F coincides with the set of $\mathcal{K}$-states. We consider fair model checking just for technical convenience. For the decidable fragment of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ introduced in Section 3.3, we will obtain the same complexity bounds for both fair model checking and standard model checking (see Section 4).

3.3 The Simple Fragment of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$

We introduce now a fragment of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, that we call *simple* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, which syntactically subsumes the simple fragment of $\mathsf{HyperLTL}_{\mathsf{S}}$ [7].

In order to define the syntax of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, we first consider some shorthands, obtained by a restricted use of the context modalities. The *pointed existential quantifier* $\exists^{\mathsf{P}} x$ and the *pointed universal quantifier* $\forall^{\mathsf{P}} x$ are defined as follows: $\exists^{\mathsf{P}} x.\, \varphi := \exists x.\, \langle x \rangle\, \mathbf{F}\, \langle \mathsf{VAR} \rangle \varphi$ and $\forall^{\mathsf{P}} x.\, \varphi ::= \neg \exists^{\mathsf{P}} x.\, \neg\varphi$. Thus the pointed quantifiers quantify on arbitrary pointed traces over the given set of traces and set the global context for the given operand. Formally, $(\Pi, C) \models_{\mathcal{L}} \exists^{\mathsf{P}} x.\, \varphi$ if for some pointed trace (σ, i) with $\sigma \in \mathcal{L}$, $(\Pi[x \mapsto (\sigma, i)], \mathsf{VAR}) \models_{\mathcal{L}} \varphi$. For example, the sentence $\exists x_1.\, \exists^{\mathsf{P}} x_2.\, \big(\bigwedge_{p \in \mathsf{AP}} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \wedge \langle x_2 \rangle \mathbf{Y} \top\big)$ asserts that there are two traces σ_1 and σ_2 in the given model s.t. some *proper* suffix of σ_2 coincides with σ_1.

Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ is parameterized by a finite set Γ of PLTL formulas. The set of formulas φ_Γ in the Γ-fragment is defined as follows:

$$\varphi_\Gamma := \top \mid \langle x\rangle\psi[x] \mid \neg\varphi_\Gamma \mid \varphi_\Gamma \vee \varphi_\Gamma \mid \exists^{\mathsf{P}} x.\, \varphi_\Gamma \mid \mathbf{X}_\Gamma\varphi_\Gamma \mid \mathbf{Y}_\Gamma\varphi_\Gamma \mid \varphi_\Gamma \,\mathbf{U}_\Gamma\, \varphi_\Gamma \mid \varphi_\Gamma \,\mathbf{S}_\Gamma\, \varphi_\Gamma$$

where ψ is a PLTL formula. Note that $\exists x.\, \varphi$ can be expressed as $\exists^{\mathsf{P}} x.\, (\varphi \wedge \langle x\rangle\neg\mathbf{Y}\top)$. $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ is the class of formulas obtained with the syntax above for a given Γ. Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ is the union $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ for all Γ. We say that a formula φ of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ is *singleton-free* if for each subformula $\langle x\rangle\psi[x]$ of φ, ψ is an atomic proposition. Evidently, for an atomic proposition p, $\langle x\rangle p[x]$ is equivalent to $p[x]$.

In Section 4, we will show that (fair) model checking of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ is decidable. Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ can be seen as a very large fragment of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ with a decidable model checking problem which subsumes the simple fragment of $\mathsf{HyperLTL}_{\mathsf{S}}$, is closed under Boolean connectives, and allows an unrestricted nesting of temporal modalities. We conjecture (without proof) that this is the largest such sub-class of $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ because:

1. Model checking of $\mathsf{HyperLTL}_{\mathsf{S}}$ is already undecidable [7] for sentences whose relativized temporal modalities exploit two distinct sets of LTL formulas and, in particular, for two-variable quantifier alternation-free sentences of the form $\exists x_1.\, \exists x_2.\, (\varphi \wedge \mathbf{G}_\Gamma\psi)$, where ψ is a propositional formula, Γ is a nonempty set of propositions, and φ is a quantifier-free formula which use only the temporal modalities $\mathbf{F}_\emptyset$ and $\mathbf{G}_\emptyset$.
2. Model checking of $\mathsf{HyperLTL}_{\mathsf{C}}$ is undecidable [8] even for the fragment consisting of two-variable quantifier alternation-free sentences of the form $\exists x_1.\exists x_2.\, \psi_0 \wedge \langle x_2\rangle\, \mathbf{F}\, \langle\{x_1, x_2\}\rangle\psi$, where ψ_0 and ψ are quantifier-free $\mathsf{HyperLTL}$ formulas (note that ψ_0 is evaluated in the global context $\langle\{x_1, x_2\}\rangle$).

The second undecidability result suggests to consider the extension of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ where singleton-context subformulas of the form $\langle x\rangle\psi[x]$ are replaced with *quantifier-free* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ formulas with multiple variables of the form $\langle x\rangle\xi$, where ξ only uses singleton contexts $\langle y\rangle$ and temporal modalities with subscript $\emptyset$. However, we can show that the resulting logic is not more expressive than simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$: a sentence in the considered extension can be translated into an equivalent simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ sentence though with a non-elementary blowup (for details, see [4]).

3.4 Examples of Specifications in Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$

We consider some relevant properties which can be expressed in simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$. Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ subsumes the simple fragment of $\mathsf{HyperLTL}_{\mathsf{S}}$, and this fragment (as shown in [7]) can express relevant information-flow security properties for asynchronous frameworks such as distributed systems or cryptographic protocols. An example is the asynchronous variant of the *noninterference* property, as defined by Goguen and Meseguer [18], which asserts that the observations of low users (users accessing only to public information) do not change when all inputs of high users (users accessing secret information) are removed.

Observational Determinism

An important information-flow property is observational determinism, which states that traces which have the same initial low inputs are indistinguishable to a low user. In an asynchronous setting, a user cannot infer that a transition occurred if consecutive observations remain unchanged. Thus, for instance, observational determinism with equivalence of traces up to stuttering (as formulated in [36]) can be captured by the following simple $\mathsf{HyperLTL_S}$ sentence (where LI is the set of propositions describing inputs of low users and LO is set of propositions describing outputs of low users):

$$\forall x. \forall y. \bigwedge_{p \in LI} (p[x] \leftrightarrow p[y]) \to \mathbf{G}_{LO} \bigwedge_{p \in LO} (p[x] \leftrightarrow p[y])$$

After-initialization Properties

Simple $\mathsf{GHyperLTL_{S+C}}$ also allows to specify complex combinations of asynchronous and synchronous constraints. As an example, we consider the property [21] that for an $\mathsf{HyperLTL}$ sentence $\mathsf{Q}_1 x_1. \ldots \mathsf{Q}_n x_n. \psi(x_1, \ldots, x_n)$, the quantifier-free formula $\psi(x_1, \ldots, x_n)$ holds along the traces bound by variables $x_1, \ldots, x_n$ after an initialization phase. Note that this phase can take a different (and unbounded) number of steps on each trace. Let in be a proposition characterizing the initialization phase. The formula $PI(x) := \langle x \rangle(\neg in[x] \wedge (\neg \mathbf{Y} \top \vee \mathbf{Y} in[x]))$ is a simple $\mathsf{GHyperLTL_{S+C}}$ formula that asserts that for the pointed trace (σ, i) assigned to variable x, the position i is the first position of σ following the initialization phase. In other words, i is the first position at which $\neg in[]$ holds. Then, the previous requirement can be expressed in simple $\mathsf{GHyperLTL_{S+C}}$ as follows:

$$\mathsf{Q}_1 x_1 \ldots . \mathsf{Q}_n x_n . \big(PI(x_1) \circ_1 \ldots PI(x_n) \circ_n \psi(x_1, \ldots x_n)$$

where $\circ_i$ is $\wedge$ if $\mathsf{Q}_i = \exists$ and $\circ_i$ is $\to$ if $\mathsf{Q}_i = \forall$.

Global Promptness

As another meaningful example, we consider global promptness (in the style of Prompt LTL [25]), where one need to check that there is a uniform bound on the response time in all the traces of the system, that is, "*there is k such that for every trace, each request q is followed by a response p within k steps*". Global promptness is expressible in simple $\mathsf{GHyperLTL_{S+C}}$ as follows:

$$\exists^{\mathsf{P}} x. \big(q[x] \;\wedge\; \forall^{\mathsf{P}} y. \left(q[y] \to (\neg p[x] \;\mathbf{U}\; p[y])\right)\big)$$

The previous sentence asserts that there is request (x in the formula) that has the longest response. Note that y is quantified universally (so it can be instantiated to the same trace as x), and that the use of until in $(\neg p[x] \;\mathbf{U}\; p[y]))\big)$ implies that the response $p[y]$ eventually happens. Hence, all requests, including receive a response. Now, the pointed trace (σ_x, i) assigned to x is such that $\sigma_x(i)$ is a request ($q[x]$) and for every pointed trace (σ_y, j) in the model such that $\sigma_y(j)$ is a request ($q[y]$), it holds that (i) the request $\sigma_y(j)$ is followed by a response $\sigma_y(j + k)$ for some $k \geq 0$, and (ii) no response occurs in σ_x in the interval $[i, i + k)$. Therefore, the response time h for x is the smallest h such that $\sigma_x(i + h)$ is a response is a global bound on the response time.

Diagnosability

We now show that simple $\mathsf{GHyperLTL_{S+C}}$ is also able to express *diagnosability* of systems [31, 5, 3] in a general asynchronous setting. In the diagnosis process, faults of a critical system (referred as the plant) are handled by a dedicated module (the *diagnoser*) which runs in parallel with the plant. The diagnoser analyzes the observable information from the plant – made available by predefined sensors – and triggers suitable alarms in correspondence to (typically unobservable) conditions of interest, called faults. An alarm condition specifies the relation (delay) between a given diagnosis condition and the raising of an alarm. A plant $\mathcal{P}$ is *diagnosable* with respect to a given alarm condition α, if there is a diagnoser $\mathcal{D}$ which satisfies α when $\mathcal{D}$ runs in parallel with $\mathcal{P}$.

The given set of propositions AP is partitioned into a set of observable propositions Obs and a set of unobservable propositions Int. Two finite traces σ and σ' are *observationally equivalent* iff the projections of $\mathit{stfr}_{\mathsf{Obs}}(\sigma \cdot P^{\omega})$ and $\mathit{stfr}_{\mathsf{Obs}}(\sigma' \cdot (P')^{\omega})$ over Obs coincide, where P is the last symbol of σ and similarly P' is the last symbol of σ'. Given a pointed trace (σ, i), i is an *observation point* of σ if either $i = 0$, or $i > 0$ and $\sigma(i-1) \cap \mathsf{Obs} \neq \sigma(i) \cap \mathsf{Obs}$. Then a plant $\mathcal{P}$ can be modeled as a finite Kripke structure $\langle S, S_0, E, \mathit{Lab}\rangle$, where E is partitioned into internal transitions (s, s') where $\mathit{Lab}(s) \cap \mathsf{Obs} = \mathit{Lab}(s') \cap \mathsf{Obs}$ and observable transitions where $\mathit{Lab}(s) \cap \mathsf{Obs} \neq \mathit{Lab}(s') \cap \mathsf{Obs}$. A diagnoser $\mathcal{D}$ is modelled as a finite deterministic Kripke structure over $\mathsf{AP}' \supseteq \mathsf{Obs}$ (with $\mathsf{AP}' \cap \mathsf{Int} = \emptyset$). In the behavioural composition of the plant $\mathcal{P}$ with $\mathcal{D}$, the diagnoser only reacts to the observable transitions of the plant, that is, every transition of the diagnoser is associated with an observable transition of the plant. Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ can express diagnosability with *finite delay*, *bounded delay*, or *exact delay* as defined in [5, 3]. Here, we focus for simplicity on finite delay diagnosability. Consider a diagnosis condition specified by a PLTL formula β. A plant $\mathcal{P}$ is *finite delay diagnosable* with respect to β whenever for every pointed trace (σ, i) of $\mathcal{P}$ such that $(\sigma, i) \models \beta$, there exists an observation point $k \geq i$ of σ such that for all pointed traces (σ', k') of $\mathcal{P}$ so that k' is an observation point of σ' and $\sigma[0, k]$ and $\sigma'[0, k']$ are observationally equivalent, it holds that $(\sigma', i') \models \beta$ for some $i' \leq k'$. Finite delay diagnosability w.r.t. β can be expressed in simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ as follows:

$$\forall^{\mathsf{P}} x. \Big(\langle x\rangle \beta[x] \to \mathbf{F}_{\mathsf{Obs}}\big(\mathsf{ObsPt}(x) \wedge \forall^{\mathsf{P}} y. \{(\mathsf{ObsPt}(y) \wedge \theta_{\mathsf{Obs}}(x,y)) \to \langle y\rangle\, \mathbf{O}\, \beta[y]\}\big)\Big)$$

where

$$\theta_{\mathsf{Obs}}(x,y) \;:=\; \bigwedge_{p \in \mathsf{Obs}} \mathbf{H}_{\mathsf{Obs}}\,(p[x] \leftrightarrow p[y]) \;\wedge\; \mathbf{O}_{\mathsf{Obs}}(\langle x\rangle \neg \mathbf{Y}\top \wedge \langle y\rangle \neg \mathbf{Y}\top)$$

$$\mathsf{ObsPt}(x) \;:=\; \langle x\rangle(\neg \mathbf{Y}\top \wedge \bigvee_{p \in \mathsf{Obs}} (p[x] \leftrightarrow \neg \mathbf{Y} p[x])$$

Essentially $\mathsf{ObsPt}(x)$ determines the observation points and θ_{Obs} captures that both traces have the same history of observations. The main formula establishes that if x detects a failure β then there is future observation point in x and for all other traces that are observationally equivalent to x have also detected β.

3.5 Expressiveness Issues

In this section, we present some results and conjectures about the expressiveness comparison among $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ (which subsumes $\mathsf{HyperLTL}_{\mathsf{S}}$ and $\mathsf{HyperLTL}_{\mathsf{C}}$), its simple fragment and the logic $\mathsf{HyperLTL}_{\mathsf{S}}$. We also consider the logics for linear-time hyperproperties based on the equal-level predicate whose most powerful representative is $\mathsf{S1S[E]}$. Recall that the first-order fragment $\mathsf{FO[<,E]}$ of $\mathsf{S1S[E]}$ is already strictly more expressive than $\mathsf{HyperLTL}$ [16] and, unlike $\mathsf{S1S[E]}$, its model-checking problem is decidable [11]. Moreover, we show that $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ and its simple fragment represent a unifying framework in the linear-time setting for specifying both hyperproperties and the knowledge modalities of epistemic temporal logics under both the synchronous and asynchronous perfect recall semantics.

Our expressiveness results about linear-time hyper logics can be summarized as follows.

▶ **Theorem 3.2.** *The following hold:*

- $\mathsf{GHyperLTL_{S+C}}$ *is more expressive than* $\mathsf{HyperLTL_S}$*, simple* $\mathsf{GHyperLTL_{S+C}}$*, and* $\mathsf{FO[<,E]}$.
- *Simple* $\mathsf{GHyperLTL_{S+C}}$ *is more expressive than simple* $\mathsf{HyperLTL_S}$.
- *Simple* $\mathsf{GHyperLTL_{S+C}}$ *and* $\mathsf{HyperLTL_S}$ *are expressively incomparable.*
- *Simple* $\mathsf{GHyperLTL_{S+C}}$ *and* $\mathsf{S1S[E]}$ *are expressively incomparable.*

Proof. We first show that there are hyperproperties expressible in simple $\mathsf{GHyperLTL_{S+C}}$ but not in $\mathsf{HyperLTL_S}$ and in $\mathsf{S1S[E]}$. Given a sentence φ, the *trace property denoted by* φ is the set of traces σ such that the singleton set of traces $\{\sigma\}$ satisfies φ. It is known that $\mathsf{HyperLTL_S}$ and $\mathsf{S1S[E]}$ capture only *regular* trace properties [8]. In contrast simple $\mathsf{GHyperLTL_{S+C}}$ can express powerful non-regular trace properties. For example, consider the so called *suffix property* over $\mathsf{AP} = \{p\}$: a trace σ satisfies the suffix property if there exists a proper suffix σ^k of σ for some $k > 0$ such that $\sigma^k = \sigma$. This non-regular trace property can be expressed in $\mathsf{SHyperLTL^{\emptyset}_{S+C}}$ as follows:

$$\forall x_1.\,\forall x_2.\; \bigwedge_{p\in\mathsf{AP}} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \wedge \forall x_1.\,\exists^{\mathsf{P}} x_2.\, \Big(\bigwedge_{p\in\mathsf{AP}} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \wedge \langle x_2\rangle \mathbf{Y}\top \Big)$$

The first conjunct asserts that each model is a singleton, and the second conjunct requires that for the unique trace σ in a model, there is $k > 0$ such that $\sigma(i) = \sigma(i + k)$ for all $i \geq 0$.

Next, we observe that $\mathsf{FO[<,E]}$ can be easily translated into $\mathsf{GHyperLTL_{S+C}}$, since the pointer quantifiers of $\mathsf{GHyperLTL_{S+C}}$ correspond to the quantifiers of $\mathsf{FO[<,E]}$. Moreover, the predicate $x \leq x'$ of $\mathsf{FO[<,E]}$, expressing that for the pointed traces (σ, i) and (σ', i') bound to x and x', $\sigma = \sigma'$ and $i \leq i'$, can be easily captured in $\mathsf{GHyperLTL_{S+C}}$. This is also the case for the equal-level predicate $\mathsf{E}(x, x')$, which can be expressed as $\langle\{x, x'\}\rangle\, \mathbf{O}\, (\langle x\rangle\neg\mathbf{Y}\top \wedge \langle x'\rangle\neg\mathbf{Y}\top)$.

In Section 4 we show that model checking of simple $\mathsf{GHyperLTL_{S+C}}$ is decidable. Thus, since model checking of both $\mathsf{HyperLTL_S}$ and $\mathsf{S1S[E]}$ are undecidable [7, 11] and $\mathsf{GHyperLTL_{S+C}}$ subsumes $\mathsf{HyperLTL_S}$, by the previous argumentation, the theorem follows. ◀

It remains an open question whether $\mathsf{FO[<,E]}$ is subsumed by simple $\mathsf{GHyperLTL_{S+C}}$. We conjecture that neither $\mathsf{HyperLTL_C}$ nor the fix-point calculus H_μ [21] (which captures both $\mathsf{HyperLTL_C}$ and $\mathsf{HyperLTL_S}$ [8]) subsume simple $\mathsf{GHyperLTL_{S+C}}$. The motivation for our conjecture is that H_μ sentences consist of a prefix of quantifiers followed by a quantifier-free formula where quantifiers range over *initial* pointed traces $(\sigma, 0)$. Thus, unlike simple $\mathsf{GHyperLTL_{S+C}}$, H_μ cannot express requirements which relate at some point in time an unbounded number of traces. Diagnosability (see Subsection 3.4) falls in this class of requirements. It is known that the following property, which can be easily expressed in simple $\mathsf{GHyperLTL_{S+C}}$, is not definable in $\mathsf{HyperLTL}$ [6]: for some $i > 0$, every trace in the given set of traces does not satisfy proposition p at position i. We conjecture that similarly to $\mathsf{HyperLTL}$, such a property (and diagnosability as well) cannot be expressed in H_μ.

Epistemic Temporal Logic KLTL and its relation with $\mathsf{GHyperLTL_{S+C}}$. The logic KLTL [23] is a well-known extension of LTL obtained by adding the unary knowledge modalities $\mathbf{K}_a$, where a ranges over a finite set Agts of agents, interpreted under the synchronous or asynchronous (perfect recall) semantics. The semantics is given with respect to an observation map $\mathsf{Obs} : \mathsf{Agts} \mapsto 2^{\mathsf{AP}}$ that assigns to each agent a the set of propositions which agent a can observe. Given two finite traces σ and σ' and $a \in \mathsf{Agts}$, σ and σ' are *synchronously equivalent for agent* a, written $\sigma \sim^{sy}_a \sigma'$, if the projections of σ and σ' over $\mathsf{Obs}(a)$ coincide. The finite traces σ and σ' are *asynchronously equivalent for agent* a, written $\sigma \sim^{as}_a \sigma'$, if the projections of $stfr_{\mathsf{Obs}(a)}(\sigma \cdot P^\omega)$ and $stfr_{\mathsf{Obs}(a)}(\sigma' \cdot (P')^\omega)$ over $\mathsf{Obs}(a)$ coincide, where

P is the last symbol of σ and P' is the last symbol of σ'. For a set of traces $\mathcal{L}$ and a pointed trace (σ, i) over $\mathcal{L}$, the semantics of the knowledge modalities is as follows, where $\sim_a$ is $\sim_a^{sy}$ under the synchronous semantics, and $\sim_a^{as}$ otherwise: $(\sigma, i) \models_{\mathcal{L},\mathsf{Obs}} \mathbf{K}_a \varphi \;\Leftrightarrow$ for all pointed traces (σ', i') on $\mathcal{L}$ such that $\sigma[0, i] \sim_a \sigma'[0, i']$, $(\sigma', i') \models_{\mathcal{L},\mathsf{Obs}} \varphi$.

We say that $\mathcal{L}$ *satisfies* φ *w.r.t. the observation map* Obs, written $\mathcal{L} \models_{\mathsf{Obs}} \varphi$, if for all traces $\sigma \in \mathcal{L}$, $(\sigma, 0) \models_{\mathcal{L},\mathsf{Obs}} \varphi$. The logic KLTL can be easily embedded into $\mathsf{GHyperLTL}_{\mathsf{S+C}}$. In particular, the following holds (for details, see [4]).

▶ **Theorem 3.3.** *Given an observation map Obs and a* KLTL *formula* ψ *over AP, one can construct in linear time a* $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ *sentence* $\varphi_\emptyset$ *and a* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ *sentence* φ *such that* $\varphi_\emptyset$ *is equivalent to* ψ *w.r.t. Obs under the synchronous semantics and* φ *is equivalent to* ψ *w.r.t. Obs under the asynchronous semantics. Moreover,* φ *is a simple* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ *sentence if* ψ *is in the single-agent fragment of* KLTL.

4 Decidability of Model Checking against Simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$

In this section, we show that (fair) model checking against simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ is decidable. We first prove the result for the fragment $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ by a linear-time reduction to satisfiability of *full* Quantified Propositional Temporal Logic (QPTL, for short) [32], where the latter extends PLTL by quantification over propositions. Then, we show that (fair) model checking of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ can be reduced in time singly exponential in the size of the formula to fair model checking of $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. We also provide optimal complexity bounds for (fair) model checking the fragment $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ in terms of a parameter of the given formula called *strong alternation depth*. For this, we first give similar optimal complexity bounds for satisfiability of QPTL.
The syntax of QPTL formulas φ over a finite set AP of atomic propositions is as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \mathbf{Y}\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \varphi\,\mathbf{S}\,\varphi \mid \exists p\,.\varphi$$

where $p \in \mathsf{AP}$ and $\exists p$ is the propositional existential quantifier. A QPTL formula φ is a *sentence* if each proposition p occurs in the scope of a quantifier binding p and each temporal modality occurs in the scope of a quantifier. By introducing $\wedge$ and the operators $\mathbf{R}$ (*release*, dual of $\mathbf{U}$), $\mathbf{P}$ (*past release*, dual of $\mathbf{S}$) and $\forall p$ (propositional universal quantifier), every QPTL formula can be converted into an equivalent formula in *negation normal form*, where negation only appears in front of atomic propositions. QPTL formulas are interpreted over pointed traces (σ, i) over AP. All QPTL temporal operators have the same semantics as in PLTL. The semantics of propositional quantification is as follows:

$$(\sigma, i) \models \exists p.\varphi \;\Leftrightarrow\; \text{there is a trace } \sigma' \text{ such that } \sigma =_{\mathsf{AP}\setminus\{p\}} \sigma' \text{ and } (\sigma', i) \models \varphi$$

where $\sigma =_{\mathsf{AP}\setminus\{p\}} \sigma'$ means that the projections of σ and σ' over $\mathsf{AP} \setminus \{p\}$ coincide. A formula φ is satisfiable if $(\sigma, 0) \models \varphi$ for some trace σ. We now give a generalization of the standard notion of alternation depth between existential and universal quantifiers which corresponds to the one given in [30] for HyperCTL*. Our notion takes into account also the occurrences of temporal modalities between quantifier occurrences, but the nesting depth of temporal modalities is not considered (intuitively, it is collapsed to one). Formally, the *strong alternation depth* $\mathit{sad}(\varphi)$ of a QPTL formula φ in negation normal form is inductively defined as follows, where an existential formula is a formula of the form $\exists p.\,\psi$, a universal formula is of the form $\forall p.\,\psi$, and for a formula ψ, $\widetilde{\psi}$ denotes the negation normal form of $\neg\psi$:

- For $\varphi = p$ and $\varphi = \neg p$ for a given $p \in \mathsf{AP}$: $\mathit{sad}(\varphi) := 0$.
- For $\varphi = \varphi_1 \vee \varphi_2$ and for $\varphi = \varphi_1 \wedge \varphi_2$: $\mathit{sad}(\varphi) := \max(\{\mathit{sad}(\varphi_1), \mathit{sad}(\varphi_2)\})$.
- For $\varphi = \exists p.\, \varphi_1$: if there is no universal sub-formula $\forall\psi$ of φ_1 such that $\mathit{sad}(\forall\psi) = \mathit{sad}(\varphi_1)$, then $\mathit{sad}(\varphi) := \mathit{sad}(\varphi_1)$. Otherwise, $\mathit{sad}(\varphi) := \mathit{sad}(\varphi_1) + 1$.
- For $\varphi = \forall p.\, \varphi_1$: $\mathit{sad}(\varphi) := \mathit{sad}(\exists p.\, \widetilde{\varphi_1})$.
- For $\varphi = \mathbf{X}\varphi_1$ or $\varphi = \mathbf{Y}\varphi_1$: $\mathit{sad}(\varphi) := \mathit{sad}(\varphi_1)$.
- For $\varphi = \varphi_1 \,\mathbf{U}\, \varphi_2$ or $\varphi = \varphi_1 \,\mathbf{S}\, \varphi_2$: let h be the maximum over the strong alternation depths of the universal and existential sub-formulas of φ_1 and φ_2 (the maximum of the empty set is 0). If the following conditions are met, then $\mathit{sad}(\varphi) := h$; otherwise, $\mathit{sad}(\varphi) := h + 1$:
  - there is no existential or universal sub-formula ψ of φ_1 with $\mathit{sad}(\psi) = h$;
  - there is no universal sub-formula ψ of φ_2 with $\mathit{sad}(\psi) = h$;
  - no existential formula ψ with $\mathit{sad}(\psi) = h$ occurs in the left operand (resp., right operand) of a sub-formula of φ_2 of the form $\psi_1 \mathcal{O} \psi_2$, where $\mathcal{O} \in \{\mathbf{U}, \mathbf{S}\}$ (resp., $\mathcal{O} \in \{\mathbf{R}, \mathbf{P}\}$).
- Finally, for $\varphi = \varphi_1 \,\mathbf{R}\, \varphi_2$ or $\varphi = \varphi_1 \,\mathbf{P}\, \varphi_2$: $\mathit{sad}(\varphi) := \mathit{sad}(\widetilde{\varphi})$.

For example, $\mathit{sad}(\exists p.(p \,\mathbf{U}\, \exists q.q)) = 0$ and $\mathit{sad}(\exists p.(\exists p.p \,\mathbf{U}\, q)) = 1$. The strong alternation depth of an arbitrary QPTL formula corresponds to the one of its negation normal form. The strong alternation depth of a simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ formula is defined similarly but we replace quantification over propositions with quantification over trace variables. For all $n, h \in \mathbb{N}$, $\mathsf{Tower}(h, n)$ denotes a tower of exponentials of height h and argument n: $\mathsf{Tower}(0, n) = n$ and $\mathsf{Tower}(h + 1, n) = 2^{\mathsf{Tower}(h,n)}$. Essnetially, the strong alternation depth corresponds to the (unavoidable) power set construction related to the alternation of quantifiers to solve the model-checking problem.

The following result represents an improved version of Theorem 6 in [6] where h-EXPSPACE is the class of languages decided by deterministic Turing machines bounded in space by functions of n in $O(\mathsf{Tower}(h, n^c))$ for some constant $c \geq 1$. While the lower bound directly follows from [32], the upper bound improves the result in [6], since there, occurrences of temporal modalities immediately preceding propositional quantification always count as additional alternations (for details, see [4]).

▶ **Theorem 4.1.** *For all $h \geq 0$, satisfiability of* QPTL *sentences with strong alternation depth at most h is h-*EXPSPACE*-complete.*

(Fair) Model checking against $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. We provide now linear-time reductions of (fair) model checking against $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ to (and from) satisfiability of QPTL which preserve the strong alternation depth. We start with the reduction of (fair) model checking $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ to QPTL satisfiability.

▶ **Theorem 4.2.** *Given a fair finite Kripke structure $(\mathcal{K}, F)$ and a $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ sentence φ, one can construct in linear time a* QPTL *sentence ψ with the same strong alternation depth as φ such that ψ is satisfiable if and only if $\mathcal{L}(\mathcal{K}, F) \models \varphi$.*

Sketched proof. Let $\mathcal{K} = \langle S, S_0, E, \mathit{Lab}\rangle$. In the reduction of model checking $(\mathcal{K}, F)$ against φ to QPTL satisfiability, we need to merge multiple traces into a unique trace where just one position is considered at any time. An issue is that the hyper quantifiers range over arbitrary pointed traces so that the positions of the different pointed traces in the current trace assignment do not necessarily coincide (intuitively, the different pointed traces are not aligned with respect to the relative current positions). However, we can solve this issue because the offsets between the positions of the pointed traces in the current trace assignment remain constant during the evaluation of the temporal modalities. In particular, assume that

(σ, i) is the first pointed trace selected by a hyper quantifier during the evaluation along a path in the syntax tree of φ. We encode σ by keeping track also of the variable x to which (σ, i) is bound and the F-fair path of $\mathcal{K}$ whose associated trace is σ. Let (σ', i') be another pointed trace introduced by another hyper quantifier y during the evaluation of φ. If $i' < i$, we consider an encoding of σ' which is similar to the previous encoding but we precede it with a *padding prefix* of length $i - i'$ of the form $\{\#_{\overrightarrow{y}}\}^{i-i'}$. The arrow → indicates that the encoding is along the *forward direction*. Now, assume that $i' > i$. In this case, the encoding of σ' is the merging of two encodings over disjoint sets of propositions: one along the forward direction which encodes the suffix $(\sigma')^{i'-i}$ and another one along the *backward direction* which is of the form $\{\#_{\overleftarrow{y}}\} \cdot \rho \cdot \{\#_{\overleftarrow{y}}\}^{\omega}$, where ρ is a backward encoding of the *reverse* of the prefix of σ' of length $i' - i$. In such a way, the encodings of the pointed traces later introduced in the evaluation of φ are aligned with the reference pointed trace (σ, i). Since the positions in the backward direction overlap some positions in the forward direction, in the translation, we keep track of whether the current position refers to the forward or to the backward direction. The details of the reduction are in [4]. ◀

By an adaptation of the known reduction of satisfiability of QPTL without past to model checking of HyperCTL* [9], we obtain the following result (for details, see [4]).

▶ **Theorem 4.3.** *Given a* QPTL *sentence ψ over* AP*, one can build in linear time a finite Kripke structure $\mathcal{K}_{\mathsf{AP}}$ (depending only on* AP*) and a singleton-free* $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ *sentence φ having the same strong alternation depth as ψ such that ψ is satisfiable* iff $\mathcal{L}(\mathcal{K}_{\mathsf{AP}}) \models \varphi$.

Hence, by Theorems 4.1–4.3, we obtain the following result.

▶ **Corollary 4.4.** *For all $h \geq 0$, (fair) model checking against* $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ *sentences with strong alternation depth at most h is h-*EXPSPACE*-complete.*

Reduction to fair model checking against $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. We solve the (fair) model checking problem for simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ by a reduction to fair model checking against the fragment $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. Our reduction is exponential in the size of the given sentence and is an adaptation of the reduction from model checking simple $\mathsf{HyperLTL}_{\mathsf{S}}$ to model checking HyperLTL shown in [7]. As a preliminary step, we first show, by an easy adaptation of the standard automata-theoretic approach for PLTL [34], that the problem for a simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ sentence φ can be reduced in exponential time to the fair model checking problem against a singleton-free sentence in the fragment $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ for some set Γ of *atomic propositions* depending on φ. For details, see [4].

▶ **Proposition 4.5.** *Given a simple* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ *sentence φ and a fair finite Kripke structure $(\mathcal{K}, F)$ over* AP*, one can build in single exponential time in the size of φ, a fair finite Kripke structure $(\mathcal{K}', F')$ over an extension* AP′ *of* AP *and a singleton-free* $\mathsf{SHyperLTL}^{\Gamma'}_{\mathsf{S+C}}$ *sentence φ' for some $\Gamma' \subseteq \mathsf{AP}'$ such that $\mathcal{L}(\mathcal{K}', F') \models \varphi'$ if and only if $\mathcal{L}(\mathcal{K}, F) \models \varphi$. Moreover, φ' has the same strong alternation depth as φ, $|\varphi'| = O(|\varphi|)$, and $|\mathcal{K}'| = O(|\mathcal{K}| * 2^{O(|\varphi|)})$.*

Let us fix a non-empty finite set $\Gamma \subseteq \mathsf{AP}$ of atomic propositions. We now show that fair model checking of the singleton-free fragment of $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ can be reduced in polynomial time to fair model checking of $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. We observe that in the singleton-free fragment of $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$, when a pointed trace (σ, i) is selected by a pointed quantifier $\exists^{\mathsf{P}} x$, the positions of σ which are visited during the evaluation of the temporal modalities are all in the (Γ, ω)-stutter factorization of σ with the possible exception of the position i chosen by $\exists^{\mathsf{P}} x$. Thus, given a set $\mathcal{L}$ of traces and a special proposition $\# \notin \mathsf{AP}$, we define an extension

$\mathit{stfr}^{\#}_{\Gamma}(\mathcal{L})$ of the set $\mathit{stfr}_{\Gamma}(\mathcal{L}) = \{\mathit{stfr}_{\Gamma}(\sigma) \mid \sigma \in \mathcal{L}\}$ as follows. Intuitively, we consider for each trace $\sigma \in \mathcal{L}$, its Γ-stutter trace $\mathit{stfr}_{\Gamma}(\sigma)$ and the extensions of $\mathit{stfr}_{\Gamma}(\sigma)$ which are obtained by adding an extra position marked by proposition $\#$ (this extra position does not belong to the (Γ, ω)-stutter factorization of σ). Formally, $\mathit{stfr}^{\#}_{\Gamma}(\mathcal{L})$ is the smallest set containing $\mathit{stfr}_{\Gamma}(\mathcal{L})$ and satisfying the following condition:

- for each trace $\sigma \in \mathcal{L}$ with (Γ, ω)-stutter factorization $\{\ell_k\}_{k\geq 0}$ and position $i \in (\ell_k, \ell_{k+1})$ for some $k \geq 0$, the trace $\sigma(\ell_0)\ldots\sigma(\ell_k)\,(\sigma(i) \cup \{\#\})\,\sigma(\ell_{k+1})\,\sigma(\ell_{k+2})\ldots \in \mathit{stfr}^{\#}_{\Gamma}(\mathcal{L})$.

Given a singleton-free formula φ in $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$, we denote by $\mathsf{T}_{\#}(\varphi)$ the formula in $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ obtained from φ by applying inductively the following transformations:

- the Γ-relativized temporal modalities are replaced with their $\emptyset$-relativized counterparts;
- each formula $\exists^{\mathsf{P}} x.\,\phi$ is replaced with $\exists^{\mathsf{P}} x.\,\big(\mathsf{T}_{\#}(\phi) \wedge \langle x\rangle(\mathbf{X}\,\mathbf{G}\,\neg\#[x] \wedge (\mathbf{Y}\top \rightarrow \mathbf{Y}\mathbf{H}\neg\#[x]))\big)$.

Intuitively, the formula $\mathsf{T}_{\#}(\exists^{\mathsf{P}} x.\,\phi)$ states that for the pointed trace (σ, i) selected by the pointed quantifier, at most position i may be marked by the special proposition $\#$. By the semantics of the logics considered, the following holds.

▶ Remark 4.6. Given a singleton-free sentence φ of $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$ and a set $\mathcal{L}$ of traces, it holds that $\mathcal{L} \models \varphi$ if and only if $\mathit{stfr}^{\#}_{\Gamma}(\mathcal{L}) \models \mathsf{T}_{\#}(\varphi)$.

Let us fix now a fair finite Kripke structure $(\mathcal{K}, F)$. We first show that one can build in polynomial time a finite Kripke structure $(\mathcal{K}_{\Gamma}, F_{\Gamma})$ and a LTL formula θ_{Γ} such that $\mathit{stfr}^{\#}_{\Gamma}(\mathcal{L}(\mathcal{K}, F))$ coincides with the traces of $\mathcal{L}(\mathcal{K}_{\Gamma}, F_{\Gamma})$ satisfying θ_{Γ} (details are in [4]).

▶ **Proposition 4.7.** *Given $\emptyset \neq \Gamma \subseteq \mathit{AP}$ and a fair finite Kripke structure $(\mathcal{K}, F)$ over AP, one can construct in polynomial time a fair finite Kripke structure $(\mathcal{K}_{\Gamma}, F_{\Gamma})$ and a* LTL *formula θ_{Γ} such that $\mathit{stfr}^{\#}_{\Gamma}(\mathcal{L}(\mathcal{K}, F))$ is the set of traces $\sigma \in \mathcal{L}(\mathcal{K}_{\Gamma}, F_{\Gamma})$ so that $\sigma \models \theta_{\Gamma}$.*

Fix now a singleton-free sentence φ of $\mathsf{SHyperLTL}^{\Gamma}_{\mathsf{S+C}}$. For the given fair finite Kripke structure $(\mathcal{K}, F)$ over AP, let $(\mathcal{K}_{\Gamma}, F_{\Gamma})$ and θ_{Γ} as in the statement of Proposition 4.7. We consider the $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ sentence $\mathsf{T}(\varphi)$ obtained from $\mathsf{T}_{\#}(\varphi)$ by inductively replacing each subformula $\exists^{\mathsf{P}} x.\,\phi$ of $\mathsf{T}_{\#}(\varphi)$ with $\exists^{\mathsf{P}} x.\,(\mathsf{T}(\phi) \wedge \langle x\rangle\;\mathbf{O}\;(\neg\mathbf{Y}\top \wedge \theta_{\Gamma}[x]))$. In other terms, we ensure that in $\mathsf{T}_{\#}(\varphi)$ the hyper quantification ranges over traces which satisfy the LTL formula θ_{Γ}. By Remark 4.6 and Proposition 4.7, we obtain that $\mathcal{L}(\mathcal{K}, F) \models \varphi$ if and only if $\mathcal{L}(\mathcal{K}_{\Gamma}, F_{\Gamma}) \models \mathsf{T}(\varphi)$. Thus, together with Proposition 4.5, we obtain the following result.

▶ **Theorem 4.8.** *The (fair) model checking problem against simple* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ *can be reduced in singly exponential time to fair model checking against* $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$.

5 Conclusion

We have introduced a novel hyper logic $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ which merges two known asynchronous temporal logics for hyperproperties, namely stuttering HyperLTL and context HyperLTL. Even though model checking of the resulting logic $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ is undecidable, we have identified a useful fragment, called *simple* $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, that has a decidable model checking, is strictly more expressive than HyperLTL and than previously proposed fragments of asynchronous temporal logics for hyperproperties with a decidable model checking. For the fragment $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$ of simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, we have given optimal complexity bounds of (fair) model checking in terms of the strong alternation depth of the given sentence. For arbitrary sentences in simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$, (fair) model checking is reduced in exponential time to fair model checking of $\mathsf{SHyperLTL}^{\emptyset}_{\mathsf{S+C}}$. It is worth noting that simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ can express non-regular trace properties over singleton sets of traces which are not definable in S1S[E]. An intriguing open question is whether FO[<,E] can be embedded in simple

$\mathsf{GHyperLTL}_{\mathsf{S+C}}$. In a companion paper, we study asynchronous properties on finite traces by adapting simple $\mathsf{GHyperLTL}_{\mathsf{S+C}}$ in prenex form to finite traces, and introduce practical model-checking algorithms for useful fragments of this logic.

References

1 Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A Temporal Logic for Asynchronous Hyperproperties. In *Proc. 33rd CAV*, volume 12759 of *LNCS 12759*, pages 694–717. Springer, 2021. doi:10.1007/978-3-030-81685-8_33.

2 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Niklas Metzger. Second-Order Hyperproperties. In *Proc. 35th CAV*, volume 13965 of *Lecture Notes in Computer Science*, pages 309–332. Springer, 2023. doi:10.1007/978-3-031-37703-7_15.

3 Benjamin Bittner, Marco Bozzano, Alessandro Cimatti, Marco Gario, Stefano Tonetta, and Viktoria Vozárová. Diagnosability of fair transition systems. *Artif. Intell.*, 309:103725, 2022. doi:10.1016/J.ARTINT.2022.103725.

4 Alberto Bombardelli, Laura Bozzelli, César Sánchez, and Stefano Tonetta. Unifying asynchronous logics for hyperproperties, 2024. doi:10.48550/arXiv.2404.16778.

5 Marco Bozzano, Alessandro Cimatti, Marco Gario, and Stefano Tonetta. Formal Design of Asynchronous Fault Detection and Identification Components using Temporal Epistemic Logic. *Log. Methods Comput. Sci.*, 11(4), 2015. doi:10.2168/LMCS-11(4:4)2015.

6 Laura Bozzelli, Bastien Maubert, and Spophie Pinchinat. Unifying Hyper and Epistemic Temporal Logics. In *Proc. 18th FoSSaCS*, LNCS 9034, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

7 Laura Bozzelli, Adriano Peron, and César Sánchez. Asynchronous Extensions of HyperLTL. In *Proc. 36th LICS*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470583.

8 Laura Bozzelli, Adriano Peron, and César Sánchez. Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties. In *Proc. 33rd CONCUR*, volume 243 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CONCUR.2022.27.

9 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal Logics for Hyperproperties. In *Proc. 3rd POST*, LNCS 8414, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

10 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

11 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *Proc. 34th LICS*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785713.

12 Rayna Dimitrova, Bernd Finkbeiner, Máté M Kovács, Markus N. Rabe, and Helmut Seidl. Model Checking Information Flow in Reactive Systems. In *Proc. 13th VMCAI*, LNCS 7148, pages 169–185. Springer, 2012. doi:10.1007/978-3-642-27940-9_12.

13 E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.

14 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about knowledge*, volume 4. MIT Press Cambridge, 1995. doi:10.7551/mitpress/5803.001.0001.

15 Bernd Finkbeiner and Christopher Hahn. Deciding Hyperproperties. In *Proc. 27th CONCUR*, LIPIcs 59, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.13.

16 Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In *Proc. 34th STACS*, LIPIcs 66, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.30.

17 Michael J. Fischer and Richard E. Ladner. Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

18 Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.

19 Jens Oliver Gutsfeld, Arne Meier, Christoph Ohrem, and Jonni Virtema. Temporal Team Semantics Revisited. In *Proc. 37th LICS*, pages 44:1–44:13. ACM, 2022. doi:10.1145/3531130.3533360.

20 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Propositional dynamic logic for hyperproperties. In *Proc. 31st CONCUR*, LIPIcs 171, pages 50:1–50:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.50.

21 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 4(POPL), 2021. doi:10.1145/3434319.

22 Joseph Y. Halpern and Kevin R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1), 2008. doi:10.1145/1410234.1410239.

23 Joseph Y. Halpern and Moshe Y. Vardi. The Complexity of Reasoning about Knowledge and Time: Extended Abstract. In *Proc. 18th STOC*, pages 304–315. ACM, 1986. doi:10.1145/12130.12161.

24 Andreas Krebs, Arne Meier, Jonni Virtema, and Martin Zimmermann. Team Semantics for the Specification and Verification of Hyperproperties. In *Proc. 43rd MFCS*, LIPIcs 117, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.10.

25 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods Syst. Des.*, 34(2):83–103, 2009. doi:10.1007/S10703-009-0067-Z.

26 Martin Lück. On the complexity of linear temporal logic with team semantics. *Theor. Comput. Sci.*, 837:1–25, 2020. doi:10.1016/j.tcs.2020.04.019.

27 Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems - Specification*. Springer-Verlag, 1992. doi:10.1007/978-1-4612-0931-7.

28 John D. McLean. A General Theory of Composition for a Class of "Possibilistic" Properties. *IEEE Trans. Software Eng.*, 22(1):53–67, 1996. doi:10.1109/32.481534.

29 Amir Pnueli. The Temporal Logic of Programs. In *Proc. 18th FOCS*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

30 Markus N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Saarland University, 2016. URL: http://scidok.sulb.uni-saarland.de/volltexte/2016/6387/.

31 Meera Sampath, Raja Sengupta, Stephen Lafortune, Kazin Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Autom. Control.*, 40(9):1555–1575, 1995. doi:10.1109/9.412626.

32 A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987. doi:10.1016/0304-3975(87)90008-9.

33 Ron van der Meyden and Nikolay V. Shilov. Model checking knowledge and time in systems with perfect recall (extended abstract). In *Proc. 19th FSTTCS*, LNCS 1738, pages 432–445. Springer, 1999. doi:10.1007/3-540-46691-6_35.

34 Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

35 Jonni Virtema, Jana Hofmann, Bernd Finkbeiner, Juha Kontinen, and Fan Yang. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. In *Proc. 41st IARCS FSTTCS*, LIPIcs 213, pages 52:1–52:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.52.

36 Steve Zdancewic and Andrew C. Myers. Observational Determinism for Concurrent Program Security. In *Proc. 16th IEEE CSFW-16*, pages 29–43. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

Promptness and Fairness in Muller LTL Formulas

Damien Busatto-Gaston
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Youssouf Oualhadj
Univ Paris Est Creteil, LACL, F-94010 Creteil, France
CNRS, ReLaX, IRL 2000, Siruseri, India

Léo Tible
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Daniele Varacca
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Abstract

In this paper we consider two different views of the model checking problem for the Linear Temporal Logic (LTL). On the one hand, we consider the *universal* model checking problem for LTL, where one asks that for a given system and a given formula all the runs of the system satisfy the formula. On the other hand, the *fair* model checking problem for LTL asks that for a given system and a given formula almost all the runs of the system satisfy the formula.

It was shown that these two problems have the same theoretical complexity, *i.e.* PSPACE-complete. A less expensive fragment was identified in a previous work, namely the *Muller fragment*, which consists of combinations of repeated reachability properties.

We consider *prompt* LTL formulas (pLTL), that extend LTL with an additional operator, *i.e.* the *prompt-eventually*. This operator ensures the existence of a bound such that reachability properties are satisfied within this bound. This extension comes at no cost since the model checking problem remains PSPACE-complete.

We show that the corresponding Muller fragment of pLTL, with prompt repeated reachability properties, enjoys similar computational improvements. Another feature of Muller formulas is that the model checking problem becomes easier under the fairness assumption. This distinction is lost in the prompt setting, as we show that the two problems are equivalent instance-wise. Subsequently, we identify a new prefix independent fragment of pLTL for which the fair model checking problem is less expensive than the universal one.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Model checking, Fairness, Temporal logics

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.15

Related Version *Full Version*: https://arxiv.org/abs/2204.13215v5 [4]

1 Introduction

Linear Temporal Logic (LTL) allows system designers to easily describe behavioral properties of a system [17]. Its expressive power and convenience of use proved useful in many areas such as system design and verification [8, 20], agent planning [5, 12], knowledge representation [11], and control and synthesis [18, 19]. At the heart of these applications a fundamental formal approach is always to be found, *i.e.* the *model checking problem* [22].

Universal and fair model checking. When trying to verify a system against a specification, one usually models the system as directed graphs called *Labeled Transition Systems* (LTS). A run of the system is an infinite path in the LTS. The standard approach to model checking consists in verifying that all possible runs of the LTS comply with the specification. Some

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 15; pp. 15:1–15:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

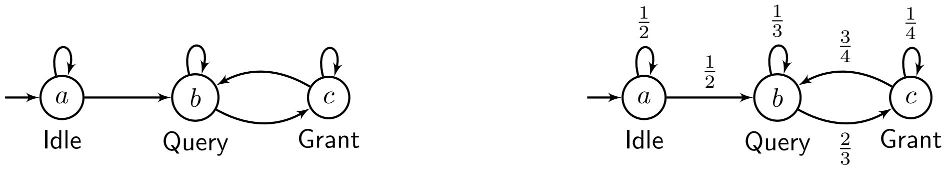


Figure 1 A protocol modeled as an LTS on the left, and as a probabilistic system on the right.

systems may not satisfy the specification because of a few *unlikely* runs. To avoid discarding these systems, the *fair* model checking approach gives a formal definition of *small sets* of executions, and then verifies that the set of executions that do not satisfy the specification is indeed small.

▶ **Example 1.** Consider the example on the left of Figure 1, this figure models a "toy protocol" that could either be in an idle state, a querying state or granting state. Assume now that the modeler wants to check that **the protocol is not always idle**. Without any fairness assumption, this system will be rejected since it could always repeat the loop on state a. Now, consider that this LTS is an abstracted view of the real protocol, which is a probabilistic system modeled as a Markov chain on the right of Figure 1. Then, the probability associated with the system remaining idle forever is 0. We can decide to ignore this "unlikely" possibility and say that the system *fairly* satisfies the specification.

In general, a set of executions of a Markov chain is considered *small* if it has probability 0. It turns out that the precise probabilities that appear in the system have no impact on which sets of executions have probability 0. Thus, we can assume without loss of generality every probability distribution to be uniform, and represent systems as LTS instead.

LTL model checking. The complexity of verifying that every run of a system satisfies an LTL formula is known to be a PSPACE-complete problem [22]. This complexity is measured with respect to the size of the LTS and its specification, an LTL formula in our case. This high complexity is due to the fact that one has to encode the specification as a *non-deterministic Büchi automaton*, that can be exponential in size. This yields an exponential blow-up and further techniques are required to keep the complexity within PSPACE, see *e.g.* [1].

In order to circumvent this blow-up, a natural idea is to identify fragments of LTL where the model checking problem becomes easier to solve. In the seminal work of Sistla and Clarke [22], they identified fragments where model checking is coNP-complete, which is more amenable to implementation than PSPACE as it opens the door to symbolic approaches using SAT-solvers. In particular they showed this complexity for the class of formulas that consists of Boolean combinations of reachability properties. We also cite the fragment identified by Muscholl et al. [16]. This fragment is obtained by prohibiting the use of reachability operators (until) and restricting the formula to exclusively using next operators indexed by a letter. They showed that the model checking problem is NP-complete for this fragment. Finally we highlight the work of Emerson et al. [10] where they studied the fragment of *Muller formulas*, *i.e.* the fragment combining repeated reachability (*i.e.* Büchi) properties. They showed that model checking becomes coNP-complete for Muller formulas.

The PSPACE-complete complexity result also holds for the *fair* model checking problem [9]. However, for the fragment of *Muller Formulas*, while the universal model checking problem becomes coNP-complete, the fair model checking problem can be solved by a polynomial time algorithm presented in [21]. In other words, this fragment allows one to take advantage of the fairness assumption to obtain tractable model checking procedures.

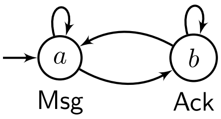


Figure 2 An LTS that satisfies a Muller formula but not its *prompt* variant, as per Example 3.

▶ **Example 2.** We go back to the example of Figure 1, and consider the following specification: **infinitely often a query is made and infinitely often a grant is granted**. This specification can be expressed as a conjunction of repeated reachability objectives, and thus as a Muller formula.

Prompt Formulas. Consider the following natural specification for messaging protocols: **All along the execution, whenever a message is sent an acknowledgment is received at a later step**. A system may satisfy this specification in an unpractical way, where the waiting time for the acknowledgment grows arbitrarily large along some executions. *Prompt LTL* (pLTL), introduced by Kupferman et al. [15], can express a variant of this specification that enforces an upper limit on waiting times: **There exists a bound k such that, along any execution, whenever a message is sent an acknowledgment is received within the next k steps**

▶ **Example 3.** Consider the LTS of Figure 2, and consider the specification asking that either Msg or Ack is seen infinitely often. Clearly enough, any infinite path in the system will satisfy this specification. Consider now the prompt variant of this specification, asking for the existence of a bound k such that either Msg is seen infinitely often in a prompt way, *i.e.* with a maximum of k steps in between successive occurrences, or Ack is seen infinitely often in a prompt way. Now, for any bound $k \geq 0$, the run $a^{k+1}b^{k+1}\cdots$ does not satisfy this specification. As such, the system does not satisfy the prompt specification.

The model checking problem for pLTL is also known to be PSPACE-complete [15]. This is achieved through an *efficient translation* into LTL. The fair model checking problem has the same complexity as well: although an explicit proof is not published, a careful inspection of the proof of [15] shows that the translation into LTL formulas holds for the fair setting, and thus it is sufficient to invoke the algorithm from [9] without further blowup.

In this paper we consider the *Muller fragment* of pLTL, that combines the prompt variants of repeated reachability properties such as the one described in Example 3.

Contributions and organisation. Our original contributions are as follows.

We first show that universal model checking for the Muller fragment of pLTL is coNP-complete. In order to show the membership, we had to depart from the already existing reduction to classical LTL and develop new technical tools. Roughly speaking, we develop combinatorial tools to represent runs and the reasons why they might not satisfy a prompt repeated reachability property of bound k. If one thinks about a faulty run as an infinite run containing a finite window of k consecutive faulty states, then *pumping* a cycle within that window leads to a longer window of faulty states. Thus, this run can be used as a generator for faulty runs of arbitrarily long bounds $k' > k$, cf. Lem. 14. However, one has to pay particular attention to the case where temporal operators are nested. In particular, “pumping” these finite sequences of faulty runs is done according to a well chosen order, cf. Section 3.1 for a formal definition of the notion of *multi-pumpings*.

Table 1 Summary of our contributions on variants of the model checking problem.

Model checking	non-prompt		prompt		
	LTL	Muller	pLTL	Muller	initialized Muller
Universal	PSPACE-c [22]	coNP-c [10]	PSPACE-c [15]	coNP-c Thm. 9, 21	coNP Thm. 26
Fair	PSPACE-c [9]	PTIME [21]	PSPACE-c [15]		PTIME Thm. 28

In Section 3.2, we prove a *small witness property.* This notion, in some sense, efficiently stores data about the existence of counter-examples, and results in a coNP procedure.

Our second contribution is to show that the fair model checking problem for this fragment does not need to be studied separately as it coincides with the universal model checking problem, cf. Thm. 21. Indeed we prove that if a system is a fair model for some pLTL Muller formula, then every path must satisfy the formula.

Further, we note that prompt Muller formulas may sometimes require "too much" from a system. In particular it is possible for a system to violate a specification during an *initial* phase of the execution, but once it enters a *steady regime* the specification might be satisfied.

Our last contribution is to address this issue by introducing the *initialized Muller* fragment for pLTL. This fragment expresses the fact that a system should satisfy a specification in the long run, *i.e.* we ignore the finite initialization phase. This vision is inspired from *prefix independent* specifications. Such specification are only interested in the asymptotic behavior of a system. For instance, *parity, Rabin, Street, Büchi* are all ω-regular specifications whose satisfaction is independent of any finite prefix [1]. Not only do these specifications seem to be more suited to real life applications, they also in general enjoy nice properties, especially in the probabilistic setting, cf. [6, 13, 14]. We also mention results [3, 7] where prefix independence has been considered in a setting rather close to ours, and there again they exhibited well behaved properties [2].

In our case, we show that the fair model checking of prompt Muller formulas is more tractable on initialized formulas: the universal model checking is still in coNP, but there is a polynomial time algorithm solving the fair model checking problem for this fragment.

Due to page limit, ommitted material can be found at `https://arxiv.org/pdf/2204.13215`.

2 Preliminaries

Throughout the document we will use the following notations and conventions: AP is a finite set of atomic propositions. For an arbitrary set E, E^* is the set of finite sequences in E, and E^ω is the set of infinite sequences of E. When E is a finite set, $|E|$ will denote its size.

Labelled Transition System. An *LTS* is a tuple $\mathcal{S} = \left\langle \mathsf{S}, s_{\mathsf{init}}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \right\rangle$ such that S is a set of states, $s_{\mathsf{init}} \in \mathsf{S}$ is an initial state, $\mathsf{T} \subseteq \mathsf{S} \times \mathsf{S}$ is a set of transitions, and $\mathsf{lbl} \ : \ S \to 2^{\mathsf{AP}}$ is a labeling function mapping every state to the atomic propositions that hold on it.

For a state $s \in \mathsf{S}$, the set of successors of s is $\mathsf{Succ}(s) = \{t \in \mathsf{S} \mid (s,t) \in \mathsf{T}\}$. A *finite path* in $\mathcal{S}$ is a finite sequence of states $\pi = s_0 s_1 \cdots s_k$ of length $k+1$ such that $\forall 0 \leq i \leq k-1, s_{i+1} \in \mathsf{Succ}(s_i)$. We denote by $|\pi|$ the length of π, *i.e.* $|\pi| = k+1$. A *run* in $\mathcal{S}$ is an infinite sequence of states $\rho = s_0 s_1 \cdots$ such that $\forall i \geq 0$, $s_{i+1} \in \mathsf{Succ}(s_i)$. Let ρ be a run and let $i \geq 0$, then $\rho[i] = s_i$, $\rho[i..] = s_i s_{i+1} \cdots$, that is the infinite suffix starting in the

$(i+1)$th letter, and $\rho[..i]$ is the prefix up to the (i)th position, that is $\rho[..i] = s_0 \cdots s_{i-1}$. For $i < j$, $\rho[i..j]$ is the finite path $s_i \cdots s_{j-1}$. We use the same notations for a finite path π, and in this case $\pi[i..]$ will be a finite suffix. The concatenation of a finite path π with a finite path (or a run) π' is denoted $\pi\pi'$. A cycle (or a loop) is a finite path $\pi = s_0 s_1 \cdots s_k$ such that $s_0 \in \mathsf{Succ}(s_k)$. In particular, the finite path π^n repeats for n iterations the cycle π.

The set of states visited infinitely often by a run ρ is denoted $\mathsf{Inf}(\rho)$, and is formally defined as $\{s \in \mathsf{S} \mid \forall i \geq 0, \exists j \geq i, \rho[j] = s\}$. Finally, let FPaths (resp. Runs) be the set of all the finite paths (resp. runs) in $\mathcal{S}$, and let $\mathsf{Runs}_{\mathsf{init}}$ be the set of all runs starting from s_{init}, that is $\{\rho \in \mathsf{Runs} \mid \rho[0] = s_{\mathsf{init}}\}$. These notations assume that $\mathcal{S}$ is clear from context.

Linear Temporal Logic. An *LTL* formula φ is defined using the following grammar:

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi\ ,\ \text{ where } \alpha \text{ ranges over } \mathsf{AP}.$$

Runs ρ of $\mathcal{S}$ are evaluated inductively over LTL formulas as follows:

$$\begin{aligned}
&\rho \models \alpha \text{ iff } \alpha \in \mathsf{lbl}(\rho[0])\ , && \rho \models \neg\varphi \text{ iff } \rho \not\models \varphi\ ,\\
&\rho \models \varphi \vee \psi \text{ iff } \rho \models \varphi \text{ or } \rho \models \psi\ , && \rho \models \mathsf{X}\,\varphi \text{ iff } \rho[1..] \models \varphi\ ,\\
&\rho \models \varphi\,\mathsf{U}\,\psi \text{ iff } \exists i \geq 0,\ \rho[i..] \models \psi \text{ and } \forall j < i,\ \rho[j..] \models \varphi\ ,
\end{aligned}$$

A *state formula* is a formula that only contains Boolean operators ($\neg$ and $\vee$) and atomic propositions, and is thus entirely evaluated on the first position of ρ. The operators X and U are called *temporal operators*. We derive all standard Boolean operators from $\neg$ and $\vee$, let $\top$ and $\perp$ be atomic propositions that are respectively always true and always false, and define a few extra temporal operators as syntactic sugar: $\mathsf{F}\,\varphi = \top\,\mathsf{U}\,\varphi$, $\mathsf{G}\,\varphi = \neg\,\mathsf{F}\,\neg\varphi$, $\mathsf{F}^{\infty}\,\varphi = \mathsf{G}\,\mathsf{F}\,\varphi$. The formula $\mathsf{F}\,\varphi$ is true for any run where φ is *eventually* true, while the formula $\mathsf{G}\,\varphi$ is true for any run where φ *always* holds. The formula $\mathsf{F}^{\infty}\,\varphi$ holds for any run where there exists infinitely many positions from where φ is true, and is used to encode repeated reachability properties, such as the winning condition of a Büchi automaton.

► **Problem 4** (Universal model checking)**.** *Given an LTS $\mathcal{S}$ and an LTL formula φ, does $\rho \models \varphi$ hold true for every run $\rho \in \mathsf{Runs}_{\mathsf{init}}$? In this case, we write $\mathcal{S} \models \varphi$.*

The universal model checking problem is PSPACE-complete [22]. We express the complexity of our model checking problems with respect to both the size of the formula $|\varphi|$, *i.e.* the number of operators that appear in φ, and the size of the system $|\mathcal{S}|$, *i.e.* $|\mathsf{T}| + |\mathsf{S}| \max_{s \in \mathsf{S}} |\mathsf{lbl}(s)|$.

Fairness in model checking. The fairness assumption presented in Example 1 relies on the idea that a set of runs is sometimes considered quantitatively *small.* In particular, we use a *fair coin* to view an LTS as a probabilistic system and derive a measure over paths. At each state, a fair coin is flipped and the successor state is chosen accordingly. This coin flipping procedure is assumed to be i.i.d. and it induces a natural probability measure over $\mathsf{Runs}_{\mathsf{init}}$. In the fair model checking problem, one asks whether *almost all* the runs of an LTS satisfy a given formula φ *i.e.* if a run obtained from the fair coin process satisfies φ with probability 1.

Formally, in order to build the probability measure over $\mathsf{Runs}_{\mathsf{init}}$, we use the classical notion of *cylinders.* For $\pi \in \mathsf{FPaths}$, the cylinder induced by π, denoted $\mathsf{Cyl}(\pi)$, is the set $\{\rho \in \mathsf{Runs} \mid \pi \text{ is a prefix of } \rho\}$. Then, the probability of a run being in a cylinder $\mathbb{P}_{\mathcal{S}}(\mathsf{Cyl}(\pi))$ is defined as the probability that π is followed under the fair coin process. This measure can be uniquely extended over the set $\mathsf{Runs}_{\mathsf{init}}$ using Carathéodory's extension theorem.

▶ **Problem 5** (Fair model checking). *Given an LTS $\mathcal{S}$ and an LTL formula φ, does it holds that $\mathbb{P}_{\mathcal{S}}(\{\rho \in \mathsf{Runs}_{\mathsf{init}} \mid \rho \models \varphi\}) = 1$? In this case, we will $\mathcal{S} \models_{\mathsf{AS}} \varphi$, where $\models_{\mathsf{AS}}$ stands for the "almost sure" satisfaction of a formula.*

Under the fairness assumption, a model checking procedure is intuitively allowed to ignore unrealistic behaviours. However, this does not simplify the complexity of the problem, as it has been shown that the fair model checking problem is also PSPACE-complete [9].

The *Muller* fragment of LTL. In an effort to obtain lower complexity results for the model checking problem, subclasses of LTL formulas, defined by syntactic restrictions, have been considered. In particular, an LTL formula φ is in the *Muller fragment*, denoted $\varphi \in \mathcal{L}(\mathsf{F}^\infty)$, if the repeated reachability operator F^∞ is the only temporal operator that is allowed:

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{F}^\infty \varphi \ .$$

The key property of Muller formulas is that their satisfaction on a run ρ is entirely determined by the initial state and $\mathsf{Inf}(\rho)$, the states visited infinitely often. This leads to lower complexity results: in [21], it was shown that the universal model checking problem for Muller formulas is coNP-complete, while the fair model checking problem can be solved in polynomial time.

Promptness in LTL. A prompt LTL formula φ is defined according to the following grammar [15]: $\varphi ::= \alpha \mid \neg\alpha \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\,\varphi \mid \varphi\,\mathsf{U}\,\varphi \mid \varphi\,\mathsf{R}\,\varphi \mid \mathsf{F_P}\,\varphi$.

The main difference with LTL lies in the addition of $\mathsf{F_P}$. This operator states that φ has to be satisfied eventually, but in a "prompt" fashion. The semantics of this operator are defined with respect to a bound $k \in \mathbb{N}$. For a given k, we write $(\rho, k) \models \mathsf{F_P}\,\varphi$ if $\exists i \leq k,\ (\rho[i..], k) \models \varphi$, in contrast, recall that $\rho \models \mathsf{F}\,\varphi$ if $\exists i \in \mathbb{N}, \rho[i..] \models \varphi$. The other operators ignore the bound k and are evaluated using the semantics of LTL defined earlier. Another difference with LTL is that Boolean negations are not allowed in pLTL, as the negation of the newly added operator $\mathsf{F_P}$ is deemed unnatural from a modelling point of view. Therefore, the grammar explicitly contains the conjunction $\wedge$, and the *release* operator R, the dual of the *until* operator U, previously expressed with negations.

▶ **Problem 6** (Universal prompt model checking). *Given an LTS $\mathcal{S}$ and a pLTL formula φ, does there exists a bound $k \in \mathbb{N}$ such that for all $\rho \in \mathsf{Runs}_{\mathsf{init}}$ in $\mathcal{S}$, $(\rho, k) \models \varphi$? In this case, we write $\mathcal{S} \models \varphi$.*

▶ **Problem 7** (Fair prompt model checking). *Given an LTS $\mathcal{S}$ and a pLTL formula φ, does there exists a bound $k \in \mathbb{N}$ such that $\mathbb{P}_{\mathcal{S}}(\{\rho \in \mathsf{Runs}_{\mathsf{init}} \mid (\rho, k) \models \varphi\}) = 1$? In this case, we write $\mathcal{S} \models_{\mathsf{AS}} \varphi$.*

The addition of the prompt eventually operator $\mathsf{F_P}$ comes at no extra cost compared with LTL, as both universal and fair model checking remain PSPACE-complete [15].

The prompt Muller fragment. In this work, we consider a subclass of pLTL formulas inspired by Muller formulas. We define a new operator, $\mathsf{F}^\infty_\mathsf{P}\,\varphi = \mathsf{G}\,\mathsf{F_P}\,\varphi$, as a prompt variant of F^∞. Thus, $\mathsf{F}^\infty_\mathsf{P}\,\varphi$ holds true for a pair (ρ, k) if from every position $i \in \mathbb{N}$, a position $j \in [i, i+k]$ can be found so that $(\rho[j..], k) \models \varphi$. A pLTL fromula φ is in the *prompt Muller fragment*, denoted $\varphi \in \mathcal{L}(\mathsf{F}^\infty_\mathsf{P})$, if it is obtained by the following grammar:

$$\varphi ::= \alpha \mid \neg\alpha \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{F}^\infty_\mathsf{P}\,\varphi \,.$$

▶ **Example 8.** Consider the specifications mentioned in Example 3, that asks that either Msg or Ack is seen infinitely often. This is expressed by the Muller formula $\mathsf{F}^{\infty}\,\mathsf{Msg} \vee \mathsf{F}^{\infty}\,\mathsf{Ack}$. The prompt variant is the formula $\mathsf{F}^{\infty}_{\mathsf{P}}\,\mathsf{Msg} \vee \mathsf{F}^{\infty}_{\mathsf{P}}\,\mathsf{Ack}$, that intuitively asks that either Msg or Ack are seen *frequently*, *i.e.* with a frequency that does not vanish along the execution.

3 Prompt Muller formulas

A remarkable property of the Muller fragment of LTL is that the satisfaction of a formula only depends on the asymptotic behavior of the system. Therefore, as shown in [21] (a corollary of a result from [10]), a system satisfies a Muller formula when every strongly connected set satisfies the formula. This property yields an easier model checking problem, namely coNP-complete. In this section we focus on the prompt Muller fragment of pLTL. We show that the complexity of the model checking problem is again coNP-complete. However, the techniques involved are different. Indeed, they do not follow from structural properties of the transition system. We introduce combinatorial tools to establish a small witness property. We further deepen our study by considering runs obtained under the fairness assumption. We show that prompt Muller formulas describe the same set of runs with or without the fairness assumption. This differs from (non-prompt) Muller formulas, where fairness allowed for more tractable approaches. As a corollary, we obtain that fair model checking is as expensive as universal model checking for the prompt Muller fragment.

Most of this section will be devoted to the proof of the following theorem:

▶ **Theorem 9.** *The universal model checking problem for $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$ is* coNP*-complete.*

3.1 Pumping a counter example

The first stepping stone to prove Thm. 9 is to define the notion of *pumping*. The idea is quite simple: for a run ρ, a pumping of ρ is a run where some cycle of ρ has been repeated. Formally, it is defined as follows.

▶ **Definition 10** (Pumping). *Given an LTS $\mathcal{S}$ and a run $\rho \in$ Runs, a* pumping *of ρ is a run $\rho' \in$ Runs such that there exist $0 \leq i < j$ and $l > 0$, with $\rho[i..j]$ a cycle, so that $\rho' = \rho[..i]\rho[i..j]^{l-1}\rho[i..]$.*

If $i = 0$, then by convention $\rho[..0] = \varepsilon$ is the empty path. In particular, ρ' contains l copies of the cycle $\rho[i..j]$, as the last one is in $\rho[i..]$. We will say that a pumping is a 1-pumping, as one cycle is iterated. A natural extension of this notion is to allow for the pumping of several cycles in a run. Formally, we define a *multi-pumping* inductively as follows, with a run being the only 0-pumping of itself.

▶ **Definition 11** (Multi-pumping). *Given an LTS $\mathcal{S}$, a run $\rho \in$ Runs, and $n > 0$, an n-*pumping *of ρ is a run $\rho' \in$ Runs such that there exist $0 \leq i < j$, with $\rho[i..j]$ a cycle, and there is a $(n-1)$-pumping $\tilde{\rho}$ of $\rho[i..]$ and some $l > 0$ so that $\rho' = \rho[..i]\rho[i..j]^{l-1}\tilde{\rho}$. A* multi-pumping *of ρ is a run ρ' such that ρ' is an n-pumping of ρ for some $n \geq 0$.*

The construction of an n-pumping allows for several cycles of a run to be pumped, but only consecutively in a left-to-right fashion. This keeps the n individual pumpings ordered and will allow for inductive proofs on n.

▶ **Example 12.** Consider the run $\rho = ab(abcb)aad^{\omega}$ represented if Figure 3, and note that $abcb$ is a cycle that can be pumped. Therefore, $\rho' = ab(abcb)^4aad^{\omega}$ is a 1-pumping of ρ. Now, imagine that you also want to pump the cycle bc. By definition, further pumpings can only occur starting from the last "copy" of the cycle $abcb$. In particular, $\rho'' = ab(abcb)^3a(bc)^3baad^{\omega}$ is a 2-pumping of ρ.

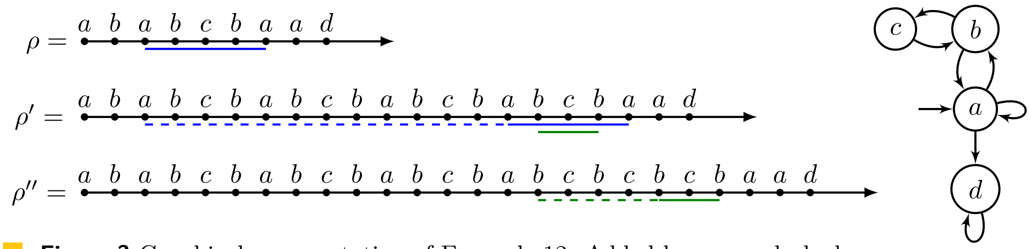

$\rho =$
a b a b c b a a d
$\rho' =$
a b a b c b a b c b a b c b a b c b a a d
$\rho'' =$
a b a b c b a b c b a b c b a b c b c b c b a a d
c
b
a
d

Figure 3 Graphical representation of Example 12. Added loops are dashed.

The main property of multi-pumpings is that they preserve counter-examples, *i.e.* if a run ρ does not satisfy a prompt Muller formula φ for some bound, then for the same bound any multi-pumping of ρ will not satisfy φ either.

▶ **Proposition 13.** *Given an lts $\mathcal{S}$, a run $\rho \in \mathsf{Runs}$, a multi-pumping ρ' of ρ, a formula $\varphi \in \mathcal{L}(\mathsf{F}_\mathsf{P}^\infty)$ and $k \geq 0$, if $(\rho, k) \not\models \varphi$, then $(\rho', k) \not\models \varphi$.*

Proof scheme. Intuitively, if there is no nesting of $\mathsf{F}_\mathsf{P}^\infty$ operator in the formula, then the formula being false on (ρ, k) only depends on "faulty" windows of length k that can be found in ρ. As a multi-pumping only extends faulty windows by duplicating cycles, any faulty window in the original run also exists somewhere in the multi-pumping. If φ has some nesting of $\mathsf{F}_\mathsf{P}^\infty$ operators, then a window being faulty or not depends on the suffix run that comes after it, which makes these considerations significantly more involved technically, as pumping changes the suffix of our runs. A full proof is detailed in Appendix A. ◀

The second core property of multi-pumpings, derived from Prop. 13, is that they can be used to generate counter-examples for arbitrarily large bounds k, as long as there is a counter example for the bound $N = |S| + 1$. This is formalised as follows:

▶ **Lemma 14.** *Let $\mathcal{S} = \langle \mathsf{S}, s_\mathsf{init}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \rangle$ be an LTS, $\varphi \in \mathcal{L}(\mathsf{F}_\mathsf{P}^\infty)$, and let $N = |S| + 1$. If there is $\rho_N \in \mathsf{Runs}_\mathsf{init}$ such that $(\rho_N, N) \not\models \varphi$ then for all $k \geq N$, there is a multi-pumping ρ_k of ρ_N such that $(\rho_k, k) \not\models \varphi$.*

Thus, in order to show that a system *does not satisfy* a prompt Muller formula for any bound k, it is enough to exhibit a single run ρ so that $(\rho, N) \not\models \varphi$. Indeed, $(\rho, k) \not\models \varphi$ is immediate for every $k \leq N$ by definition of pLTL,[1] and for any $k > N$, Lem. 14 can be used to generate another run that will not satisfy φ either.

3.2 Canonical representation for witnesses

Let ρ be a run, and assume that $(\rho, N) \not\models \varphi$. Such a run witnesses that $\mathcal{S} \not\models \varphi$, however, it is an infinite sequence. Moreover, even with a finite representation of ρ, checking that $(\rho, N) \not\models \varphi$ remains a challenging task. In this section, we introduce a new data structure, that carries a representation of ρ and enough information to efficiently check that $(\rho, N) \not\models \varphi$.

Given an LTL formula φ, let $\mathsf{SubF}(\varphi)$ be the set of all subformulas of φ.

▶ **Definition 15.** *Given an LTS $\mathcal{S} = \langle \mathsf{S}, s_\mathsf{init}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \rangle$, a run $\rho \in \mathsf{Runs}$, a bound $k \geq 0$ and a formula $\varphi \in \mathcal{L}(\mathsf{F}_\mathsf{P}^\infty)$, a* witness *for ρ with bound k is a function $\mathsf{W}_{\rho,k} : \mathsf{SubF}(\varphi) \mapsto 2^{\mathbb{N}}$ that maps subformulas to finite sets and satisfies the following conditions:*

[1] Intuitively, if a faulty window exists to falsify an $\mathsf{F}_\mathsf{P}^\infty$ operator for some bound, then every shorter bound admits the same faulty window to falsify $\mathsf{F}_\mathsf{P}^\infty$.

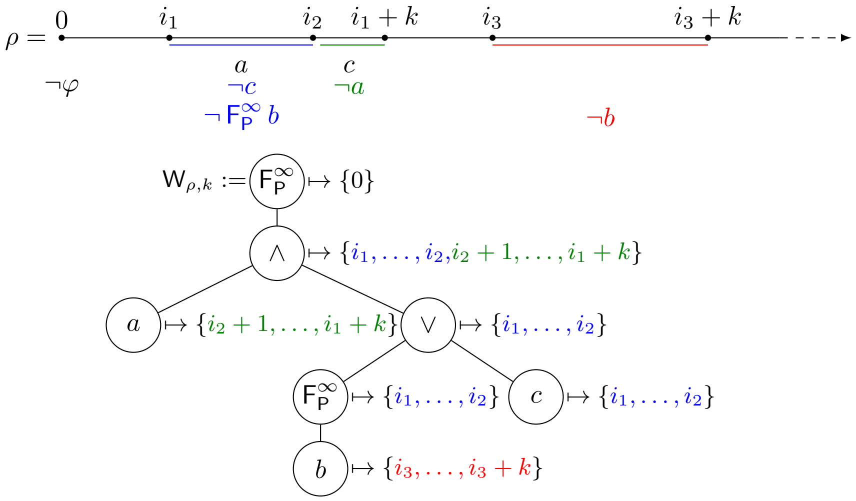

$\rho =$
0
i_1
i_2
$i_1 + k$
i_3
$i_3 + k$
$\neg\varphi$
a
$\neg c$
$\neg \mathsf{F}^{\infty}_{\mathsf{P}}\, b$
c
$\neg a$
$\neg b$
$\mathsf{W}_{\rho,k} :=$
$\mathsf{F}^{\infty}_{\mathsf{P}} \mapsto \{0\}$
$\wedge \mapsto \{i_1, \dots, i_2, i_2 + 1, \dots, i_1 + k\}$
$a \mapsto \{i_2 + 1, \dots, i_1 + k\}$
$\vee \mapsto \{i_1, \dots, i_2\}$
$\mathsf{F}^{\infty}_{\mathsf{P}} \mapsto \{i_1, \dots, i_2\}$
$c \mapsto \{i_1, \dots, i_2\}$
$b \mapsto \{i_3, \dots, i_3 + k\}$

Figure 4 Example of a run that does not satisfy a formula, and the corresponding witness.

- $\forall i \in \mathsf{W}_{\rho,k}(\alpha), \alpha \notin \mathsf{lbl}(\rho[i])$
- $\forall i \in \mathsf{W}_{\rho,k}(\neg\alpha), \alpha \in \mathsf{lbl}(\rho[i])$
- $\forall i \in \mathsf{W}_{\rho,k}(\psi_1 \vee \psi_2), i \in \mathsf{W}_{\rho,k}(\psi_1) \cap \mathsf{W}_{\rho,k}(\psi_2)$
- $\forall i \in \mathsf{W}_{\rho,k}(\psi_1 \wedge \psi_2), i \in \mathsf{W}_{\rho,k}(\psi_1) \cup \mathsf{W}_{\rho,k}(\psi_2)$
- $\forall i \in \mathsf{W}_{\rho,k}(\mathsf{F}^{\infty}_{\mathsf{P}}\, \psi), \exists i' \geq i, \forall i' \leq j \leq i' + k, j \in \mathsf{W}_{\rho,k}(\psi)$
- $0 \in \mathsf{W}_{\rho,k}(\varphi)$

Then, we denote $\max(\mathsf{W}_{\rho,k}) = \max\{i \in \mathbb{N} \mid \psi \in \mathsf{SubF}(\varphi) \wedge i \in \mathsf{W}(\psi)\}$.

Intuitively, such a witness justifies that $(\rho, k) \not\models \varphi$ by describing for each subformula a relevant set of positions from where they are falsified. In particular, $0 \in \mathsf{W}_{\rho,k}(\varphi)$ ensures $(\rho, k) \not\models \varphi$. Then, the other conditions ensure that inductively, the positions falsify subformulas up to reaching the atomic propositions.

▶ **Example 16.** Consider the run ρ of Figure 4, and the formula $\varphi = \mathsf{F}^{\infty}_{\mathsf{P}}(a \wedge (\mathsf{F}^{\infty}_{\mathsf{P}}\, b \vee c))$. Some of the labels of states in ρ are represented, so that a holds between positions i_1 and i_2 for example. The syntactic tree of the formula is also represented in Figure 4, where every subtree corresponds to a subformula. Here, for a given bound k, it is assumed that $(\rho, k) \not\models \varphi$, and a potential witness for ρ with bound k is described on the syntactic tree of φ: every subformula is mapped to a set of positions. Then, one can check that every inductive rule of Def. 15 holds. For example, for the subformula a, no state $\rho[i]$ such that $i \in \mathsf{W}_{\rho,k}$ has label a, and for the subformula $a \wedge (\mathsf{F}^{\infty}_{\mathsf{P}}\, b \vee c)$, it holds that $\mathsf{W}_{\rho,k}(a \wedge (\mathsf{F}^{\infty}_{\mathsf{P}}\, b \vee c)) = \mathsf{W}_{\rho,k}(a) \cup \mathsf{W}_{\rho,k}(\mathsf{F}^{\infty}_{\mathsf{P}}\, b \vee c)$. Note how the witness describes which positions along ρ are relevant to prove where each subformula is falsified, up to the position 0 at the root $\mathsf{W}_{\rho,k}(\varphi)$, so that $(\rho, k) \not\models \varphi$.

Note that Def. 15 does not require a witness W to be small in size, as it could:

- map subformulas ψ_1 and ψ_2 to positions that are lost by intersection in $\mathsf{W}(\psi_1 \vee \psi_2)$,
- map ψ to more than the k positions needed to falsify $\mathsf{W}(\mathsf{F}^{\infty}_{\mathsf{P}}\, \psi)$, and
- use positions that are needlessly far away, so that $\max(\mathsf{W})$ is too large.

We address all of these concerns by proving a *small witness property*, *i.e.*, there is a path ρ with $(\rho, k) \not\models \varphi$ iff there exists a *small* witness for some path ρ' with bound k. Here, small is meant as a polynomial upper bound on the size needed to represent W, *i.e.* $|\varphi| \max(\mathsf{W})$.

▶ **Proposition 17.** *Given a formula $\varphi \in \mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$, an LTS $\mathcal{S} = \langle \mathsf{S}, s_{\mathsf{init}}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \rangle$ and a bound $k \geq 0$, there exists a run $\rho \in \mathsf{Runs}$ such that $(\rho, k) \not\models \varphi$ iff there exists a finite path π and a function W such that for all runs $\pi\rho'$, W is a witness for $\pi\rho'$ with bound k and $|\pi| \leq (k+1)|\varphi|(|\mathsf{S}|+1)$ and $\max(\mathsf{W}) < |\pi|$.*

Proof scheme. First, we show that if $\mathcal{S} \not\models \varphi$ then a witness W can be obtained from any counter-example run based on the semantics of pLTL. Then, we show that if there exists a witness then there is one of size polynomial in k and φ. This intuitively comes from the fact that for the case $\mathsf{F}^{\infty}_{\mathsf{P}}\,\psi$, one window of length k falsifying ψ is sufficient, which prevents the sets of indexes from needing to be large. Third, we show that given a witness, we can construct a finite path π of length polynomial in the size of the witness and the size of the system such that for any run ρ', we have $(\pi\rho', k) \not\models \varphi$. This holds because the states of ρ that appear in the witness are enough to guarantee that the formula is falsified, and in-between those states, we can always find a short path. This construction yields a polynomial bound for $\max(\mathsf{W})$, and the proposition follows. ◀

Therefore, in order to show that a system does not satisfy φ, it is now enough to search for a finite path π and a witness W of polynomial size, as described in Prop. 17.

3.3 Universal model checking

We are now ready to prove coNP membership, as a corollary of Prop. 17:

▶ **Lemma 18.** *The universal model checking problem for $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$ is in* coNP*.*

Proof. Given a formula φ and a system $\mathcal{S} = \langle \mathsf{S}, s_{\mathsf{init}}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \rangle$, one can guess a witness of polynomial size for the bound $N = |\mathsf{S}| + 1$, and check that it is indeed a witness. The check can be done bottom up on the formula and is clearly polynomial, as it consists of label checks and standard set operations. Prop. 17 guarantees that the algorithm is correct, while Lem. 14 ensures that checking the bound k equal to N is equivalent to checking every bound in $\mathbb{N}$. This non-deterministic procedure is detailed in Algorithm 1. ◀

In order to finish the proof of Thm. 9, we show the following lower complexity bound.

▶ **Lemma 19.** *The universal model checking problem for $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$ is* coNP*-hard.*

This result is obtained as a reduction from the Boolean satisfiability problem to model checking, based on Figure 5. Somewhat classically, the reduction draws parallels between executions in this system and valuations over the Boolean variables $x_1, \ldots, x_n$, based on which states x_i are visited. The main novelty resides in the $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$ formula built in the reduction: as we cannot use reachability properties directly, we need to carefully encode "reaching x_i" in a roundabout way, that uses $\mathsf{F}^{\infty}_{\mathsf{P}}$ operators and the self-loops on each x_i.

Proof. Let $\bigwedge_{i=1}^{l} \bigvee_{j=1}^{3} l_{i,j}$ be an instance of 3-SAT over variables $x_1, \ldots, x_n$, where every literal $l_{i,j}$ is either a variable x or its negation $\bar{x}$. Consider the system $\mathcal{S}$ in Figure 5, and the $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$ formula $\varphi = \varphi_1 \vee \varphi_2$, obtained from $\varphi_1 = \bigvee_{i=1}^{n} (\mathsf{F}^{\infty}_{\mathsf{P}} \neg x_i \wedge \mathsf{F}^{\infty}_{\mathsf{P}} \neg \bar{x_i})$ and $\varphi_2 = \bigvee_{i=1}^{l} \bigwedge_{j=1}^{3} \mathsf{F}^{\infty}_{\mathsf{P}} \neg l_{i,j}$. We show that the 3-SAT formula is *unsatisfiable* (a coNP-hard problem) iff $\mathcal{S} \models \varphi$.

Algorithm 1 coNP algorithm for the universal model checking of $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$.

```
Data: An lts $\mathcal{S} = \langle \mathsf{S}, s_{\mathsf{init}}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \rangle$ and a formula $\varphi \in \mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$
Result: whether $\mathcal{S} \not\models \varphi$

GuessAndCheck $(\mathcal{S}, \varphi)$
    guess a finite path $\pi$ such that $|\pi| \leq (|\mathsf{S}| + 2)|\varphi|(|\mathsf{S}| + 1)$;
    guess a function $\mathsf{W} : \mathsf{SubF}(\varphi) \mapsto 2^{\mathbb{N}}$ such that $\max(\mathsf{W}) < |\pi|$;
    return $(0 \in \mathsf{W}(\varphi)) \wedge$ CheckW$(\mathcal{S}, \mathsf{W}, \varphi, \pi)$;
CheckW$(\mathcal{S}, \mathsf{W}, \varphi, \pi)$
    if $\varphi = \alpha$ then
        return $\forall i \in \mathsf{W}(\varphi), a \notin \mathsf{lbl}(\pi[i])$;
    else if $\varphi = \neg\alpha$ then
        return $\forall i \in \mathsf{W}(\varphi), a \in \mathsf{lbl}(\pi[i])$;
    else if $\varphi = \psi_1 \vee \psi_2$ then
        return $(\mathsf{W}(\varphi) = \mathsf{W}(\psi_1) \cap \mathsf{W}(\psi_2)) \wedge$ CheckW$(\mathcal{S}, \mathsf{W}, \psi_1, \pi)$ $\wedge$
          CheckW$(\mathcal{S}, \mathsf{W}, \psi_2, \pi)$;
    else if $\varphi = \psi_1 \wedge \psi_2$ then
        return $(\mathsf{W}(\varphi) = \mathsf{W}(\psi_1) \cup \mathsf{W}(\psi_2)) \wedge$ CheckW$(\mathcal{S}, \mathsf{W}, \psi_1, \pi)$ $\wedge$
          CheckW$(\mathcal{S}, \mathsf{W}, \psi_2, \pi)$;
    else if $\varphi = \mathsf{F}^{\infty}_{\mathsf{P}}\, \psi$ then
        return $(\exists 0 \leq i \leq |\pi|, \forall i \leq j \leq i + |\mathsf{S}| + 1, j \in \mathsf{W}(\psi)) \wedge$ CheckW$(\mathcal{S}, \mathsf{W}, \psi, \pi)$;
```

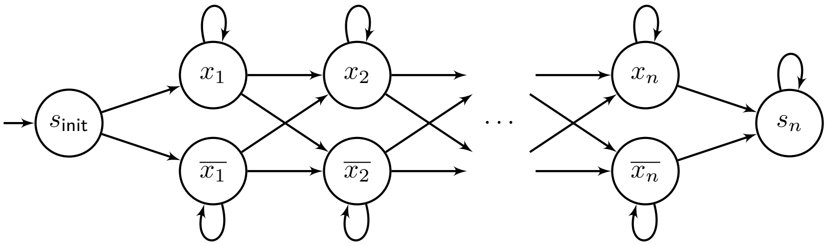


Figure 5 System used in Lem. 19. States are labeled by their name, *e.g.* $x_1 \in \mathsf{lbl}(x_1)$.

Assume that for every valuation ν, $\nu \not\models \bigwedge_{i=1}^{l} \bigvee_{j=1}^{3} l_{i,j}$, and thus $\nu \models \bigvee_{i=1}^{l} \bigwedge_{j=1}^{3} \bar{l}_{i,j}$. Every path from s_{init} to s_n can be seen as encoding a valuation ν, based on the visited variables. Then, for every run ρ in $\mathcal{S}$ that reaches s_n, we have $\rho \models \bigvee_{i=1}^{l} \bigwedge_{j=1}^{3} \mathsf{F}\, \bar{l}_{i,j}$. Note that a run that does not reach s_n (and gets stuck in one of the loops) also satisfies this formula, as removing F terms from conjunctions can only help. Moreover, we note that $\rho \models \mathsf{F}\, \bar{l}_{i,j}$ implies $\rho \models \mathsf{G}\, \neg l_{i,j}$, as a run of $\mathcal{S}$ cannot visit both a variable and its negation. Then, by definition of $\mathsf{F}^{\infty}_{\mathsf{P}}$ as $\mathsf{G}\,\mathsf{F}_{\mathsf{P}}$, we have that $\rho \models \mathsf{G}\, \neg l_{i,j}$ is equivalent with $(\rho, 0) \models \mathsf{F}^{\infty}_{\mathsf{P}}\, \neg l_{i,j}$. Therefore, for every run ρ, $(\rho, 0) \models \varphi_2$. This implies that there exists a k such that every run satisfies $\varphi_1 \vee \varphi_2$, *i.e.* $\mathcal{S} \models \varphi$.

Assume now that $\mathcal{S} \models \varphi$, *i.e.* there is a k so that every run satisfies $\varphi_1 \vee \varphi_2$. Let R be the set of runs that go through $\mathcal{S}$ by iterating every self-loop on the states x_i or $\bar{x}_i$ exactly $k + 1$ times before continuing, until s_n is reached. Then, for every run $\rho \in R$ and every variable x, we have that either $(\rho, k) \models \mathsf{F}^{\infty}_{\mathsf{P}}\, \neg x$ holds (if x is not visited), or $(\rho, k) \models \mathsf{F}^{\infty}_{\mathsf{P}}\, \neg \bar{x}$ holds (if $\bar{x}$ is not visited). This is an exclusive either/or because every state that is seen is iterated $k+1$ times, so that $(\rho, k) \not\models \varphi_1$. It follows that the runs in R must all satisfy φ_2. Since a run in R satisfies $\mathsf{F}^{\infty}_{\mathsf{P}} \neg x$ iff it satisfies $\mathsf{F}\,\bar{x}$, every run in R satisfies $\bigvee_{i=1}^{l} \bigwedge_{j=1}^{3} \mathsf{F}\,\bar{l}_{i,j}$. Then, if we interpret the runs in R as valuations ν based on which variables are visited, we have $\nu \models \bigvee_{i=1}^{l} \bigwedge_{j=1}^{3} \bar{l}_{i,j}$ for every valuations ν, so that the 3-SAT formula $\bigwedge_{i=1}^{l} \bigvee_{j=1}^{3} l_{i,j}$ is unsatisfiable. ◀

3.4 Expressiveness under the fairness assumption

Let us focus our attention on the expressiveness of the prompt Muller fragment under fairness. It turns out that for this fragment, assuming fairness does not change the interpretation of a formula. In particular, the complexity of the model checking problem remains the same.

Indeed, we note the following property of $\mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$, obtained as a corollary of Prop. 17:

▶ **Corollary 20.** *Given a formula $\varphi \in \mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$, an LTS $\mathcal{S}$ and a bound $k \geq 0$, if there exists a run ρ so that $(\rho, k) \not\models \varphi$, then there is a finite path π such that for all $\rho' \in \mathsf{Cyl}(\pi)$, $(\rho', k) \not\models \varphi$.*

Thus, $\mathcal{S} \not\models \varphi$ implies for any choice of k the existence of an entire cylinder $\mathsf{Cyl}(\pi)$ of counterexamples for the bound k. The cylinder of a finite path must always have non-zero measure under $\mathbb{P}_{\mathcal{S}}$, by definition of our coin-toss process. As such, $\mathcal{S} \not\models \varphi$ implies $\mathbb{P}_{\mathcal{S}}(\{\rho) \in \mathsf{Runs}_{\mathsf{init}} \mid (\rho, k) \models \varphi\}) < 1$ for every k, so that $\mathcal{S} \not\models_{\mathsf{AS}} \varphi$. This proves the following theorem, as the reverse implication holds by definition.

▶ **Theorem 21.** *For all LTS $\mathcal{S}$ and every formula $\varphi \in \mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$, $\mathcal{S} \models \varphi$ iff $\mathcal{S} \models_{\mathsf{AS}} \varphi$.*

Let us give some alternative intuition as to why the complexity drop between the universal and fair model checking problems for Muller formulas does not extend under promptness:

- The study of an $\mathcal{L}(\mathsf{F}^{\infty})$ formula in a system under the fairness assumption is based on the core principle that the only thing that matters is the "bottom SCC" the run ends up in, as $\mathsf{Inf}(\rho)$ is almost always equal to such a component. A PTIME algorithm is derived from this principle [21]. Crucially, this approach exploits the fact that a formula of the shape $\mathsf{F}^{\infty}\,\varphi$ is *prefix independent*, in the sense that any deviation made within a finite prefix of a run ρ will not change $\mathsf{Inf}(\rho)$, and thus will not change its satisfaction of $\mathsf{F}^{\infty}\,\varphi$.
- In the prompt setting however, a formula of the shape $\mathsf{F}^{\infty}_{\mathsf{P}}\,\varphi$ is clearly *not prefix independent*, as its satisfaction for a bound k is impacted by whether finite prefixes contain faulty windows of length k or not. Thus, the fairness assumption does not allow us to restrict the analysis of prompt Muller formulas to bottom SCCs in the same way.

4 Initialized systems

We now reflect on the expressiveness of the prompt Muller fragment. The motivation to replace $\mathsf{F}^{\infty}\,\varphi$ by $\mathsf{F}^{\infty}_{\mathsf{P}}\,\varphi$ was to reinforce the guarantees obtained by executions of the system: By enforcing a strong notion of regularity in the occurrences of φ, we prevent a good event φ from being seen infinitely often but with a vanishing frequency. Indeed, requiring $\mathsf{F}^{\infty}_{\mathsf{P}}\,\varphi$ is sufficient to imply a frequency for φ of at least $1/k$ for some k.

We now argue that this requirement may be "too strong", as some systems may be rejected despite enforcing this kind of non-zero frequency guarantee on the occurrences of φ.

▶ **Example 22.** Consider the system presented on the left in Figure 6, of atomic propositions a and b, and the prompt Muller formula $\varphi = \mathsf{F}_{\mathsf{P}}^{\infty} A \vee \mathsf{F}_{\mathsf{P}}^{\infty} B$. Despite satisfying the Muller formula $\mathsf{F}^{\infty} A \vee \mathsf{F}^{\infty} B$ (every run will either stay in a forever or jump to b and stay there forever), this system does not satisfy the prompt variant φ. However, in every run either a or b happen with a long-term average frequency of 1. the key difference here, once again, is that the long-term average frequency of an event is a prefix independent notion, unlike φ.

4.1 Towards prefix independence

We argue that being unaffected by the addition of a prefix is a desirable property for a specification: in practice, a system might require a "guarantee-less" initialization period before reaching a steady regime where stronger conditions can be enforced. We introduce a *new fragment* of pLTL, capturing specifications that allow the system to pass the initialization period while enforcing prompt Muller guarantees on the regularity of good events eventually. The intuition for this construction is based on the following reasoning:

- Coro. 20 means that the satisfaction of a formula in $\mathcal{L}(\mathsf{F}_{\mathsf{P}}^{\infty})$ can always be proven false because of some finite path. From that point of view, prompt Muller formulas behave like safety conditions $\mathsf{G}\,\alpha$, as opposed to Muller formulas that behave like repeated reachability objectives. This is the crux of what prevents prefix independence.
- In order to allow a safety formula $\mathsf{G}\,\alpha$ to accommodate for an initialisation period in the system, a natural idea is to replace it with the formula $\mathsf{F}\,\mathsf{G}\,\alpha$, that is prefix independent.
- Following this intuition, we introduce the *initialized* variant of $\mathcal{L}(\mathsf{F}_{\mathsf{P}}^{\infty})$, that contains formulas of the shape $\mathsf{F}\,\varphi$, with $\varphi \in \mathcal{L}(\mathsf{F}_{\mathsf{P}}^{\infty})$.

▶ **Example 23.** Consider again the system on the left of Figure 6, and the prompt Muller formula $\varphi = \mathsf{F}_{\mathsf{P}}^{\infty} A \vee \mathsf{F}_{\mathsf{P}}^{\infty} B$. Despite not satisfying φ, this system does satisfy $\mathsf{F}\,\varphi$, as in every run ρ either $\mathsf{G}\,a$ or $\mathsf{G}\,b$ eventually holds, and thus $(\rho, 0) \models \mathsf{F}(\mathsf{F}_{\mathsf{P}}^{\infty} A \vee \mathsf{F}_{\mathsf{P}}^{\infty} B)$. Consider now the system on the right of Figure 6. There, the run $abaabbaaabbb\ldots$ witnesses that neither φ nor $\mathsf{F}\,\varphi$ are satisfied.

We must finally address a last point of detail. Even without promptness, a Muller formula in $\mathcal{L}(\mathsf{F}^{\infty})$ may not be prefix-independent: indeed, a state formula with no F^{∞} operators solely depends on the initial state. In more general terms, the presence of atomic propositions outside of the scope of an F^{∞} prevents prefix independence, and must be forbidden. This is without loss of generality for $\mathcal{L}(\mathsf{F}^{\infty})$ formulas, as replacing such atomic propositions by true or false depending on the initial state of the system can be done as a pre-processing step [23].

Formally, we introduce $\mathcal{L}^{+}(\mathsf{F}_{\mathsf{P}}^{\infty})$ as formulas where every atomic proposition is in the scope of an $\mathsf{F}_{\mathsf{P}}^{\infty}$ operator. They are generated by the nonterminal φ in the following grammar:

$$\begin{aligned}\varphi &::= \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathsf{F}_{\mathsf{P}}^{\infty}\,\psi\\ \psi &::= \alpha \mid \neg\alpha \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathsf{F}_{\mathsf{P}}^{\infty}\,\psi\ .\end{aligned}$$

For example, $\mathsf{F}_{\mathsf{P}}^{\infty}\,\mathsf{A}$ belongs to $\mathcal{L}^{+}(\mathsf{F}_{\mathsf{P}}^{\infty})$, but $\mathsf{B} \vee \mathsf{F}_{\mathsf{P}}^{\infty}\,\mathsf{A}$ does not.

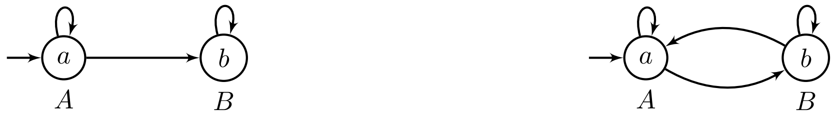


Figure 6 Two systems that satisfy $\mathsf{F}^{\infty} A \vee \mathsf{F}^{\infty} B$. The one on the left does not satisfy $\mathsf{F}_{\mathsf{P}}^{\infty} A \vee \mathsf{F}_{\mathsf{P}}^{\infty} B$ unless its "initialization period" is ignored. The rightmost one does not satisfy it either ways.

▶ **Definition 24.** *The* initialized *fragment* $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P})\right)$ *is defined as* $\{\mathsf{F}\,\varphi \mid \varphi \in \mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P})\}$*.*

This fragment enjoys the property of prefix independence:

▶ **Proposition 25.** *If ρ is a run and $\mathsf{F}\,\varphi$ is a formula in $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P})\right)$, then for any $k \geq 0$ and $i \geq 0$, $(\rho, k) \models \mathsf{F}\,\varphi$ iff $(\rho[i..], k) \models \mathsf{F}\,\varphi$.*

4.2 Universal model checking

In this section, we show that the universal model checking problem remains in coNP for $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P})\right)$, by adapting the techniques we developed for $\mathcal{L}(\mathsf{F}^\infty_\mathsf{P})$. However, we note that the lower bound is lost in the process, as the reduction detailed in Lemma 19 does not play well with $\mathsf{F}\,\varphi$ formulas.

▶ **Theorem 26.** *The universal model checking problem for $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P})\right)$ is in* coNP*.*

The idea is to make use of the same pumpings and witness structures as Section 3 to build a short witness for $\mathcal{S} \not\models \mathsf{F}\,\varphi$, while taking into account the fact that only long-term behaviours of the system should matter. This means that a "faulty window" where a subformula ψ does not hold can only serve as counter-example to $\mathsf{F}(\mathsf{F}^\infty_\mathsf{P}\,\psi)$ if it can be reached after an arbitrarily long prefix. Intuitively, faulty windows that are reached infinitely often along a run fit that description, as any suffix of the run will eventually reach them. We show that these are the only witnesses that we need, and that they remain small in size:

▶ **Proposition 27.** *Given an LTS $\mathcal{S} = \left\langle \mathsf{S}, s_{\mathsf{init}}, \mathsf{T}, \mathsf{lbl}\colon \mathsf{S} \to 2^{\mathsf{AP}} \right\rangle$, a formula $\mathsf{F}\,\varphi \in \mathsf{F}(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P}))$ and a bound $k \geq 0$, there exists a run $\rho \in \mathsf{Runs}$ such that $(\rho, k) \not\models \mathsf{F}\,\varphi$ iff there exists a finite path π and a function W such that for all run $\pi\rho'$, W is a witness for $\pi\rho'$ and k, $|\pi| \leq (k+1)|\varphi|(|\mathsf{S}|+1)$, $\max(\mathsf{W}) < |\pi|$ and $\pi[0]$ is reachable from $\rho[0]$ and $\pi[|\pi|-1]$.*

This statement is very close to Prop. 17, the only difference is the last condition, that is $\pi[0]$ is reachable from $\rho[0]$ and $\pi[|\pi|-1]$, so that the finite path π can be repeated infinitely often, as part of a lasso $\pi_0(\pi\pi_1)^\omega$ for some finite paths π_0 and π_1.

Proof scheme. If $(\rho, k) \not\models \mathsf{F}\,\varphi$, then for every suffix of ρ we can apply Prop. 17 to get a short witness of $(\rho[i..], k) \not\models \varphi$. Note that these paths π and mappings W are all bounded in size by the same bounds on k and $|\varphi|$, so that there are finitely many of them. Eventually, we must visit the same state at two positions i and j far enough apart to enforce that the witness path π for $(\rho[i..], k) \not\models \varphi$ ends before j. Thus, we get $\pi[0]$ is reachable from $\rho[0]$ and $\pi[|\pi|-1]$.

The converse direction of the proof is straight-forward, as we can deduce from such a lasso-shaped witness run that $(\rho[i..], k) \not\models \varphi$ for infinitely many positions i. This implies $(\rho, k) \not\models \mathsf{F}\,\varphi$ by prefix-independence. ◀

The proof of Thm. 26 is then obtained from Prop. 27. More precisely, we use a variation of Algorithm 1, that guesses π and W and adds the extra reachability checks of Prop. 27 over the end-points of π. Notice that this extra step is polynomial and therefore is not detrimental to membership in coNP.

4.3 Fair model checking

Going back to the bigger picture, we recall that in the non-prompt setting a polynomial procedure is obtained for the Muller fragment under fairness. This is based on exploiting the prefix-independence property on the one side, and the propensity of almost all runs to maximise the strongly connected sets of states they visit infinitely often on the other side. As such, only runs visiting bottom SCCs (maximal strongly connected sets of states almost always reached by runs in probabilistic systems) are relevant under fairness.

The intuition is then that every such SCC is either entirely winning for a prefix-independent $\mathcal{L}(\mathsf{F}^\infty)$ objective φ, regardless of what initial state is picked, or entirely losing. Moreover, checking this fact for a given bottom SCC and a given formula is straightforward, as φ satisfaction is entirely determined by $\mathsf{Inf}(\rho)$ on a given run ρ, and $\mathsf{Inf}(\rho)$ is almost always equal to the full bottom SCC by fairness.

For a $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}_\mathsf{P}^\infty)\right)$ formula, we follow the same recipe: bottom SCCs will also be eventually reached, and will either be entirely winning for a given φ, or entirely losing for φ. The main difference brought by promptness is that we can no longer simply rely on fairness to make sure that every state of the SCC is visited infinitely often: in order to guarantee *prompt* visits, we must assume the fair coin to be adversarial and make sure that every state of the SCC *will be visited shortly* no matter what. This results in a kind of attractor computation to check if a bottom SCC is winning or not, that can then be paired with the algorithm from $\mathcal{L}(\mathsf{F}^\infty)$. This is the key to proving that indeed, prefix independence makes the fair model checking problem for $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}_\mathsf{P}^\infty)\right)$ polynomial.

▶ **Theorem 28.** *The fair model checking problem for* $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}_\mathsf{P}^\infty)\right)$ *is in* PTIME.

5 Discussion and conclusion

In this section we discuss our results, some immediate corollaries and future work. The first point we address might seem technical at a first glance, but we believe that it is worth mentioning; it concerns the quantification of the bound k used in the semantics of pLTL.

Strong against weak semantics. Recall that pLTL formulas are evaluated with respect to a uniform bound k, *i.e.* for a system to be a model to some formula, all its runs are evaluated using the same bound k. Let us refer to this semantics as *strong*. One could want to relax this semantics and define the following semantics: A system $\mathcal{S}$ satisfies a formula φ if for every run ρ there exists a bound k such that $(\rho, k) \models \varphi$. Let us call this semantics *weak*. Indeed one could argue that the strong semantics is too conservative and that a system designer might be interested in a more permissive behavior. This raises the following questions: *are both semantics equivalent?* Obviously, the strong semantics always implies the weak one. However, the converse does not hold. To separate these two semantics, consider the system depicted on the left of Figure 6, and the formula $\varphi = \mathsf{F}_\mathsf{P}^\infty\,\mathsf{A} \vee \mathsf{F}_\mathsf{P}^\infty\,\mathsf{B}$. This system satisfies the formula φ with respect to the weak semantics but not the strong one. This raises in turn another natural question: *Are these two semantics equivalent in some fragment?* We answer positively to this question by noticing that they collapse for formulas in $\mathsf{F}(\mathcal{L}^+(\mathsf{F}_\mathsf{P}^\infty))$. This follows from the following observation:

The weak and strong semantics are equivalent for formulas in $\varphi \in \mathcal{L}(\mathsf{F}_\mathsf{P}^\infty)$ *if they are evaluated over a strongly connected LTS.*

Indeed, if for every k there is a counter-example run ρ_k so that $(\rho_k, k) \not\models \varphi$, then by our small witness property (Prop. 17) we can get paths $\pi_1, \pi_2, \ldots$ that are sufficient to falsify φ for $k = 1, 2, \ldots$ respectively. It is then sufficient to chain them one after the other – by exploiting the strongly connected assumption – to construct a single path ρ that falsifies every bound k, as required by the weak semantics. As a consequence, both semantics are equivalent in our prefix independent fragment $\mathsf{F}\left(\mathcal{L}^+(\mathsf{F}_\mathsf{P}^\infty)\right)$, where satisfaction for a run ρ is determined by the long-term behavior of ρ, *i.e.* the suffixes that stay confined to the strongly connected set $\mathsf{Inf}(\rho)$.

Probabilistic model checking. The last point we want to discuss is the complexity of quantitative verification for the fragment $\mathsf{F}(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P}))$. Once again, our fragment behaves well. We argue that one can compute the *satisfaction probability* of a system with respect to a formula in $\mathsf{F}(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P}))$ in polynomial time. This nice property follows from the following *zero-one-law* for the bottom SCCs of an LTS $\mathcal{S}$:

Let B be a bottom SCC and φ be a formula in $\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P})$, then either almost all the runs of B satisfy φ or almost no run of B does.

This is a consequence of Coro. 20, as any finite path that witnesses the falsification of φ in a bottom SCC will eventually be visited with probability 1. Now to conclude, one has to notice that computing the probability of reaching B can be done in PTIME, and use the prefix independence of $\mathsf{F}(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P}))$.

Future work. Finally, we hint at some future directions. The fragment $\mathsf{F}(\mathcal{L}^+(\mathsf{F}^\infty_\mathsf{P}))$ turned out to behave well in the presence of fairness, leading to a tractable model checking procedure. While, this is an improvement over the coNP procedure for the universal problem, we are still missing a matching lower bound to separate the two formally. A more ambitious perspective is the study of the controller synthesis problem induced by these fragments in 2-player games.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

2 Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour. Life is random, time is not: Markov decision processes with window objectives. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: https://lmcs.episciences.org/6975.

3 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016*, volume 226 of *EPTCS*, pages 135–148, 2016. doi:10.4204/EPTCS.226.10.

4 Damien Busatto-Gaston, Youssouf Oualhadj, Léo Tible, and Daniele Varacca. Fairness and promptness in muller formulas, 2024. arXiv:2204.13215.

5 Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Reasoning about actions and planning in LTL action theories. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams, editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 593–602. Morgan Kaufmann, 2002.

6 Krishnendu Chatterjee. Concurrent games with tail objectives. *Theor. Comput. Sci.*, 388(1-3):181–198, 2007. doi:10.1016/j.tcs.2007.07.047.

7 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

8 Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986. doi:10.1145/5397.5399.

9 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995. doi:10.1145/210332.210339.

10 E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987. doi:10.1016/0167-6423(87)90036-0.

11 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* MIT Press, Cambridge, MA, USA, 2003.

12 Giuseppe De Giacomo and Moshe Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In Susanne Biundo and Maria Fox, editors, *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, volume 1809 of *Lecture Notes in Computer Science*, pages 226–238. Springer, 1999. doi:10.1007/10720246_18.

13 Hugo Gimbert and Florian Horn. Solving simple stochastic tail games. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 847–862. SIAM, 2010. doi:10.1137/1.9781611973075.69.

14 Hugo Gimbert and Edon Kelmendi. Two-player perfect-information shift-invariant submixing stochastic games are half-positional. *CoRR*, abs/1401.6575, 2014. arXiv:1401.6575.

15 Orna Kupferman, Nir Piterman, and Moshe Vardi. From liveness to promptness. *Formal Methods in System Design*, 34, April 2009. doi:10.1007/s10703-009-0067-z.

16 Anca Muscholl and Igor Walukiewicz. An np-complete fragment of LTL. *Int. J. Found. Comput. Sci.*, 16(4):743–753, 2005. doi:10.1142/S0129054105003261.

17 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57, 1977. doi:10.1109/SFCS.1977.32.

18 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.

19 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. doi:10.1007/BFb0035790.

20 Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982. doi:10.1007/3-540-11494-7_22.

21 Matthias Schmalz, Hagen Völzer, and Daniele Varacca. Model checking almost all paths can be less expensive than checking all paths. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science, 27th International Conference, New Delhi, India, December 12-14, 2007, Proceedings*, volume 4855 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2007. doi:10.1007/978-3-540-77050-3_44.

22 A. Sistla and Edmund Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32:733–749, July 1985. doi:10.1145/800070.802189.

23 Hagen Völzer and Daniele Varacca. Defining fairness in reactive and concurrent systems. *J. ACM*, 59(3):13:1–13:37, 2012. doi:10.1145/2220357.2220360.

A Multi-pumpings and their properties

▶ **Proposition 13.** *Given an lts $\mathcal{S}$, a run $\rho \in \mathsf{Runs}$, a multi-pumping ρ' of ρ, a formula $\varphi \in \mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$ and $k \geq 0$, if $(\rho, k) \not\models \varphi$, then $(\rho', k) \not\models \varphi$.*

In order to prove Prop. 13. we establish some structural properties of multi-pumpings.

A multi-pumping of a run can be seen as pumping multiple loops of the same run, but with an ordering of the loops first. Indeed, loops are pumped in order, one after the other, and not one inside another. This implies that merging two multi-pumpings is quite easy. It suffices to respect the order of the loops, as illustrated in Figure 7. This is formalise by the following lemma.

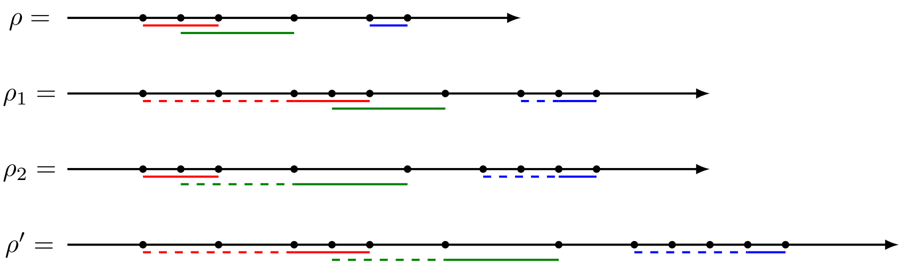


Figure 7 Illustration of the merging of two multi-pumpings, as per the construction of Lem. 29.

▶ **Lemma 29.** *Given an LTS $\mathcal{S}$ and a run $\rho \in$ Runs, let ρ_1 and ρ_2 be two multi-pumpings of ρ. Then, there exists a multi-pumping ρ' of ρ such that ρ' is also a multi-pumping of ρ_1 and of ρ_2.*

The idea behind the proof is to do all the pumpings of ρ_1 and ρ_2 in the right order in order to merge the two pumpings.

Proof. As ρ_1 and ρ_2 are two multi-pumping of ρ, there are n_1 and n_2 such that ρ_1 is a n_1-pumping of ρ and ρ_2 is a n_2-pumping of ρ. Let us prove the result by induction over $n_1 + n_2$.

- $n_1 + n_2 = 0$: In that case $n_1 = n_2 = 0$ and $\rho_1 = \rho_2 = \rho$ are trivial pumpings. Therefore the result trivially holds.
- Assume now that $n_1 + n_2 \geq 1$. If $n_1 = 0$, then $\rho_1 = \rho$ and the result trivially holds. The same is true if $n_2 = 0$. Now assume that $n_1 > 0$ and $n_2 > 0$. Then by definition $\exists i_1 < j_1, \rho[i_1] = \rho[j_1]$ such that there exists a $(n_1 - 1)$-pumping $\tilde{\rho}_1$ of $\rho[i_1..]$ and $\rho_1 = \rho[..i_1]\rho[i_1..j_1]^{l_1-1}\tilde{\rho}_1$, where $l_1 > 0$. In the same way, $\exists i_2 < j_2, \rho[i_2] = \rho[j_2]$ such that there exists a $(n_2 - 1)$-pumping $\tilde{\rho}_2$ of $\rho[i_2..]$ and $\rho_2 = \rho[..i_2]\rho[i_2..j_2]^{l_2-1}\tilde{\rho}_2$, where $l_2 > 0$. Without loss of generality assume that $i_1 \leq i_2$. By construction, as $i_1 \leq i_2$, $\rho_2[i_1..]$ is a n_2-pumping of $\rho[i_1..]$, and $\tilde{\rho}_1$ is by definition a $(n_1 - 1)$-pumping of $\rho[i_1..]$. By induction hypothesis, there exists a multi-pumping $\tilde{\rho}'$ of $\rho[i_1..]$ that is also a multi-pumping of $\rho_2[i_1..]$ and $\tilde{\rho}_1$. Then, by construction, the run $\rho' = \rho[..i_1]\rho[i_1..j_1]^{l_1-1}\tilde{\rho}'$ is a multi-pumping of both ρ_1 and ρ_2. ◀

Note that the merging ρ' is not unique. For example, if $i_1 = i_2$, one can do the pumping in any order and the resulting pumping will work, inducing two different multi-pumping. In the following, when we refer to the merging of two multi-pumping, we fix an arbitrary one.

▶ **Lemma 30.** *Given an LTS $\mathcal{S}$, a run $\rho \in$ Runs and ρ' a multi-pumping of ρ, let i_0 be the first index of a repeating state in ρ, that is, there exists a $0 \leq i' < i_0$ such that $\rho[i'] = \rho[i_0]$ and for all $0 \leq j < j' < i_0$, $\rho[j] \neq \rho[j']$. Then, $\rho'[0..i_0 + 1] = \rho[0..i_0 + 1]$.*

The intuitive idea behind this lemma is that a pumping can not modify the run before the first loop.

▶ **Lemma 31.** *Given an LTS $\mathcal{S}$, a finite path π, and two runs $\pi\rho \in$ Runs, and $\pi\rho' \in$ Runs such that ρ' is a multi-pumping of ρ, then $\pi\rho'$ is a multi-pumping of $\pi\rho$.*

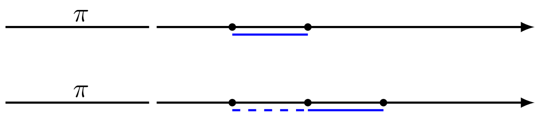


Figure 8 Graphical representation of Lem. 31.

▶ **Lemma 32.** *Given an lts $\mathcal{S}$, a run $\rho \in$ Runs, and a pumping ρ' of ρ such that $\rho' = \rho[0..i_1]\rho[i_1..i_2]^l\rho[i_2..]$, then one of the following holds for any position $j \geq 0$:*

- *$\rho'[j..]$ is a pumping of $\rho[j..]$.*
- *There is $i_1 \leq j' \leq j$ such that $\rho'[j..] = \rho[j'..]$.*
- *There is a cycle π and $i_1 \leq j' \leq j$ such that $\rho'[j..] = \pi\rho[j'..]$.*

The idea of this lemma is that every position of a pumping can be matched to a position of the original run. Depending on whether the position is before the added loop, in the middle of it or afterwards, the structure is a bit different, but the matching still holds. Mainly, this lemma is a tool used in other proofs. As the proof is rather technical, we will explain the idea of this lemma with some figures.

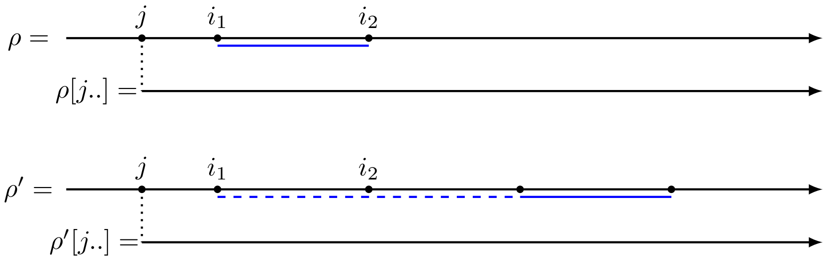


Figure 9 First case : $j < i_1$.

Consider the run ρ and the position j describe in Figure 9, and the pumping ρ' of ρ where two loop $i_1 \cdots i_2$ are added. Clearly, $\rho[..j] = \rho'[..j]$ and thus $\rho'[j..]$ is a pumping of $\rho[j..]$.

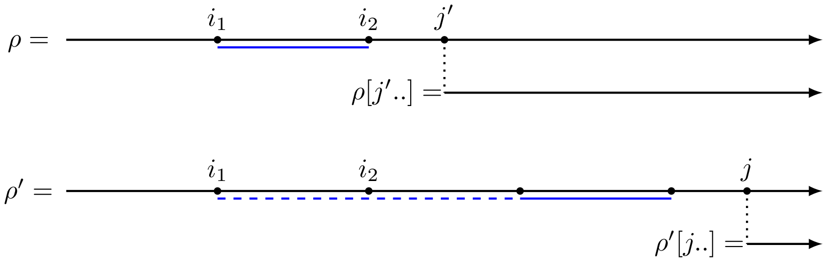


Figure 10 Second case : $j > i_2 + (i_2 - i_1)(l - 1)$.

Now, consider the same run and same pumping, but consider that $j > i_2 + (i_2 - i_1)(l - 1)$, as represented in Figure 10. Here, two loops are added, therefore $l = 2$. As j is after every added loops in ρ', one can find a j' such that $\rho'[j..] = \rho[j'..]$. Note that if j is in the last loop $i_1 \cdots i_2$, then $j > i_2 + (i_2 - i_1)(l - 1)$ and this idea still works.

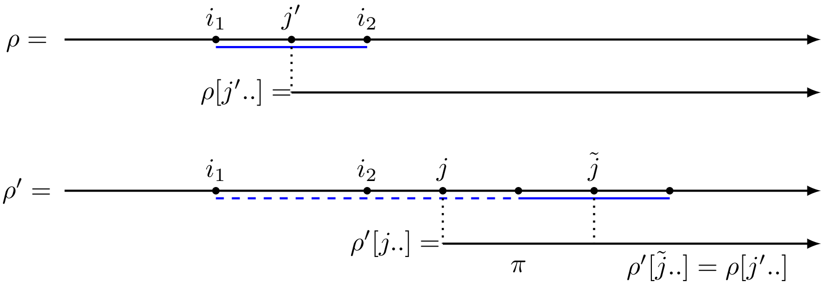


Figure 11 Third case : $i_1 \leq j \leq i_2 + (i_2 - i_1)(l - 1)$.

The last case is the most technical. Consider the same run ρ and same pumping ρ', but with $i_1 \leq j \leq i_2 + (i_2 - i_1)(l - 1)$, that is, j is somewhere inside an added loop, as described in Figure 11. Then, $\rho'[j..]$ may not be a pumping of any $\rho[j'..]$, as $\rho'[j..]$ starts with a bit of the loop, then has some complete occurrences of the loop, and then continues in the same way as ρ. Yet, one can consider the last occurrence of $\rho'[j]$, that is, the same vertex but in the last loop, let say $\rho'[\tilde{j}]$. Then, $\rho'[j..]$ can be described as a loop $\pi = \rho'[j..\tilde{j}]$ followed by $\rho'[\tilde{j}..]$. Then, by the same argument as in the second case, there exists a j' such that $\rho[j'..] = \rho'[\tilde{j}..]$.

Those ideas are formalised in the following proof.

Proof of Lem. 32. Let us look at every possibility.

Firstly, assume that $j < i_1$. Then, $\rho'[j..] = \rho[j..i_1]\rho[i_1..i_2]^l\rho[i_2..]$. By definition, $\rho[j..]$ is a pumping of $\rho[j..]$.

Secondly, assume that $j > i_2 + (i_2 - i_1)(l - 1)$. Then, let $j' = j - (i_2 - i_1)(l - 1)$. We have $j' > i_2$, and thus by construction $\rho'[j..] = \rho[j'..]$.

Thirdly, we have $i_1 \leq j \leq i_2 + (i_2 - i_1)(l - 1)$. There is a $0 \leq l' \leq l - 2$ such that

$$i_1 + (i_2 - i_1)l' \leq j \leq i_1 + (i_2 - i_1)(l' + 1).$$

Let $j' = j - (i_2 - i_1)l'$, and let us write

$$\rho' = \rho[0..i_1]\rho[i_1..i_2]^{l'}\rho[i_1..j']\rho[j'..i_2]\rho[i_1..i_2]^{l-l'-2}\rho[i_1..j']\rho[j'..i_2]\rho[i_2..]\,.$$

Note that $\rho'[j..] = \rho[j'..i_2]\rho[i_1..i_2]^{l-l'-2}\rho[i_1..j']\rho[j'..i_2]\rho[i_2..]$.
Let $\pi = \rho[j'..i_2]\rho[i_1..i_2]^{l-l'-2}\rho[i_1..j']$. Note that $\pi[0] = \rho[j'] \in \mathsf{Succ}(\pi[|\pi| - 1])$ and π is a cycle. Therefore, $\rho'[j..] = \pi\rho[j'..]$. ◀

Now, we show some properties of pumpings in regard to the satisfaction of formulas. Those properties are necessary in order to use pumpings and still keep satisfaction, or invalidation, of the formula.

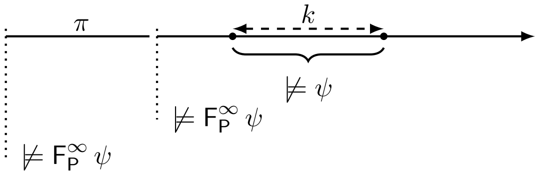

π
k
$\not\models \psi$
$\not\models \mathsf{F}^\infty_\mathsf{P}\,\psi$
$\not\models \mathsf{F}^\infty_\mathsf{P}\,\psi$

Figure 12 Visual representation of Lem. 33.

▶ **Lemma 33.** *Given an LTS $\mathcal{S}$, a finite path π, a run $\rho = \pi\rho' \in \mathsf{Runs}$, a bound $k \geq 0$, and a formula $\varphi \in \mathcal{L}(\mathsf{F}_\mathsf{P}^\infty)$, if $\rho'[0] = \pi[0]$ and $(\rho', k) \not\models \varphi$, then $(\rho, k) \not\models \varphi$.*

Intuitively, the idea is that adding a finite prefix to a run will not remove any faulty window, just postpone them, as can be seen in Figure 12. For state formulas, we have to check that the initial state stays the same.

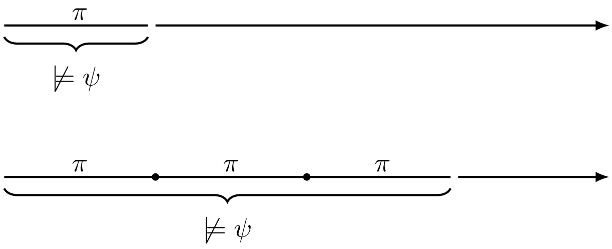


Figure 13 Visual representation of Lem. 34.

▶ **Lemma 34.** *Given an LTS $\mathcal{S}$, a finite path π, a run $\rho = \pi\rho' \in \mathsf{Runs}$ such that $\pi[0] = \rho'[0]$, a bound $k \geq 0$, a formula φ, and the run $\rho_l = \pi^l\rho'$ for $l > 0$, if for all $0 \leq j \leq |\pi|$, $(\rho[j..], k) \not\models \varphi$, then for all $0 \leq j' \leq l \times |\pi|$, $(\rho_l[j..], k) \not\models \varphi$.*

This lemma is quite technical and used only as a tool in other proofs. The idea is that if a loop does not satisfy a formula, no iteration of that loop can satisfy the formula, as represented in Figure 13. It is a direct corollary of Lem. 33, as formally shown in the following proof.

Proof of Lem. 34. First, notice that $\pi[0] = \rho'[0]$ ensures that $\pi^l\rho'$ is indeed a valid run in the system. Now, consider $0 \leq j' \leq l \times |\pi|$, and $j = j'$ modulo $|\pi|$. As $\rho_l[((l-1) \times |\pi|)..] = \rho$ by definition of ρ_l, we have that $\rho_l[((l-1) \times |\pi| + j)..] = \rho[j..]$. Moreover, by definition, $0 \leq j \leq |\pi|$, and $\rho_l[j'] = \rho[j]$. By noting that $\rho_l[j'..] = \rho_l[j'..((l-1) \times |\pi| + j)]\rho[j..]$ and $(\rho[j..], k) \not\models \varphi$, Lem. 33 is enough to conclude. ◀

We have now presented every tool needed to prove Prop. 13

Proof of Prop. 13. By definition, a multi-pumping is a n-pumping for some $n \geq 0$, that is, a multi-pumping is a succession of a finite number of pumpings. Therefore, to show the result for ρ' a pumping of ρ is enough to conclude with a structural induction.

Let us prove by structural induction over φ that if ρ' is a pumping of ρ, then $(\rho, k) \not\models \varphi$ implies that $(\rho', k) \not\models \varphi$.

- $\varphi = \alpha$ or $\varphi = \neg\alpha$: Then the result trivially holds as $\rho[0] = \rho'[0]$ by construction.
- $\varphi = \psi_1 \vee \psi_2$: By definition, $(\rho, k) \not\models \varphi$ if $(\rho, k) \not\models \psi_1$ and $(\rho, k) \not\models \psi_2$. By induction hypothesis, this implies that $(\rho', k) \not\models \psi_1$ and $(\rho', k) \not\models \psi_2$, and therefore $(\rho', k) \not\models \varphi$.
- $\varphi = \psi_1 \wedge \psi_2$: By definition, $(\rho, k) \not\models \varphi$ if $(\rho, k) \not\models \psi_1$ or $(\rho, k) \not\models \psi_2$. Without loss of generality assume that $(\rho, k) \not\models \psi_1$. By induction hypothesis, this implies that $(\rho', k) \not\models \psi_1$ and therefore $(\rho', k) \not\models \varphi$.
- $\varphi = \mathsf{F}_\mathsf{P}^\infty \psi$: By definition, there exists a position $i \geq 0$ such that for all $i \leq j \leq i + k$, $\rho[j..] \not\models \psi$. Write $\rho' = \rho[0..i_1]\rho[i_1..i_2]^l\rho[i_2..]$ with $l > 0$ and $i_1 < i_2$. Firstly, if $i_1 < i$ then for all $i \leq j \leq i+k$, $\rho'[(j+(i_2-i_1)(l-1))..] = \rho[j..]$. Therefore for all $i+(i_2-i_1)(l-1) \leq j \leq i + (i_2 - i_1)(l-1) + k$, $\rho'[j..] \not\models \psi$. Secondly, we have $i \leq i_1$. Let us show that $\forall i \leq j \leq i + k$, $\rho'[j..] \not\models \psi$. By Lem. 32, there are three cases.

- If $\rho'[j..]$ is a pumping of $\rho[j..]$, then by induction hypothesis $\rho'[j..] \not\models \psi$.
- If there exists $i_1 \leq j' \leq j$ such that $\rho'[j..] = \rho[j'..]$ then, as $i \leq i_1$, we have that $i \leq j' \leq i + k$, so that $\rho[j'..] \not\models \psi$, and thus $\rho'[j..] \not\models \psi$.
- If there exists a cycle π and $i_1 \leq j' \leq j$ such that $\rho'[j..] = \pi\rho[j'..]$, then once again $i \leq j' \leq i + k$, so that $\rho[j'..] \not\models \psi$, and by Lem. 33 $\rho'[j..] \not\models \psi$. ◀

▶ **Lemma 14.** *Let $\mathcal{S} = \langle \mathsf{S}, s_{\mathsf{init}}, \mathsf{T}, \mathsf{lbl} \colon \mathsf{S} \to 2^{\mathsf{AP}} \rangle$ be an LTS, $\varphi \in \mathcal{L}(\mathsf{F}^{\infty}_{\mathsf{P}})$, and let $N = |S| + 1$. If there is $\rho_N \in \mathsf{Runs}_{\mathsf{init}}$ such that $(\rho_N, N) \not\models \varphi$ then for all $k \geq N$, there is a multi-pumping ρ_k of ρ_N such that $(\rho_k, k) \not\models \varphi$.*

Proof. By structural induction over φ.

- $\varphi = \alpha$ or $\varphi = \neg\alpha$: The result trivially holds as the bound is irrelevant to the satisfaction of φ, and ρ_N is a multi-pumping of itself.
- $\varphi = \psi_1 \vee \psi_2$: By definition, $(\rho_N, N) \not\models \varphi$ if $(\rho_N, N) \not\models \psi_1$ and $(\rho_N, N) \not\models \psi_2$. By induction hypothesis, for all $k \geq N$, there are two multi-pumpings ρ^1_k and ρ^2_k of ρ_N such that $(\rho^1_k, k) \not\models \psi_1$ and $(\rho^2_k, k) \not\models \psi_2$. By lemma 29, there exists a multi-pumping ρ_k of ρ_N such that ρ_k is also a multi-pumping of ρ^1_k and ρ^2_k. Then, by lemma 13, $(\rho_k, k) \not\models \psi_1$ and $(\rho_k, k) \not\models \psi_2$. Therefore, $(\rho_k, k) \not\models \psi_1 \vee \psi_2$.
- $\varphi = \psi_1 \wedge \psi_2$: By definition, $(\rho_N, N) \not\models \varphi$ if $(\rho_N, N) \not\models \psi_1$ or $(\rho_N, N) \not\models \psi_2$. Without loss of generality, suppose that $(\rho_N, N) \not\models \psi_1$. By induction hypothesis, for all $k \geq N$, there exists a multi-pumping ρ_k of ρ_N such that $(\rho_k, k) \not\models \psi_1$. Then, by definition, $(\rho_k, k) \not\models \psi_1 \wedge \psi_2$.
- $\varphi = \mathsf{F}^{\infty}_{\mathsf{P}}\, \psi$: By definition, if $(\rho_N, N) \not\models \varphi$ then $\exists i, \forall 0 \leq j \leq N, (\rho_N[(i + j)..], N) \not\models \psi$. As $N > |\mathsf{S}|$, there are two positions i_1, i_2 with $i \leq i_1 < i_2 \leq i + N$ such that $\rho_N[i_1] = \rho_N[i_2]$. Moreover, one can assume w.l.o.g. that i_1 and i_2 are chosen such that the finite path $\rho_N[i_1..i_2]$ is such that for each $i_1 \leq j_1 < j_2 \leq i_2 - 1$, $\rho_N[j_1] \neq \rho_N[j_2]$. For each $i_1 \leq j \leq i_2$, we have $(\rho_N[j..], N) \not\models \psi$. For each $i_1 \leq j \leq i_2$, we can apply the induction hypothesis to obtain a multi-pumping ρ^j_k of $\rho_N[j..]$ such that $(\rho^j_k, k) \not\models \psi$. By Lem. 31, for each $i_1 \leq j \leq i_2$, we have that $\rho_N[i_1..j]\rho^j_k$ is a multi-pumping of $\rho_N[i_1..]$. By applying Lem. 29 multiple times, there exists a run $\tilde{\rho}_k$ such that for each $i_1 \leq j \leq i_2$, $\tilde{\rho}_k$ is a multi-pumping of $\rho_N[i_1..j]\rho^j_k$. Moreover, observe that it is not possible to have a loop between i_1 and i_2 in ρ. Therefore, by construction, for each $i_1 \leq j \leq i_2$, we have that $\tilde{\rho}_k[(j - i_1)..]$ is a multi-pumping of ρ^j_k and therefore, by Prop. 13, $(\tilde{\rho}_k[(j - i_1)..], k) \not\models \psi$. Consider the run $\tilde{\rho}'_k = \tilde{\rho}_k[0..(i_2 - i_1)]^k \tilde{\rho}_k[(i_2 - i_1)..]$. By Lem. 34, for all $0 \leq j \leq k \times (i_2 - i_1)$, $(\tilde{\rho}'_k[j..], k) \not\models \psi$. Consider the run $\rho_k = \rho_N[0..i_1]]\tilde{\rho}_k$. Then, as $i_2 > i_1$, $k \times (i_2 - i_1) > k$, for each $0 \leq j \leq k$, $\rho_k[(i_1 + j)..] = \tilde{\rho}'_k[j..]$, and therefore $(\rho_k[(i_1 + j)..], k) \not\models \psi$. By definition, $(\rho_k, k) \not\models \mathsf{F}^{\infty}_{\mathsf{P}}\, \psi$. ◀

Learning Partitions Using Rank Queries

Deeparnab Chakrabarty
Dartmouth College, Hanover, NH, USA

Hang Liao
Dartmouth College, Hanover, NH, USA

Abstract

We consider the problem of learning an unknown partition of an n element universe using rank queries. Such queries take as input a subset of the universe and return the number of parts of the partition it intersects. We give a simple $O(n)$-query, efficient, deterministic algorithm for this problem. We also generalize to give an $O(n + k \log r)$-rank query algorithm for a general partition matroid where k is the number of parts and r is the rank of the matroid.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Query Complexity, Hypergraph Learning, Matroids

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.16

Related Version *Extended Version*: `https://arxiv.org/abs/2409.13092`

Funding *Deeparnab Chakrabarty*: Supported by NSF grants 2041920 and 2402571.
Hang Liao: Supported by NSF grant 2041920.

1 Introduction

Let V be a universe of n elements and suppose there is an *unknown* partition $\mathcal{P} = (P_1, \ldots, P_k)$ that we want to learn. We have an oracle called rank that takes as input any subset $S \subseteq V$ and returns the number of different parts this subset intersects. More precisely $\mathsf{rank}(S) := \sum_{i=1}^{k} \min(|S \cap P_i|, 1)$. How many queries suffice to learn $\mathcal{P}$?

This natural question is a special case of the problem of *learning hypergraphs* under the *additive query* model initially studied by [28]. In this problem, we have an unknown hypergraph on a vertex set V, and an additive query $\mathsf{add}(T)$ on a subset $T \subseteq V$ returns the *number* of hyperedges completely contained in T. Our unknown partition $\mathcal{P}$ is a special hypergraph whose k hyperedges are disjoint (that is, it is a hypermatching); and for any subset S we observe that $\mathsf{rank}(S)$ is precisely $k - \mathsf{add}(V \setminus S)$. And so, the problem we study can be rephrased as in how few additive queries can a hypermatching be learnt. Although hypermatchings may feel too specialized, the now mature literature on *graph* learning (cf. [22, 17, 15, 16, 23]) began with understanding the case of graph matchings (cf. [28, 4, 3]).

The problem we study is also a special case of a *matroid learning* problem with access to rank oracle queries. Matroids are set systems, whose elements are called *independent* sets, that are defined using certain axioms and these are fundamental objects in combinatorial optimization. It is well known that a partition $\mathcal{P}$ induces the following simple partition matroid: a subset $I \subseteq V$ is independent if $|I \cap P_i| \leq 1$ for all i. The rank of a matroid is the cardinality of the largest independent set of the matroid, and more generally, the rank of subset S is the cardinality of the largest independent set that is a subset of S. A moment's notice shows that for the simple partition matroid this is precisely $\mathsf{rank}(S)$ which explains the name we give to our oracle. So, our problem we study asks: in how few rank queries can a simple partition matroid be learnt?

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 16; pp. 16:1–16:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

It is rather straightforward[1] to learn the partition using $O(n \log k)$ queries as follows. First, one learns a representative from each part with n-queries; given a set of already learned representatives R, a vertex v is in a new unrepresented part if and only if $\mathsf{rank}(R \cup v) > \mathsf{rank}(R)$. After learning the k representatives, we can learn every other vertex's part by performing a binary search style algorithm. Can one do better? It is instructive to note that the algorithm sketched above does not really utilize the full power of the query model we have. In particular, it would have sufficed if the query took a subset S and said YES if every element in S was in a different part, or NO otherwise. Using the matroid language, an *independence oracle* suffices which only states if a set S is independent or not. Now, an independence oracle answer gives at most 1 bit of information; on the other hand, there are roughly k^n different partitions possible with $\leq k$ parts. Therefore, via an information theoretic argument $\Omega(n \log k)$ independence queries are *necessary* to learn the partition. In contrast, the *rank* oracle gives the *number* of different parts hit by a subset; this is an integer in $\{0, 1, \ldots, k\}$ and the information theoretic argument only proves an $\Omega(n)$ lower bound on the number of queries. This naturally leads to the question: can an $O(n)$-query algorithm exist? The main result of this paper is a simple affirmative answer to this question.

▶ **Theorem 1.** *There is a deterministic, constructive algorithm that solves unknown partition learning problem using $O(n)$* rank *queries.*

▶ Remark. We have not optimized the constant in front of n. We think it can be made less than 10 but don't believe can be made less than 4 using our methods. The best lower bound one can prove using the above information theory argument is n. Figuring out the precise coefficient is left as an open question.

We also consider the generalization of learning a *general* partition matroid using rank queries. In this case, along with the unknown partition $\mathcal{P}$, we have unknown positive integers $r_1, \ldots, r_k$ associated with each part, where $1 \leq r_i < |P_i|$. A subset I is independent in this matroid if $|I \cap P_i| \leq r_i$, for all $1 \leq i \leq k$. When all $r_i = 1$, we have the simple partition matroid. The rank query corresponds to $\mathsf{rank}(S) := \sum_{i=1}^{k} \min(|S \cap P_i|, r_i)$. In how few rank queries can we learn a general partition matroid?

As in the simple partition matroid case, one can get an $O(n \log k)$-query algorithm using just an independent set oracle via a more delicate[2] binary-search-style algorithm. Can we obtain $O(n)$ query algorithm with rank queries? We believe the answer should be yes and take the following first step.

▶ **Theorem 2.** *There is a deterministic, constructive algorithm that learns a general partition matroid using $O(n + k \log r)$* rank *queries where $r := \mathsf{rank}(V) = \sum_i r_i$.*

▶ Remark. When the number of parts $k \leq n/\log n$, we thus get an $O(n)$-rank query algorithm. However, when $k = \Omega(n)$ we don't do any better than just with independence queries.

Perspective

Our motivation to look at the problem arose from trying to understand the *connectivity* question in hypergraphs using CUT queries. Although, as mentioned earlier, graph learning under query models has been extensively studied, over the last few years, multiple works such as [36, 27, 29, 8, 6, 20, 30] have focused on trying to understand if fewer queries can

[1] Something that can be given in an undergraduate algorithms course when teaching binary search.
[2] Maybe a challenging exercise in the aforementioned algorithms course; see Section 3 for this algorithm.

lead to understanding *properties* of graphs. Of particular interest is understanding the *connectivity*/finding spanning forest of a graph using CUT queries. A CUT query takes a subset of vertices as input and returns the number/weight of the cut edges crossing the subset. While graph learning can take $\tilde{\Theta}(m)$ cut queries, a spanning forest of an undirected graph, unweighted or weighted, can be constructed[3] in $O(n)$ queries (see [6, 30]). Can such results be generalized[4] to hypergraphs? To us, the easiest case of a hypergraph was the hypermatching whose only spanning forest is the hypergraph itself. It is not too hard to see that CUT queries and rank queries are intimately related. Formally, after n cut queries, any rank query can be simulated with 2 cut queries. The interesting open question is: *can the connectivity question of an arbitrary hypergraph be solved in $O(n)$ queries?*

The other related problem is *matroid intersection.* Given rank/independence oracle to two matroids over the same universe, the matroid intersection problem asks to find the largest common independent set. It is a classic result in combinatorial optimization due to [26] that this can be solved in polynomially many independence oracle queries. The current state-of-the-art is that $\tilde{O}(n^{1.5})$-rank queries suffice (see [19]) and $\tilde{O}(n^{7/4})$-independence oracle queries suffice (see [10]). On the other hand, no *super-linear* lower bounds are known for rank-queries, and only recently, [11] proved an $\Omega(n \log n)$-lower bound for independence queries. The big open question is: *can matroid intersection be solved in $O(n)$ rank-queries, or can a $\omega(n)$-lower bound be proved*?

As noted earlier, if we wish to obtain an $o(n \log k)$-query algorithm, we must exploit the fact that rank-queries output "more" than the independence oracle queries. Our second motivation in writing this paper is to showcase how the techniques that arise from *coin weighing* problems a la [18, 31] exploit this "more". In the basic coin-weighing problem, one is asked to recover an unknown Boolean vector x with the ability to query any subset S and obtain $\sum_{i \in S} x_i$ (a sum-query). The aforementioned papers showed how to do this making roughly $2n/\log_2 n$ sum-queries. [13] generalized this to learn a Boolean vector with at most d ones in roughly $\frac{2d \log_2 n}{\log_2 d}$ queries. In a different application, [28] showed how to use the coin-weighing result to learn a hidden perfect matching in a bipartite graph using $2n$ CUT queries. These form the backbone of our algorithms. Having said that, there are some big differences between sum-queries and rank-queries since the latter is not "linear" and this underlies the difficulties we've faced in generalizing Theorem 2 to obtain a $O(n)$-query algorithm to the general partition matroid case.

1.1 Related works

There is a vast literature on combinatorial search [1, 25], and we restrict ourselves to the works that are related the most. As mentioned above, our problem can be thought of as learning a *hypermatching* using additive/cut queries. (Hyper)-graph reconstruction questions have been widely studied in the last two decades. A significant body of work has been dedicated to reconstructing graphs using queries, as evidenced by the works of (cf. [28, 4, 3, 35, 23, 34, 15, 17, 22]). These efforts encompass various types of graphs, including unweighted graphs, graphs with positive weights, and graphs with non-zero edge weights, using CUT queries. This has culminated in a result of [22] gives an polynomial time, randomized $O(\frac{m \log n}{\log m})$-query algorithm for learning graphs on n nodes and m edges with non-zero edge weights, and this query complexity is information theoretic optimal. Concurrently, there has

[3] using a randomized Las Vegas algorithm which makes $O(n)$ queries in expectation

[4] At first glance even a polynomial query algorithm may not be clear; a little thought can lead to an $O(n \log n)$ query algorithm.

been ongoing research on recovering specific structures within graphs without necessarily reconstructing the entire graph, such as figuring its connectivity (see [28, 20, 30]). [5] started the research on learning a hypergraph using edge-detecting queries, that is, whether the input set contains a hyperedge or not; they described algorithms for r-uniform hypergraphs (every hyperedge has exactly r vertices) but the dependence on r was exponential. [14] considered the *additive* model where one gets the number of edge (this was mentioned in the Introduction above) and proved existence of algorithms to learn rank d hypergraphs (every hyperedge has at most d vertices) for constant d using $O_d(m \log(n^d/m)/\log m)$-queries; the dependence on d is exponential. [9] considered high-rank but low-degree hypergraphs, including hypermatchings. The focus was on edge-detecting queries (indicator whether additive query is zero or non-zero), and they gave $O(n\text{polylog}n)$-query algorithms which were also "low depth", that is, with few rounds of adaptivity.

Our problem is also related to the problem of recovering a *clustering* with active queries (see [7, 33, 37, 2, 21, 12, 32]). The setting is the same: there is a universe of n points which we assume is clustered into k unknown parts. The query model, however, is often quite different and much more restrictive usually constraining queries to asking whether a pair or a constant number of elements are in the same cluster/part or not. Such a study was initiated in the works of [24, 7, 33] which prove an $\Omega(nk)$ lower bound, and then provide better upper bounds with extra assumptions. The above-cited works continue on this line.

2 O(n) Query Deterministic Algorithm

Throughout the rest of the paper, unless otherwise mentioned all logarithm base is 2. We begin with an overview of the algorithm. We maintain a collection $\mathcal{J}$ of disjoint independent sets; recall that a subset is independent if it contains at most one element from each part. Initially, $\mathcal{J}$ is the collection of n independent sets each of which is a single element. Let J denote the union of all these independent sets, and so, initially $J = V$. The algorithm will modify this collection $\mathcal{J}$ in iterations, removing some elements from J while doing so. Anytime such an element e is removed, we maintain a map $\mathsf{rep}(e)$ to an element in the current J with the property that e and $\mathsf{rep}(e)$ are in the same partition in $\mathcal{P}$. We will call two elements in the same parts "friends", and so, $\mathsf{rep}(e)$ is e's friend.

The key routine in the algorithm is a merge operation over independent sets. Given two independent sets I_1, I_2, define the set of *common nodes* $\mathsf{com}(I_1, I_2) := \{v_1 \in I_1 : \exists v_2 \in I_2, P_i, \{v_1, v_2\} \subseteq P_i\}$ to be the subset of nodes in I_1 which have a friend in I_2. Note that this friend needs to be unique since I_2 is independent. The MERGE operation takes two independent sets I_1 and I_2 and then (a) finds the set $\mathsf{com}(I_1, I_2)$, (b) for each $e \in \mathsf{com}(I_1, I_2)$, finds its unique neighbor $\mathsf{rep}(e) \in \mathsf{com}(I_2, I_1)$, and (c) returns $\mathsf{com}(I_1, I_2), \mathsf{com}(I_2, I_1)$, and $I_3 := I_1 + I_2 - \mathsf{com}(I_1, I_2)$. See Figure 1 for an illustration.

Given the MERGE routine, the algorithm is very simple: while there exists two independent sets I_1 and I_2 of comparable size (within factor 2), merge them and replace I_1 and I_2 with I_3 returned by the merge. This may remove some elements from J, and in particular this is $\mathsf{com}(I_1, I_2)$, but all these elements will have $\mathsf{rep}(e)$ pointing to their friends who are still in J. When the algorithm can't do this anymore, there must be at most $\ell = \lceil \log k \rceil$ independent sets remaining in $\mathcal{J}$. These can be sequentially merged in any order to get one single independent set J, indeed a basis, in $\mathcal{J}$. To find the partition, consider the directed graph on V where we add the edge $(e, \mathsf{rep}(e))$ for all $e \in V \setminus J$; note this forms a collection of directed in-trees rooted at vertices in J, and the connected components are precisely the parts that

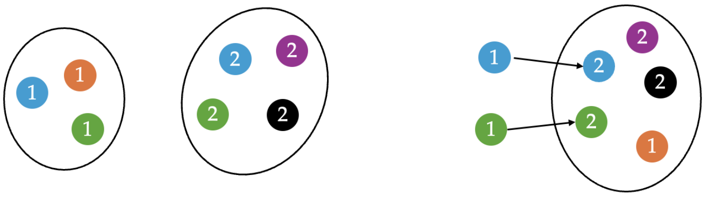


Figure 1 After we merge I_1, I_2 on the left, we get I_3 and a mapping from $\mathsf{com}(I_1, I_2)$ to $\mathsf{com}(I_2, I_1)$.

we desire. In what follows we show how to implement Merge using existing results from coin-weighing and graph reconstruction, and then argue why the total number of rank queries made by our algorithm is $O(n)$.

2.1 Definitions

We review or introduce several definitions for completeness.

▶ **Definition 3** (add query). *An* add *query on an unweighted graph $G = (V, E)$: given $S \subseteq V$, obtain $|\{e \in E : e \in S \times S\}|$.*

▶ **Definition 4** (sum query). *A* sum *query on a boolean vector $x \in \{0, 1\}^N$: given $S \subseteq [N]$, obtain $\sum_{i \in S} x_i$.*

▶ **Definition 5** (com of two sets). *Given two independent sets I_1, I_2. The set of* common nodes $\mathsf{com}(I_1, I_2) := \{v_1 \in I_1 : \exists v_2 \in I_2, P_i, \{v_1, v_2\} \subseteq P_i\}$ *is the subset of nodes in I_1 which have a friend in I_2.*

▶ **Definition 6** (rep(e) of a node). *We maintain a map* rep *with the property that a node e and* rep(e) *(representative of e) are in the same partition in $\mathcal{P}$.* rep *keeps track of the learned partition by mapping the learned node to its friend who is still in J.*

2.2 Merging Independent Sets

We begin by introducing some vector/graph reconstruction algorithms from the literature.

▶ **Lemma 7** ([13]). *Let $x \in \{0, 1\}^N$ be an unknown Boolean vector with* sum*-query access. If x has d ones, then there is a polynomial time, adaptive, deterministic algorithm to reconstruct x which makes $O(d \log(N/d)/\log d)$* sum *queries.*

▶ **Lemma 8** (Paraphrasing Theorem 4 & Section 4.3 [28]). *A bipartite graph $G = (V, W, E)$ with $|V| = |W| = m$ where E forms a perfect matching can be learnt in $O(m)$* add *queries.*

Now we are ready to describe Merge whose properties are encapsulated in the following lemma.

▶ **Lemma 9.** *Let I_1, I_2 be two independent sets and let $k_1 = |I_1|$ and $k_2 = |I_2|$. Suppose $d = |\mathsf{com}(I_1, I_2)| = |\mathsf{com}(I_2, I_1)|$. The procedure* Merge *is an adaptive deterministic polynomial time algorithm which returns $I_3 = I_1 + I_2 - \mathsf{com}(I_1, I_2)$ and $\mathsf{rep}(e) \in \mathsf{com}(I_2, I_1)$ for all $e \in \mathsf{com}(I_1, I_2)$. The procedure makes $O(\frac{d \log(\max(k_1, k_2)/d)}{\log d})$* rank *queries.*

Proof. Given I_1 and I_2, define the Boolean vector $\mathbf{x} := \mathbf{x}_{(I_1,I_2)} \in \{0,1\}^{k_1}$ where $\mathbf{x}_e = 1$ if and only if $e \in \mathsf{com}(I_1, I_2)$. We note that a sum query can be simulated on $\mathbf{x}$ using a single rank query. This is due to the observation that for all $S \subseteq I_1$, $\sum_{e \in S} \mathbf{x}_e = |S| + |I_2| - \mathsf{rank}(S \cup I_2)$. This is because the RHS precisely counts the number of parts of S that are already present in I_2, or $\mathsf{com}(I_1, I_2) \cap S$. Therefore, we can apply Lemma 7 to learn $\mathsf{com}(I_1, I_2)$ in $O(\frac{d \log(k_1/d)}{\log d})$ many rank queries. Similarly, we can get $\mathsf{com}(I_2, I_1)$ in $O(\frac{d \log(k_2/d)}{\log d})$ queries. Note that the above doesn't give us the friends for $e \in \mathsf{com}(I_1, I_2)$ in $\mathsf{com}(I_2, I_1)$. This pairing can be found as follows. For simplicity, let's use $X := \mathsf{com}(I_1, I_2)$ and $Y := \mathsf{com}(I_2, I_1)$. Consider the bipartite graph $G = (X, Y, E)$ where $e \in X$ has an edge to $f \in Y$ if and only if f is e's friend. So, G is a perfect matching whose edges are yet unknown. We can now use Lemma 8 to find them. To see why this can be done, note that we can simulate the add query because for any $S \subseteq X \cup Y$, simply because $\mathsf{add}(S) = |S| - \mathsf{rank}(S)$ This is because any edge (e, f) with both endpoints in S are precisely the pairs which are counted once in $\mathsf{rank}(S)$ but twice in $|S|$. Thus, finding this matching takes $O(d)$ rank queries. ◀

Algorithm 1 Merging Independent Sets.

```
procedure MERGE($I_1, I_2$):
    ▷ Input: Two independent sets
    ▷ Output: $\mathsf{com}(I_1, I_2)$ and $\mathsf{rep}(e) \in \mathsf{com}(I_2, I_1)$ for $e \in \mathsf{com}(I_1, I_2)$.
    Learn $\mathsf{com}(I_1, I_2)$ and $\mathsf{com}(I_2, I_1)$ as described above using $O(d \log(\max(k_1, k_2))/\log d)$ rank queries.
    Learn $\mathsf{rep}(e) \in \mathsf{com}(I_2, I_1)$ for $e \in \mathsf{com}(I_1, I_2)$ as described above in $O(d)$ rank queries.
    $I_3 \leftarrow I_1 \cup I_2 - \mathsf{com}(I_1, I_2)$.  ▷ Note that $I_3$ is independent and $\mathsf{rep}(e) \in I_3$ for all $e \in \mathsf{com}(I_1, I_2)$.
    return $(I_3, \mathsf{rep})$
```

2.3 The algorithm and analysis

We give the pseudocode of the algorithm in Algorithm 2. We now claim that the algorithm makes $O(n)$ queries. All the queries to rank occur in the calls to MERGE in line 7 or line 14. Let's take care of the second ones first since it's straightforward.

▷ Claim 10. The total number of rank queries made in MERGE calls in line 14 over the for-loop is $O(n)$.

Proof. There are $\ell = O(\log n)$ merges made; that is the only fact we will use. By Lemma 9, the tth MERGE would make at most $O(d_t \log n / \log d_t)$ many rank queries, where d_t is the size of $\mathsf{com}(I_t, I)$ at that time. All we care for is that $\sum_{t=1}^{\ell} d_t \leq n$. Now we observe (an explicit reference is Claim 3 of [20]) that if $\ell \leq C \log n$, then $\sum_{t=1}^{\ell} \frac{d_t}{\log d_t} = O(n/\log n)$. To see this, note that the contribution to this sum of all the d_t's which are $\leq \frac{n}{C \log^2 n}$ is at most $\frac{n\ell}{C \log^2 n} < n/\log n$. All the other d_t's have $\log d_t = \Omega(\log n)$ and so their contribution is $O(\sum_t d_t/\log n) = O(n/\log n)$. Altogether, we see that $O(\sum_{t=1}^{\ell} d_t \log n/\log d_t) = O(n)$. ◁

▷ Claim 11. The total number of rank queries made in MERGE calls in line 7 over the while-loop is $O(n)$.

Algorithm 2 Find Partition.

```
1:  procedure FINDPARTITION($V$, rank):
2:      ▷ Input: $n$ elements with rank query access to hidden partition $\mathcal{P}$.
3:      ▷ Output: the partition.
4:      Create $\mathcal{J} \leftarrow \{\{e_1\}, \{e_2\}, \ldots, \{e_n\}\}$; $J \leftarrow V$
5:      Create graph $G = (V, F)$ with $F \leftarrow \emptyset$. ▷ this will be used to find the parts
6:      while $\exists I_1, I_2 \in \mathcal{J} \;:\; |I_1|/|I_2| \in [1/2, 2]$ do:
7:          $(I_3, \mathsf{rep}(e)) \leftarrow$MERGE$(I_1, I_2)$.
8:          For all $e \in \mathsf{com}(I_1, I_2)$, add $(e, \mathsf{rep}(e))$ to the edge-set $F$.
9:          $\mathcal{J} \leftarrow \mathcal{J} - \{I_1, I_2\} + I_3$; $J \leftarrow J \setminus \mathsf{com}(I_1, I_2)$.
10:     ▷ At this point there can be at most $\lceil \log n \rceil$ elements in $\mathcal{J}$
11:     ▷ Merge all these sets in any order to get a single set. We provide one below.
12:     Let $\mathcal{J} = \{I_1, I_2, \ldots, I_\ell\}$ with $\ell \le \lceil \log n \rceil$; $I \leftarrow I_1$; $\mathcal{J} \leftarrow \mathcal{J} \setminus I_1$
13:     for $2 \le t \le \ell$ do:
14:         $(I_3, \mathsf{rep}(e)) \leftarrow$ MERGE$(I_t, I)$.
15:         For all $e \in \mathsf{com}(I_t, I)$, add $(e, \mathsf{rep}(e))$ to the edge-set $F$.
16:         $\mathcal{J} \leftarrow \mathcal{J} - \{I_t, I\} + I_3$; $I \leftarrow I_3$.
17:     ▷ At this point $\mathcal{J}$ has a single independent set $I$. Every element in $e \in V \setminus I$ has a
        single representative $\mathsf{rep}(e)$. So $G$ is a collection of directed in-trees with roots in $I$
18:     return Connected components of $G$.
```

Proof. To argue about the MERGE's in Line 7, we need to partition these into two classes. Note that all such merges take two independent sets I_1 and I_2 which are of similar size k_1 and k_2 respectively; without loss of generality, let $k_1 \le k_2 \le 2k_1$. Let $d := |\mathsf{com}(I_1, I_2)|$. We call a merge *thick* if $d \ge \sqrt{k_1}$ and *thin* otherwise. We argue about the thick and thin merges differently.

- Using Lemma 9, we see that a thick merge costs $O(d \log(\max(k_1, k_2))/\log d) = O(d)$ rank queries; we have used here that $k_2 \le 2k_1$ and $d \ge \sqrt{k_1}$. Thus, we can *charge* these rank queries to the d elements which *leave* J. Thus, the total number of rank queries made across all thick merges is $O(n)$.
- To argue about thin merges, we make a further definition. Let us say that an independent set I is in class t if $|I| \in [2^t, 2^{t+1})$, for $0 \le t \le \lfloor \log n \rfloor$. Fix such a t. A thin merge (I_1, I_2) is called a class t thin-merge if the smaller cardinality set is in class t. An element $e \in V$ participates in a class t thin-merge (I_1, I_2) if it is present in the smaller set. Observe that for a thin class t merge, the resulting independent set I_3 almost doubles in size; in particular, $|I_3| = |I_1| + |I_2| - |\mathsf{com}(I_1, I_2)| \ge 2^{t+1} - 2^{t/2}$. Using this one can argue that the same element cannot participate in more than *two* class t-thin merges; after two merges the set ceases to be class t. In particular, this means the number of thin class t merges is at most $2 \cdot n/2^t$, and each such merge, by Lemma 9, can be done with $O(d \log(2^{t+1})/\log d)$ many rank queries where $d = |\mathsf{com}(I_1, I_2)| < 2^{t/2}$. Since $d/\log d$ is an increasing function of d, we conclude that any class t thin-merge takes at most $O(2^{t/2} \log(2^{t+1})/\log(2^{t/2})) = O(2^{t/2})$ many rank queries. Therefore, the total number of rank queries made within thin merges is at most $\sum_{t=0}^{\log n} \frac{2n}{2^t} \cdot O(2^{t/2}) = O(n)$ ◀

The above two claims imply the proof of Theorem 1.

3 General Partition Matroids

We recall the problem. As before, the universe is V and there is a hidden partition $\mathcal{P} = (P_1, \ldots, P_k)$. Furthermore, there are integers $r_1, \ldots, r_k$ where $0 < r_i < |P_i|$.[5] This defines a partition matroid where a set I is independent if and only if $|I \cap P_i| \leq r_i$ for $1 \leq i \leq k$. The rank-oracle for this matroid is the following $\mathsf{rank}(S) = \sum_{i=1}^{k} \min(|S \cap P_i|, r_i)$.
We will prove the following theorem in this section.

▶ **Theorem 2.** *There is a deterministic, constructive algorithm that learns a general partition matroid using $O(n + k \log r)$ rank queries where $r := \mathsf{rank}(V) = \sum_i r_i$.*

Our proof technique will be a *reduction* to the simple partition matroid setting of Section 2. Before we get there, let's first begin with a simple well-known observation.

▶ **Lemma 12.** *There is an $O(n)$ rank query algorithm that finds a basis B of a partition matroid.*

Proof. This is standard and we give it below for completeness. Note that although described as a "for-loop", the above algorithm can be implemented in a single round of n many rank queries. ◀

Algorithm 3 Finding a Basis Using Rank Queries.

```
procedure FindBasis($V$, rank):
    ▷ Input: $n$ elements in $V$ with rank query access
    ▷ Output: A basis of $V$.
    $B \leftarrow \{\}$.
    for $v \in V$ do:
        if $\mathsf{rank}(B + \{v\}) = \mathsf{rank}(B) + 1$ then:
            $B \leftarrow B + \{v\}$.
    return $B$.
```

To obtain our reduction, what we need apart from this basis B are two sets of *representatives*. A subset $T \subseteq V$ is a set of representative if $|T \cap P_i| = 1$ for each $1 \leq i \leq k$. The reduction will need *two* representative sets: $T_1 \subseteq B$ and $T_2 \cap B = \emptyset$, and the subroutine FindRepresentatives(B) will find this. Furthermore, it will also return a map $\phi : T_1 \to T_2$ where for each $e \in T_1$, $\phi(e)$ belongs to the same part as e. The algorithm does so in $O(n + k \log r)$ queries; in fact, only *independence oracle* queries suffice. This is slightly non-trivial and we described this in Section 3.1. Let us now show how these representatives imply an $O(n)$-query algorithm to learn the partition $\mathcal{P}$ and the r_i's.

▷ Claim 13. Algorithm 4 returns the correct partition $\mathcal{P}$ and r_i's making $O(n)$ many rank queries.

Proof. The main idea is that the representatives allow us to simulate a simple partition matroid rank query on the basis and outside. More precisely, we claim that for any subset $S \subseteq B$, $\mathsf{rank}_1(S) = \sum_{i=1}^{k} \min(|S \cap P_i|, 1)$. If so, the correctness of Algorithm 4 follows

[5] Suppose we allow $r_i \geq |P_i|$. Let $M := \{i | r_i \geq |P_i|\}$. Now $\mathsf{rank}(S) = \sum_{i \in M} |S \cap P_i| + \sum_{i \notin M} \min(|S \cap P_i|, r_i)$. This means we get no information for partitions with index in M. To see this, we pick $x_1 \in P_{i_1}, x_2 \in P_{i_2}$ with $i_1, i_2 \in M$ and $i_1 \neq i_2$. If we swap x_1 with x_2 in every set we give to the rank-oracle, the answer it returns is the same. Thus no rank query algorithm can tell x_1, x_2 apart.

Algorithm 4 Using Representatives to Learn Partition.

```
procedure LearnMatroidWithReps($V, \mathsf{rank}, B, T_1, T_2, \phi : T_1 \to T_2$):
    ▷ Input: $n$ elements in $V$ with rank query; basis $B$, set of representatives $T_1 \subseteq B$, $T_2 \cap B = \emptyset$, $\phi(t)$ is a friend of $t$.
    ▷ Output: the partition $\mathcal{P}$.
    For any subset $S \subseteq B$, define $\mathsf{rank}_1(S) := \mathsf{rank}(B - S + T_2) - \mathsf{rank}(B - S)$.
    $\mathcal{P}_1 \leftarrow$ FindPartition($B, \mathsf{rank}_1$) ▷ Takes $O(|B|)$ queries.
    For each $1 \le i \le k$, $r_i \leftarrow |B \cap P_i^{(1)}|$ where $\mathcal{P}_1 = (P_1^{(1)}, \ldots, P_k^{(1)})$.
    For any subset $S \subseteq V \setminus B$, define $\mathsf{rank}_2(S) := \mathsf{rank}(B + S - T_1) - \mathsf{rank}(B - T_1)$.
    $\mathcal{P}_2 \leftarrow$ FindPartition($V \setminus B, \mathsf{rank}_2$) ▷ Takes $O(|V \setminus B|)$ queries.
    Use $\phi$ to merge $\mathcal{P}_1$ and $\mathcal{P}_2$ into $\mathcal{P}$: for $t \in T_1$, merge the part in $\mathcal{P}_1$ containing $T_1$ with the part in $\mathcal{P}_2$ containing $\phi(t)$.
    return $(\mathcal{P}, \{r_i\}_{i=1}^k)$
```

from Theorem 1. Indeed, $\mathsf{rank}(B-S+T_2) - \mathsf{rank}(B-S)$ gives $+1$ for each part where $B-S$ loses at least one element to which the unique element of T_2 contributes. Similarly, one argues that for any $S \subseteq V \setminus B$, $\mathsf{rank}_2(S) = \sum_{i=1}^{k} \min(|S \cap P_i|, 1)$. This is also for a similar reason; $B - T_1$ loses exactly one element from each part and so $\mathsf{rank}(B + S - T_1) - \mathsf{rank}(B - T_1)$ counts the parts that S intersects at least once. See Figure 2 for an illustration. ◁

3.1 Finding Representatives via Binary Search

We now describe a procedure which takes a basis B of the partition matroid, and finds two subsets of representatives $T_1 \subseteq B$ and $T_2 \cap B = \emptyset$. The idea behind is a delicate binary search. Fix some $e \in V \backslash B$ and without loss of generality, say $e \in P_1$. We now show how to find one element in $B \cap P_1$ in $O(\log r)$ queries. The way to do it is by halving B to $X_1 \sqcup X_2$ and keep the half with at least P_1 element in it as "search half". This can be checked by seeing whether $\mathsf{rank}(X_1 + \{e\}) = \mathsf{rank}(X_1)$: if so then X_1 contains all element of P_1 and this is the half we stick with; otherwise X_2 contains some P_1 element and takes precedence. The other half is now added to the new "test half". We keep searching in our search set until it has exactly 1 class P_1 element left. For instance, say X_2 is further divided to X_{21} and X_{22}. The next query would be to check if $\mathsf{rank}((X_1 + \{e\}) + X_{21}) = \mathsf{rank}(X_1 + X_{21})$. If so, then $X_1 + X_{21}$ contains all class P_1 elements, and so will make X_{21} our new "search set"; otherwise, we continue on X_{22}. We describe the pseudocode in detail in Algorithm 5.

▷ Claim 14. Algorithm 5 returns (T_1, T_2, ϕ) correctly and makes $O(n + k \log r)$ many rank queries.

Proof. The proof is by induction: we claim that T_1, T_2 contains at most one element from each part and the size $|T_1| = |T_2|$ equals the number of parts spanned by the elements seen by the outer for-loop. And furthermore, the ϕ-relation is correct. This is obviously true before anything occurs, and consider the for-loop for and element e. Now suppose the if-statement in line 6 is *not* true; that is, say $\mathsf{rank}(B - T_1 + e) > \mathsf{rank}(B - T_1)$. This would mean that e contains a friend in T_1; the only way the rank could increase is if e filled the "hole" in the part which has exactly one element missing in $B - T_1$. We discard this e. On the other hand if the if-statement holds, then we will discover a new part in B and thus by inductive hypothesis, in $V \setminus B$. We therefore add e to T_2.

Now consider the invariant in line 10. If that indeed holds true, then when the while-loop terminates, and it does so with $|X| = 1$, the single element $x \in X$ must be in the same part as e. Thus, adding $x \in T_1$ and setting $\phi(x) = e$ is the correct thing to do. To see that the

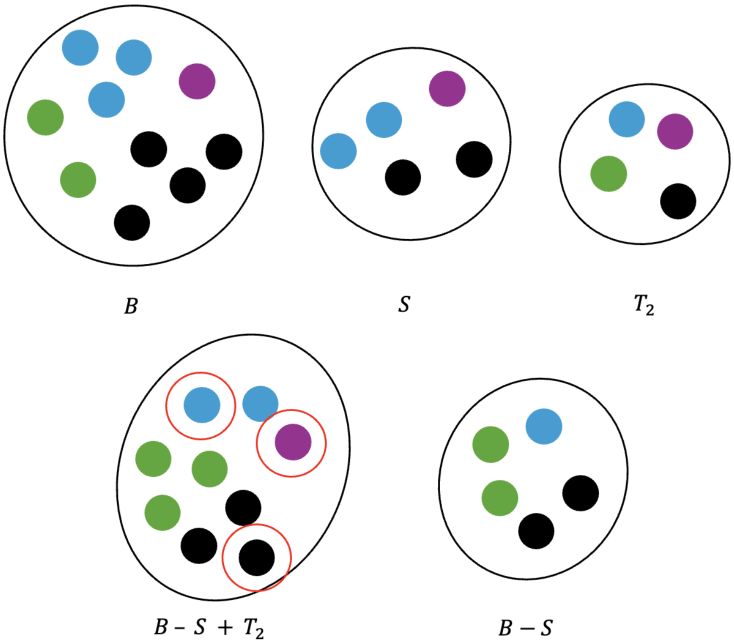


Figure 2 Illustration of how we simulate a simple partition matroid rank query inside of a basis with representatives outside. We have a basis with b_is equal to 1 (purple nodes), 2 (green), 3 (blue) and 4 (black) respectively. $\mathsf{rank}(B) = 10$, $\mathsf{rank}(B - S) = 5$. Note $|B - S + T_2| = 10$, yet $\mathsf{rank}(B - S + T_2) = 9$ because the number of green nodes is capped at 2. $\mathsf{rank}(B - S + T_2) - \mathsf{rank}(B - S) = 3$ simulate a simple partition matroid rank query for S. The circled nodes correspond to the 3 partitions included in S.

invariant in line 10 holds, we include the invariant in line 11. This is readily checked in both the "then" and "else" case of the forthcoming if-statement. If line 13 holds true, then akin to the argument above, $Y + X_1$ contains all friends of e in B. So, we focus our search on X_1 since it contains at least one friend of e because, by invariant, X contained at least one friend of e. So setting X to X_1 keeps the invariant satisfied. On the other hand, if line 13 doesn't hold true, then X_2 must contain at least one friend of e. And so setting X to X_2 keeps the invariant fulfilled.

To find the number of queries is simple. First notice that only line 6 and 13 make any queries. And even then one of them is superfluous. More precisely, since $B - T_1$ and $Y + X_1$ are independent sets, their rank is $|B| - |T_1|$ and $|Y| + |X_1|$ respectively. We make $n - r$ queries in line 13. Of these at most k many satisfy the condition. Each of them leads to a binary-search style argument which takes at most $\lceil \log r \rceil$ many queries. ◁

▶ Remark. Note that line 6 and line 13 can be implemented using only independence oracle queries since they are really asking, respectively, if $B - T_1 + e$ and $Y + X_1 + e$ are independent or not; if the ranks are equal, they are not. This also implies an $O(n \log k)$ algorithm to learn the partition matroid using only independence oracle as alluded to in the Introduction. Let $|T_1| = |T_2| = k$. Once we have the representative sets $T_1 \subseteq B$ and $T_2 \cap B = \emptyset$, for any element $e \notin V \setminus B$, we can use a binary-search style argument on T_1 to find e's friend among T_1 in $O(\log k)$ many independence oracle queries. More precisely, we halve T_1 into (X, Y) and check if $B - X + e$ is independent or not. If it is, then X contains e's friend; otherwise, Y does. Similarly, for any $e \in B$, we can find e's friend in T_2 in $O(\log k)$ many independence oracle queries.

Algorithm 5 Finding Representatives.

```
procedure FindRepresentatives($V$, rank, $B$):
    ▷ Input: $n$ elements in $V$ with rank query; basis $B$
    ▷ Output: Sets of Representatives $T_1 \subseteq B$, $T_2 \cap B = \emptyset$ and map $\phi : T_1 \to T_2$.
    $T_1, T_2 \leftarrow \emptyset$.
    for $e \in V \setminus B$ do:
        if $\mathsf{rank}(B - T_1 + \{e\}) = \mathsf{rank}(B - T_1)$ then:▷ $e$ is an element with no friends in $T_1$ and $T_2$:
            $T_2 \leftarrow T_2 + e$.
            $X \leftarrow B$; $Y \leftarrow \emptyset$.
            while $|X| > 1$ do:
                ▷ Invariant: $X$ has at least one element in same part as $e$
                ▷ Invariant: $X \cup Y = B$
                $(X_1, X_2) \leftarrow$ arbitrary equipartition of $X$.
                if $\mathsf{rank}(Y + X_1 + e) = \mathsf{rank}(Y + X_1)$ then: ▷ $X_2$ contains no friends of $e$
                    $Y \leftarrow Y + X_2$;    $X \leftarrow X_1$.
                else: ▷ $X_2$ contains at least one friend of $e$
                    $Y \leftarrow Y + X_1$;    $X \leftarrow X_2$.
            ▷ $X$ is a singleton element of $B$; let $X = \{x\}$
            $T_1 \leftarrow T_1 + x$; Set $\phi(x) = e$.
    return $(T_1, T_2, \phi)$.
```

Algorithm 6 Learning a Partition Matroid.

```
procedure LearnPartition($V$, rank):
    ▷ Input: partition matroid on $n$ elements in $V$ with rank query
    ▷ Output: the partition $\mathcal{P}$ and $r_i$'s
    Learn a basis $B$ using FindBasis($V$, rank) a la Algorithm 3.
    $(T_1, T_2, \phi) \leftarrow$ FindRepresentatives ($V$, $B$, rank) a la Algorithm 5.
    return $(\mathcal{P}, \{r_i\}) \leftarrow$ LearnMatroidWithReps($V$, rank, $B$, $T_1$, $T_2$, $\phi$) a la Algorithm 4.
```

For completeness, we end the section by giving the pseudocode for the final algorithm in Algorithm 6. Lemma 12 establishes that Algorithm 6 makes n rank queries, Claim 14 establishes that Algorithm 6 makes $n + k \log r$ rank (in fact independence oracle) queries, and Claim 13 establishes that Algorithm 6 makes $O(n)$ rank queries. This completes the proof of Theorem 2.

4 Conclusion

In this paper we looked at the question of learning a hidden partition using rank queries which given a subset tells how many different parts it hits. We gave a simple but non-trivial, deterministic, and efficient algorithm which makes $O(n)$-rank queries. This is optimal up to constant factors. The main non-triviality arises in the use of techniques devised in coin-weighing algorithms a la [18, 31], and our work falls in a growing line of such results [28, 23, 14, 6, 20, 30] which explores the use of these techniques to solve combinatorial search problems.

The obvious question left open by our paper is whether there are $O(n)$ algorithms to learn general partition matroids especially when $k = \Theta(n)$. We have not been able to directly port the coin-weighing techniques to solve this problem even in the case of $r_i = 2$ for all i. The main technical challenge that the rank query, ultimately, is not a linear query and in Section 3 we could make it "behave linear" with the help of representatives. Our algorithm to find representatives, however, didn't utilize the "more information" given by rank-queries over independence oracle queries. Investigating this may lead to new algorithmic primitives. On the other hand, perhaps there is a $\omega(n)$ lower bound for this problem when $k = \Theta(n)$.

References

1 Martin Aigner. *Combinatorial search.* John Wiley & Sons, Inc., 1988.

2 Nir Ailon, Anup Bhattacharya, and Ragesh Jaiswal. Approximate correlation clustering using same-cluster queries. In *Proc., Latin American Theoretical Informatics Symposium*, pages 14–27, 2018. doi:10.1007/978-3-319-77404-6_2.

3 Noga Alon and Vera Asodi. Learning a hidden subgraph. *SIAM Journal on Discrete Mathematics (SIDMA)*, 18(4):697–712, 2005. doi:10.1137/S0895480103431071.

4 Noga Alon, Richard Beigel, Simon Kasif, Steven Rudich, and Benny Sudakov. Learning a hidden matching. *SIAM Journal on Computing (SICOMP)*, 33(2):487–501, 2004. doi:10.1137/S0097539702420139.

5 Dana Angluin and Jiang Chen. Learning a hidden hypergraph. In *Proc., Conf. on Learning Theory (COLT)*, pages 561–575. Springer, 2005. doi:10.1007/11503415_38.

6 Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut query algorithms with star contraction. *Proc., IEEE Conference on the Foundations of Computer Science (FOCS)*, 2022.

7 Hassan Ashtiani, Shrinu Kushagra, and Shai Ben-David. Clustering with same-cluster queries. *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 29, 2016.

8 Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *arXiv preprint arXiv:2109.02115*, 2021. arXiv:2109.02115.

9 Eric Balkanski, Oussama Hanguir, and Shatian Wang. Learning low degree hypergraphs. In *Proc., Conf. on Learning Theory (COLT)*, pages 419–420. PMLR, 2022. URL: https://proceedings.mlr.press/v178/balkanski22a.html.

10 Joakim Blikstad. Breaking O(nr) for Matroid Intersection. In *Proc., International Conference on Algorithms, Logic, and Programming (ICALP)*, pages 31:1–31:17, 2021. doi:10.4230/LIPICS.ICALP.2021.31.

11 Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, and Ta-Wei Tu. Fast algorithms via dynamic-oracle matroids. In *Proc., ACM Symposium on the Theory of Computing (STOC)*, pages 1229–1242, 2023. doi:10.1145/3564246.3585219.

12 Marco Bressan, Nicolò Cesa-Bianchi, Silvio Lattanzi, and Andrea Paudice. On margin-based cluster recovery with oracle queries. *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, pages 25231–25243, 2021.

13 Nader H. Bshouty. Optimal algorithms for the coin weighing problem with a spring scale. In *Proc., Conf. on Learning Theory (COLT)*, 2009.

14 Nader H. Bshouty and Hanna Mazzawi. Optimal Query Complexity for Reconstructing Hypergraphs. In *Proc., Symposium on the Theoretical Aspects of Computer Science (STACS)*, pages 143–154, 2010. doi:10.4230/LIPICS.STACS.2010.2496.

15 Nader H. Bshouty and Hanna Mazzawi. Algorithms for the coin weighing problems with the presence of noise. *Electron. Colloquium Comput. Complex.*, page 124, 2011. URL: https://eccc.weizmann.ac.il/report/2011/124, arXiv:TR11-124.

16 Nader H. Bshouty and Hanna Mazzawi. On parity check (0, 1)-matrix over $\mathbb{Z}_p$. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1383–1394, 2011.

17 Nader H. Bshouty and Hanna Mazzawi. Toward a deterministic polynomial time algorithm with optimal additive query complexity. *Theoretical Computer Science*, 417:23–35, 2012. doi:10.1016/J.TCS.2011.09.005.

18 David G. Cantor and W. H. Mills. Determination of a subset from certain combinatorial properties. *Canadian Journal of Mathematics*, 18:42–48, 1966.

19 Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. Faster matroid intersection. In *Proc., IEEE Conference on the Foundations of Computer Science (FOCS)*, pages 1146–1168, 2019. doi:10.1109/FOCS.2019.00072.

20 Deeparnab Chakrabarty and Hang Liao. A query algorithm for learning a spanning forest in weighted undirected graphs. In *Proc., International Conference on Algorithmic Learning Theory (ALT)*, pages 259–274, 2023. URL: https://proceedings.mlr.press/v201/chakrabarty23a.html.

21 I Eli Chien, Huozhi Zhou, and Pan Li. hs^2: Active learning over hypergraphs with pointwise and pairwise queries. In *Proc., International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2466–2475, 2019. URL: http://proceedings.mlr.press/v89/chien19a.html.

22 Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proc., Conf. on Learning Theory (COLT)*, volume 30, pages 797–818, 2013. URL: http://proceedings.mlr.press/v30/Choi13.html.

23 Sung-Soon Choi and Jeong Han Kim. Optimal query complexity bounds for finding graphs. *Artif. Intell.*, 174(9-10):551–569, 2010. doi:10.1016/J.ARTINT.2010.02.003.

24 Susan Davidson, Sanjeev Khanna, Tova Milo, and Sudeepa Roy. Top-k and clustering with noisy comparisons. *ACM Transactions on Database Systems (TODS)*, 39(4):1–39, 2014. doi:10.1145/2684066.

25 Dingzhu Du and Frank K Hwang. *Combinatorial group testing and its applications*, volume 12. World Scientific, 2000.

26 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial Structures and their Applications*, 18:69–87, 1970.

27 Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S Matthew Weinberg. New query lower bounds for submodular function minimization. *Proc., Innovations in Theoretical Computer Science (ITCS)*, page 64, 2020.

28 Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000. doi:10.1007/S004530010033.

29 Troy Lee, Miklos Santha, and Shengyu Zhang. Quantum algorithms for graph problems with cut queries. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 939–958, 2021. doi:10.1137/1.9781611976465.59.

30 Hang Liao and Deeparnab Chakrabarty. Learning spanning forests optimally in weighted undirected graphs with cut queries. In *Proc., International Conference on Algorithmic Learning Theory (ALT)*, 2024.

31 Bernt Lindström. On a combinatorial problem in number theory. *Canadian Mathematical Bulletin*, 8(4):477–490, 1965.

32 Xizhi Liu and Sayan Mukherjee. Tight query complexity bounds for learning graph partitions. In *Proc., Conf. on Learning Theory (COLT)*, pages 167–181. PMLR, 2022. URL: https://proceedings.mlr.press/v178/liu22a.html.

33 Arya Mazumdar and Barna Saha. Query complexity of clustering with side information. *Adv. in Neu. Inf. Proc. Sys. (NeurIPS)*, 2017.

34 Hanna Mazzawi. Optimally reconstructing weighted graphs using queries. In *Proc., ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 608–615, 2010. doi:10.1137/1.9781611973075.51.

35 Lev Reyzin and Nikhil Srivastava. Learning and verifying graphs using queries with a focus on edge counting. In *Proc., International Conference on Algorithmic Learning Theory (ALT)*, pages 285–297. Springer, 2007. doi:10.1007/978-3-540-75225-7_24.

36 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In *Proc., Innovations in Theoretical Computer Science (ITCS)*, pages 39:1–39:16, 2018. doi:10.4230/LIPICS.ITCS.2018.39.

37 Barna Saha and Sanjay Subramanian. Correlation clustering with same-cluster queries bounded by optimal cost. In *Proc., European Symposium on Algorithms*, pages 81:1–81:17, 2019. doi:10.4230/LIPICS.ESA.2019.81.

Two Results on LPT: A Near-Linear Time Algorithm and Parcel Delivery Using Drones

L. Sunil Chandran
Indian Institute of Science, Bengaluru, India

Rishikesh Gajjala
Indian Institute of Science, Bengaluru, India

Shravan Mehra
Indian Institute of Science, Bengaluru, India
University of Birmingham, UK

Saladi Rahul
Indian Institute of Science, Bengaluru, India

Abstract

The focus of this paper is to increase our understanding of the *Longest Processing Time First (LPT)* heuristic. LPT is a classical heuristic for the fundamental problem of uniform machine scheduling. For different machine speeds, LPT was first considered by Gonzalez et al. *(SIAM J. Comput. 6(1):155–166, 1977)*. Since then, extensive work has been done to improve the approximation factor of the LPT heuristic. However, all known implementations of the LPT heuristic take $O(mn)$ time, where m is the number of machines and n is the number of jobs. In this work, we come up with the first *near-linear time* implementation for LPT. Specifically, the running time is $O((n+m)(\log^2 m+\log n))$. Somewhat surprisingly, the result is obtained by mapping the problem to dynamic maintenance of lower envelope of lines, which has been well studied in the computational geometry community.

Our second contribution is to analyze the performance of LPT for the *Drones Warehouse Problem (DWP)*, which is a natural generalization of the uniform machine scheduling problem motivated by drone-based parcel delivery from a warehouse. In this problem, a warehouse has multiple drones and wants to deliver parcels to several customers. Each drone picks a parcel from the warehouse, delivers it, and returns to the warehouse (where it can also get charged). The speeds and battery lives of the drones could be different, and due to the limited battery life, each drone has a bounded range in which it can deliver parcels. The goal is to assign parcels to the drones so that the time taken to deliver all the parcels is minimized. We prove that the natural approach of solving this problem via the LPT heuristic has an approximation factor of ϕ, where $\phi \approx 1.62$ is the golden ratio.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Scheduling, Approximation algorithms, Fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.17

Related Version *Full Version*: https://arxiv.org/abs/2407.16323

Funding *Shravan Mehra*: This work was supported by SERB Core Research Grant (CRG/2022/006770): “Bridging Quantum Physics with Theoretical Computer Science and Graph Theory”.
Saladi Rahul: This work was supported by the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-230001).

1 LPT heuristic for uniform scheduling

Uniform machine scheduling with the minimum makespan objective is a fundamental problem. In this problem, we are given a set of n jobs (not necessarily of the same size) and a set of m machines (not necessarily of the same speeds). The goal is to schedule the n jobs on m

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 17; pp. 17:1–17:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

machines so that the time required to execute the schedule (makespan) is minimised. This is an NP-hard problem even for two machines [13] of the same speed, but polynomial time approximation schemes (PTASs) are known [20, 21].

A commonly studied heuristic for this problem is the *Longest Processing Time First* (LPT) heuristic. In the LPT heuristic, each job is assigned one by one, in non-increasing order of size, so that every job is assigned to a machine where it will be completed earliest (with ties being broken arbitrarily). Note that a machine might already have some jobs assigned to it and the execution of the current job happens only after finishing the already assigned jobs.

More intricate algorithms were designed in the literature to get a good approximation factor for uniform scheduling. For example, Horowitz and Sahni gave an exact dynamic programming algorithm which runs in exponential time [22]. When there are only two machines, they could build upon this algorithm to obtain a Polynomial-time approximation scheme (PTAS). Later, Hochbaum and Shmoys gave a PTAS when all the machines had identical speeds [20]. This was later extended to obtain a PTAS for the uniform scheduling problem (USP) [21].

However, the LPT algorithm remains popular in practice due to its simplicity and scalability (compared to the PTAS-type results which are relatively complicated and have expensive running time). As a result, there has been a long line of research on improving the approximation ratio of the LPT algorithm for USP. In two independent works, the ratio of the LPT algorithm was improved by Dobson [10] to $\frac{19}{12}$ and by Friesen [12] to $\frac{5}{3}$. Kovacs [28] further improved the approximation factor of the LPT algorithm to 1.58 and proved that the LPT algorithm cannot give an approximation factor better than 1.54.

2 First result: A near-linear time implementation of LPT

In spite of all the focus in the literature on adapting LPT to various settings of machine scheduling and analyzing its approximation factor, to the best of our knowledge, there has been no work on fast implementation of LPT. The known implementation of the LPT heuristic takes $O(mn)$ time (via the naive approach). In this work, we give the first near-linear time implementation of the LPT heuristic.

▶ **Theorem 1.** *There is an $O((n+m)(\log^2 m + \log n))$ time implementation of the LPT heuristic.*

For a set of given lines in 2- D, the *lower envelope* is the point-wise minimum of the lines (a more formal definition will follow later). In the dynamic maintenance of the lower envelope problem, in each step, a new line is added (or removed), as shown in Figure 2, and one has to maintain the lower envelope with a small update time. This has been well-studied in the computational geometry community. We establish a connection from LPT to the dynamic maintenance of the lower envelope of lines to prove Theorem 1 in Section 5.

3 Second result: LPT for the drones warehouse problem (DWP)

Our second contribution is to analyze the performance of LPT for the *Drones warehouse problem (DWP)* (formally defined in Section 3.1) which is a natural generalization of the uniform scheduling problem to drone-based parcel delivery from a warehouse.

Vehicle routing [14, 47] is a classical problem in which parcel deliveries are done by a single truck or a collection of trucks. Researchers have explored variations with different vehicle velocities [15] or scenarios where each parcel can only be delivered by a specific subset of vehicles [49]. With the advent of drones, a generalization of the vehicle routing problem has been studied in the literature in which a truck is carrying drones along with it. The drones pick up parcels from the truck, deliver the package and return to the truck (the truck might have now moved to a different location). This problem is more challenging than the traditional vehicle routing problem [8]. Several MIP (mixed integer programming) formulations and heuristics have been used to solve this problem [38, 2]. Theoretical guarantees have also been proved for this problem by Carlsson and Song using geometric methods [8].

The transition towards using only drones like in DWP (instead of trucks with drones) is evident in the gradual shift within the research community, as it is a more sustainable option for the future. Extensive efforts have been dedicated to developing algorithms for scheduling drones under various constraints [44]. Some MIP formulations were studied to minimize various objectives like the number of drones [18, 24] and bio-inspired algorithms were used to manage large fleets of drones [40]. As drones have a limited battery life, considerations for fuel stations were explored in [25]. Further, the drones might not all have the same features like speed and battery life and this was taken into account in [46]. For a comprehensive review of research in several related problems involving only drones, the reader can refer to the survey presented in [41]. There has also been a lot of work by the multi-agent community for scheduling [26], pathfinding [9, 19] and coordinating drones [39, 36, 6, 48].

In this paper, we consider the problem where a warehouse wants to use *drones* to deliver a large number of parcels to customers around it. Due to limited battery life, each drone has a restricted range around the warehouse in which it can deliver parcels. Also, depending on the manufacturer, the speed of each drone can vary. We will now formally define the DWP problem, state the result obtained by applying the LPT heuristic for DWP and provide a high-level overview of the analysis. We also provide a detailed literature review on the use of LPT for several other machine scheduling problems and connect it to our result on DWP in Section 4.

3.1 The drones warehouse problem (DWP)

For the sake of better readability, we deviate from the notation used in scheduling literature. The warehouse has a set of m drones $\mathcal{D} = \{D_1, D_2 \cdots D_m\}$ and a set of n parcels $\mathcal{P} = \{P_1, P_2 \cdots P_n\}$. The parcels in set $\mathcal{P}$ need to be delivered by drones in set $\mathcal{D}$. Each drone can pick up one parcel at a time from the warehouse, deliver it and return to the warehouse. For all $1 \leq j \leq n$, let the distance at which parcel P_j needs to be delivered be $\ell_j/2$. So each drone must travel a distance of ℓ_j in total to deliver the parcel P_j and come back to the warehouse.

Additionally, the drones have a limited battery life which can be different for each drone. Let d_i be the distance which drone D_i can travel, for all $1 \leq i \leq m$. Therefore, for a parcel P_j to be delivered by a drone D_i, it must be the case that $\ell_j \leq d_i$. The speed at which the drone D_i travels is v_i, for all $1 \leq i \leq m$. After each delivery, the drone recharges its battery in the warehouse, and for the sake of simplicity we assume the time taken to recharge is negligible. Our goal is to assign each parcel to a drone such that the time taken to execute the schedule is minimised.

More precisely, we define a *valid schedule* $f : \mathcal{D} \to 2^{\mathcal{P}}$ (power set of $\mathcal{P}$) such that the following properties hold

- Each parcel is assigned to exactly one drone, i.e., $f(D_i) \cap f(D_j) = \emptyset$ for all $i \neq j$ and $\bigcup_{D \in \mathcal{D}} f(D) = \mathcal{P}$.
- Each drone must be able to deliver the parcel assigned to it, i.e., $\ell_i \leq d_j$ for all $j \in [m]$ and $i : P_i \in f(D_j)$.

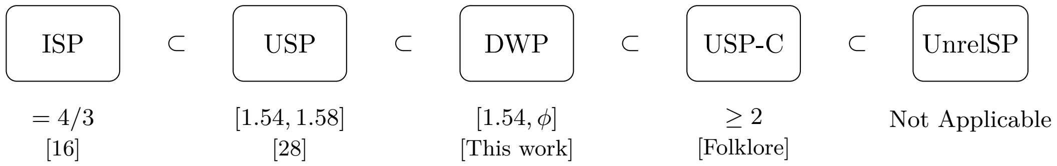


Figure 1 Landscape of the approximation ratio of the LPT heuristic for machine scheduling problems and our drone warehouse problem (DWP). The relation $A \subset B$ in the figure implies that A is a special case of B. Therefore, the approximation factor increases from left to right in the figure. The interval $[a, b]$ means that the approximation ratio of the LPT heuristic is at least a and at most b. As there is no total order among jobs in UnrelSP, LPT is not applicable for UnrelSP.

Our goal is to find a *valid schedule* such that $T(f)$ is minimised where

$$T(f) = \max_{j \in [m]} \sum_{i : P_i \in f(D_j)} \frac{\ell_i}{v_j}$$

We assume that there is at least one drone capable of delivering all parcels. Otherwise, there would be no valid solution (this can be checked in linear time).

3.2 Our results and techniques

We implement the LPT algorithm with additional battery life constraints to solve DWP in near-linear time. Our key contribution is to prove that this algorithm always returns a solution which has delivery time at most ϕ times the optimal solution, where $\phi \approx 1.62$ is the golden ratio. We summarize our results and compare them with previous work in Figure 1 (discussed in more detail in Section 4)

▶ **Theorem 2.** *There is a ϕ-approximation algorithm for the DWP problem where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio. The algorithm runs in $O((n+m)(\log^2 m + \log n))$ time, where m and n are the number of drones and parcels, respectively.*

At a high level, our proof is inspired by the analysis of the LPT algorithm for the uniform scheduling problem (USP) [28]. However, the constraint of battery life makes our problem significantly more challenging. As the total distance travelled by the drones is the same for any valid schedule, if some drone in the LPT algorithm travels more distance than its counterpart in the optimal assignment, then some other drone in the LPT algorithm must travel lesser distance than its counterpart in the optimal assignment as a compensation. However, if we assume that the approximation factor is greater than ϕ, we can create instances for which such compensation does not occur, which leads to a contradiction. The proof requires ideas such as (a) working with a minimal instance, (b) removing the parcel which is closest to the warehouse from the schedule, (c) classifying the parcels into three categories based on their distance and (d) truncating the parcel distances. We give the complete proof in Section 6.

4 Related work on machine scheduling

We will now give a detailed literature review of the use of LPT for machine scheduling and connect our problem DWP with the other variants.

4.1 Uniform machines scheduling problem (USP)

In the *uniform machines scheduling problem*, let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of n jobs of size $\{\ell_1, \ell_2, \ldots, \ell_n\}$ respectively and $\mathcal{D} = \{D_1, D_2, \ldots, D_m\}$ be a set of m machines of speed $\{v_1, v_2, \ldots, v_m\}$ respectively. Our goal is to schedule the n jobs to the m machines so that the completion time of the schedule is minimised. We will now formally define the problem.

We define a *schedule* as a function $f : \mathcal{D} \to 2^{\mathcal{P}}$ (power set of $\mathcal{P}$) where $f(D_i)$ represents the set of all jobs assigned to machine D_i. We call a *schedule* a *valid schedule* if each job is assigned to exactly one machine, i.e., f is a *valid schedule* if and only if $f(D_i) \cap f(D_j) = \emptyset$ for all $i \neq j$ and $\bigcup_{D \in \mathcal{D}} f(D) = \mathcal{P}$. Each schedule f has an associated completion time

$$T(f) = \max_{j \in [m]} \sum_{i : P_i \in f(D_j)} \frac{\ell_i}{v_j}$$

Our goal is to find a *valid schedule* f such that $T(f)$ is minimised.

4.2 ISP ⊂ USP ⊂ DWP

The special case of USP when all the machines have equal speed is called the *identical-machines scheduling problem (ISP)*. Therefore, we denote ISP ⊂ USP, where the notation $A \subset B$ means that A is a special case of B (See Figure 1) and therefore, a lower bound of A is also a lower bound for B and an upper bound of B is also an upper bound for A. Graham [16, 17] proved that the approximation factor of the LPT algorithm is $\frac{4}{3}$ for ISP. For USP, after a series of works, Kovacs [28] proved that the approximation factor of the LPT algorithm is at most 1.58 and at least 1.54. It is easy to see that USP is a special case of DWP (USP ⊂ DWP) when all battery lives are large enough to deliver all parcels. So, the approximation factor of LPT for DWP can not be better than 1.54. On the other hand, due to Theorem 2, we get that the LPT heuristic is a ϕ-approximation, which is one of the main contributions of this work.

4.3 DWP ⊂ USP-C ⊂ UnrelSP

A lot of work in the literature has been devoted to a generalization of USP, namely uniform scheduling problems with *processing constraints* (USP-C) [30]. We are given a set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ and a set of machines $\mathcal{M}$, where each job J_j has a set of machines $\mathcal{M}_j \subseteq \mathcal{M}$ to which they can be assigned to. Different structural restrictions of $\mathcal{M}_j$ lead to different models in USP-C, like the inclusive processing set [42, 35], nested processing set [37, 11], interval processing set [45, 23], and tree-hierarchical processing set restrictions [34, 33]. The goal is to assign each job to a machine to minimize the completion time.

DWP is also a special case of USP-C as each parcel can only be delivered by a subset of drones determined by its battery life. Therefore, DWP ⊂ USP-C. We emphasise that these two classes are strictly different (and DWP is also different from all known variants of USP-C, including nested intervals). Consider two machines and two jobs. Let the size of J_1, J_2 be $10, 10 + \epsilon$ respectively and the speeds of M_1, M_2 be $10, 10 + \epsilon$ respectively for $\epsilon > 0$. Moreover, assume that the job J_1 can only be done by M_2, but J_2 can be done by both M_1 and M_2. The LPT heuristic would take $\frac{20+\epsilon}{10+\epsilon}$ time, while the optimum assignment would take only $\frac{10+\epsilon}{10}$ time. This gives us an approximation ratio of 2 as ϵ approaches zero in this example (whether this ratio is optimal or not for the LPT heuristic on USP-C is an open question). We also note that this is not a valid lower bound instance for DWP as any drone which can do a job at a distance of $10 + \epsilon$ can also do the job at a distance of 10.

The unrelated Scheduling Problem (UnrelSP) is the scheduling problem in which the time taken by machine $D \in \mathcal{D}$ to complete job $P \in \mathcal{P}$ is determined by an arbitrary function $f : \mathcal{D} \times \mathcal{P} \to \mathbb{R}$. Note that USP-C $\subset$ UnrelSP and furthermore the LPT heuristic is not applicable for UnrelSP as there is no total order among the jobs to sort. This finishes the description of Figure 1.

4.4 Other algorithms and optimization measures

A PTAS for ISP was given by Hochbaum and Shmoys [20]. This was later extended to obtain a PTAS for USP [21]. Due to the seminal result of Lenstra, Shmoys and Tardos [29], there is an LP-based 2-approximation algorithm for UnrelSP. This is also the best known approximation algorithm for USP-C. For a special case of USP-C (including DWP) with nested intervals, there is 4/3 approximation algorithm [32]. For more results on USP-C, we refer the reader to the latest survey [31]. We emphasise that despite knowing PTASs and other algorithms with a better approximation ratio, the LPT heuristic remains popular in practice due to its simplicity and scalability. This motivated researchers from the algorithms and operations research community to extensively study it as described in Figure 1.

Instead of optimizing the time taken to complete all jobs (makespan), other objectives like the average completion time [7], weighted-average completion time [22] and monotonicity and truthfulness have also been studied [5, 27, 4, 3].

5 Near linear time implementation

5.1 Longest processing time first (LPT)

A classical approach for the Uniform Scheduling problem is using a greedy algorithm called LPT scheduling which gives a 1.58-approximate solution [28]. First, we sort the jobs $\mathcal{P}$ in decreasing order of their size and let $P_1, P_2, \ldots, P_n$ be the sorted sequence. Then we initialise T_j, the time taken by the jth machine to be zero for all $j \in [m]$. Now we assign the jobs sequentially from P_1 to P_n. We assign job P_i to a machine D_j for which the value of $T_j + (\ell_i/v_j)$ is minimum and we update the value T_j to $T_j + (\ell_i/v_j)$ for this specific j (see Algorithm 1).

Algorithm 1 LPT algorithm in $O(nm)$ time.

```
Input: List of jobs $\mathcal{P}$ and machines $\mathcal{D}$.
Output: Time required for LPT scheduling.
Sort $\mathcal{P}$ in non-increasing order of size.
Initialise $T_j = 0$ for all $j \in [m]$
for $i$ in $\{1 \ldots n\}$ do
    $\alpha = \arg\min_{j\in[m]} \left(T_j + \frac{\ell_i}{v_j}\right)$
    $T_\alpha \leftarrow T_\alpha + \frac{\ell_i}{v_\alpha}$
end for
return $\max_{j\in[m]} T_j$
```

As $\arg\min_{j\in[m]} T_j + \frac{\ell_i}{v_j}$ can be found in $O(m)$ time, we can implement the above algorithm to run in $O(nm)$ time. To improve the run time of this algorithm, we implement a faster way to find $\arg\min_{j\in[m]} T_j + \frac{\ell_i}{v_j}$.

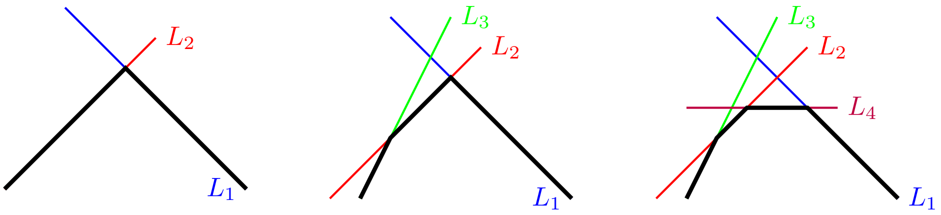


Figure 2 Dynamic lower envelope of lines $L_1 : y = -x + 4$, $L_2 : y = x$, $L_3 : y = 2x$, $L_4 : y = 1$.

5.2 Dynamic Lower Envelope

One can visualise the step in which $\arg\min_{j\in[m]} T_j + \frac{\ell_i}{v_j}$ is computed in the following way: Consider the m linear functions $h_1, h_2, \ldots, h_m$ where $h_j(x) = \dfrac{1}{v_j} \cdot x + T_j$. We need to find the index of $j \in [m]$ for which $h_j(x) = \dfrac{1}{v_j} \cdot x + T_j$ is minimum at $x = l_i$. This is exactly the same as finding the line in the lower envelope for the $h_1, h_2 \cdots h_m$ at $x = l_i$. Our idea now is to maintain a dynamic lower envelope data structure capable of inserting and deleting functions. We will now describe this formally.

Let $\mathcal{H}$ be a set of functions where $h \in \mathcal{H}$ is of the form $h : \mathbb{R} \to \mathbb{R}$. Then the lower envelope is the function $h_{min} : \mathbb{R} \to \mathbb{R}$ such that $h_{min}(x) = \min_{h\in\mathcal{H}} h(x)$. We are interested in the case when these functions correspond to straight lines, i.e., they are of the form $h_i(x) = m_i x + c_i$. More particularly, we are interested in the problem of dynamically maintaining the lower envelope problem of lines, i.e., in each step, a new line is added (or removed), as shown in Figure 2, and one has to maintain the lower envelope with a small update time. This has been well-studied in the computational geometry community. From [43], there is a data structure to maintain a dynamic lower envelope of lines (further extended to dynamic lower envelope of pseudo lines in [1]):

- $\mathcal{H}.Insert(h)$: Adds the linear function h to set $\mathcal{H}$ in $O(\log^2 |\mathcal{H}|)$ time.
- $\mathcal{H}.Delete(h)$: Removes the function h from set $\mathcal{H}$ in $O(\log^2 |\mathcal{H}|)$ time.
- $\mathcal{H}.LowerEnvelope(x)$: Returns $h^* = \arg\min_{h\in\mathcal{H}} h(x)$ in $O(\log |\mathcal{H}|)$ time.

Using this, we can implement the LPT algorithm in $O((n+m)) \log^2 m)$ time (Algorithm 2). We initialise the lower envelope data structure $\mathcal{H}$ and store lines $h_j(x) = \frac{1}{v_j}x + 0$ (initially $T_j = 0$) for all $j \in [m]$. To assign parcel P_i to drone D_j, we remove the line $h_j(x) = \frac{1}{v_j}x + T_j$ from $\mathcal{H}$ and replace it with $h'_j(x) = \frac{1}{v_j}x + T_j + \frac{\ell_i}{v_j}$. To find out which drone P_i is assigned to, we have to find $\arg\min_{j\in[m]} T_j + \frac{\ell_i}{v_j}$, which is the same as querying $\mathcal{H}.LowerEnvelope(\ell_i)$.

Observe that sorting $\mathcal{P}$ takes $O(n \log n)$ time. We have called $\mathcal{H}.Insert(\cdot)$ $O(n+m)$ times and $\mathcal{H}.Delete(\cdot)$ $O(n)$ times. Therefore, the runtime of the algorithm is $O((n + m) \log^2 m + n \log n)$ or $O((n + m)(\log^2 m + \log n))$

6 ϕ-approximation for the Drone Warehouse Problem

In this section, we will prove Theorem 2.

6.1 Algorithm

We use an algorithm similar to LPT but modify it slightly so that the battery constraints are respected. First, we sort the parcels $\mathcal{P}$ in non-increasing order of their distance. Then, we initiate T_j, the time taken by the jth drone to be zero for all $j \in [m]$. We now assign

Algorithm 2 LPT algorithm in $O((n+m)(\log^2 m + \log n)$ time.

```
Input: List of jobs $\mathcal{P}$ and machines $\mathcal{D}$.
Output: Time required for LPT scheduling.
Sort $\mathcal{P}$ in non-increasing order of size.
Initialise lower envelope data structure $\mathcal{H}$
for $j$ in $\{1 \ldots m\}$ do
    $h_j(x) = \frac{1}{v_j}x$
    $\mathcal{H}.Insert(h_j)$
end for
for $i$ in $\{1 \ldots n\}$ do
    $h_j = \mathcal{H}.LowerEnvelope(\ell_i)$
    Let $h_j(x) = \frac{1}{v_j}x + T_j$
    Set $h'_j(x) = \frac{1}{v_j}x + T_j + \frac{\ell_i}{v_j}$
    $\mathcal{H}.Delete(h_j)$
    $\mathcal{H}.Insert(h'_j)$
end for
return $\max_{j \in [m]} h_j(0)$
```

the parcels sequentially from P_1 to P_n. We assign parcel P_i to a drone D_j for which $\ell_i \leq d_j$ and the value of $T_j + (\ell_i/v_j)$ is minimum. We update the value T_j to $T_j + (\ell_i/v_j)$ (see Algorithm 3). It is easy to see that the running time of the LPT algorithm is $O(mn + n\log n)$

6.2 Implementation

Let us sort all the drones in decreasing order of their battery life. Observe that if drone D_j is capable of delivering parcel P_i, then all drones $D_{j'}, j' \leq j$ are capable of delivering parcel P_i. Therefore, we can represent all the drones capable of delivering P_i by a pointer *ptr* so that all drones $D_j, j \in \{1, \ldots, ptr\}$ can deliver parcel P_i. Also, as the parcels are sorted in decreasing order of distance, the value of *ptr* will only increase in each iteration (as any drone capable of delivering P_i can also deliver $P_{i'}, i' > i$). We again use the lower envelope data structure to implement LPT. (see Algorithm 3).

Observe, that sorting $\mathcal{P}$ and $\mathcal{D}$ takes $O(n\log n + m\log m)$ time. We call $\mathcal{H}.Insert(\cdot)$ $O(n+m)$ times and $\mathcal{H}.Delete(\cdot)$ $O(n)$ times. As these function calls only use $O(\log^2 m)$ time, the above algorithm runs in $O((n+m)(\log^2 m + logn))$ time.

6.3 Proof for ϕ-approximation

Simplifying steps

Assume that the parcels are sorted in decreasing order of distance, i.e., if $i < j$ then $\ell_i \geq \ell_j$ for all $i, j \in [n]$. Let Alg_I represent the *valid schedule* obtained from the LPT-algorithm and let Opt_I represent some fixed optimal *valid schedule* on instance $I = \{\mathcal{D}, \mathcal{P}\}$. We show that $\frac{T(Alg_I)}{T(Opt_I)} \leq \phi$. We will start by performing some simplifying steps on I.

▶ **Lemma 3.** *For any instance $I = \{\mathcal{D}, \mathcal{P}\}$, we can assume that $\ell_n = 1$, $T(Opt_I) = 1$ and no drone has battery life less than ℓ_n.*

Algorithm 3 LPT algorithm for DWP.

```
Input: List of drones $\mathcal{D}$ and parcels $\mathcal{P}$.
Output: Minimum time required to deliver all parcels.
Sort $\mathcal{P}$ in non-increasing order of distance.
Sort $\mathcal{D}$ in non-increasing order of battery life
Initialise lower envelope data structure $\mathcal{H}$
$ptr \leftarrow 0$
for $i$ in $\{1 \dots n\}$ do
    while $ptr < n$ and $d_{ptr+1} \geq \ell_i$ do
        $ptr \leftarrow ptr + 1$
        $h_{ptr}(x) = \frac{1}{v_{ptr}} x$
        $\mathcal{H}.Insert(h_{ptr})$
    end while
    $h_j = \mathcal{H}.LowerEnvelope(\ell_i)$
    Let $h_j(x) = \frac{1}{v_j} x + T_j$
    Set $h'_j(x) = \frac{1}{v_j} x + T_j + \frac{\ell_i}{v_j}$
    $\mathcal{H}.Remove(h_j)$
    $\mathcal{H}.Insert(h'_j)$
end for
return $\max_{j \in [m]} h_j(0)$
```

Proof. Observe that scaling all values $\{\ell_i\}_{i\in[n]}, \{d_j\}_{j\in[m]}$ by some constant α scales $T(Alg_I)$ and $T(Opt_I)$ by α. Similarly, scaling all values $\{v_j\}_{j\in[m]}$ by some constant β scales $T(Alg_I)$ and $T(Opt_I)$ by β^{-1}. However, this procedure does not affect the value of the approximation factor $\frac{T(Alg_I)}{T(Opt_I)}$. Therefore, we can choose values α, β such that $\ell_n = 1$ and $T(Opt_I) = 1$ (choosing $\alpha = \ell_n^{-1}, \beta = \ell_n^{-1} T(Opt_I)$ gives us the desired result). Also, as P_n is the smallest job, we can remove all drones which do not have enough battery life to deliver it as such drones would be empty in any schedule. ◀

6.3.1 Idea-1: Working with minimal instances

Our goal is to prove that $T(Alg_I) \leq \phi$ for all instances I. Towards a contradiction, assume that there exists an instance I for which $\frac{T(Alg_I)}{T(Opt_I)} > \phi$. Among all such contradicting instances, let $\mathcal{I}$ be a contradicting instance which has the *minimum* number of parcels. We will sometimes drop the subscripts in $Alg_{\mathcal{I}}$ and $Opt_{\mathcal{I}}$ and write them as Alg and Opt, respectively, for simplicity.

6.3.2 Idea-2: A schedule without the last parcel

As $\mathcal{I} = \{\mathcal{D}, \mathcal{P}\}$ is a contradicting instance with the least number of parcels, it means that for any other instance $\mathcal{I}'$ with fewer parcels that $\mathcal{I}$ is not a contradicting instance. In particular, this is true for $\mathcal{I}' = \{\mathcal{D}, \mathcal{P} \setminus \{P_n\}\}$. Intuitively, this implies $T(Alg_{\mathcal{I}'}) \leq \phi$ and $T(Alg) > \phi$, and adding parcel P_n causes the increase in time. We will now prove this rigorously.

▶ **Definition 4.** *Let Alg_0 be the schedule obtained by removing parcel P_n from the schedule Alg, i.e., Alg_0 is a schedule such that $Alg_0(D_i) = Alg(D_i) \setminus \{P_n\}$ for all $i \in [m]$.*

▶ **Definition 5.** *Let $L_j(f)$ represent the total distance travelled by drone D_j in schedule f. Then,*

$$L_j(f) = \sum_{i:P_i \in f(D_j)} \ell_i.$$

▶ **Definition 6.** *The completion time of a drone in schedule f is the time taken by the drone to deliver all parcels assigned to it by the schedule f.*

▶ **Lemma 7.** $\ell_n + L_i(Alg_0) = 1 + L_i(Alg_0) > \phi v_i$*, for all* $i \in [m]$.

Proof. We claim that $T(Alg_0) \leq \phi$. Suppose not. Then consider the instance $\mathcal{I}' = \{\mathcal{D}, \mathcal{P} \setminus \{P_n\}\}$ obtained from the minimal instance $\mathcal{I} = \{\mathcal{D}, \mathcal{P}\}$. Observe that the schedule $Alg_{\mathcal{I}'}$ and schedule Alg_0 are the same (as the assignment of the first $n-1$ parcels is independent of the n^{th} parcel). Therefore, $T(Alg_{\mathcal{I}'}) > \phi$. Also, observe that $T(Opt_{\mathcal{I}'}) \leq T(Opt_{\mathcal{I}}) = 1$. This implies that $\frac{T(Alg_{\mathcal{I}'})}{T(Opt_{\mathcal{I}'})} > \phi$ which contradicts the fact that $\mathcal{I}$ is an instance with the least number of parcels such that $\frac{T(Alg_{\mathcal{I}})}{T(Opt_{\mathcal{I}})} > \phi$.

Now as $T(Alg) > \phi$ and $T(Alg_0) \leq \phi$, this means that only the drone which delivers P_n has completion time greater than ϕ. Also as the LPT algorithm assigns parcels to drones which have the least completion time, this implies that assigning P_n to any drone D_i in Alg_0 would result in its completion time being greater than ϕ. Note that P_n can be assigned to any drone as all drones have battery life at least ℓ_n (from Lemma 3).

Therefore, $(L_i(Alg_0) + \ell_n)/v_i > \phi$. As $\ell_n = 1$, it follows that $L_i(Alg_0) + 1 > \phi v_i$. ◀

6.3.3 Idea-3: Classification of parcels

We classify parcels for which the drone travels a distance in the range $[1, \phi)$ as *small* jobs, $[\phi, 2)$ as *medium* jobs and $[2, \infty)$ as *large* jobs. We first define a rounding function $R(x)$ such that

$$R(x) = \begin{cases} 1 & \text{if } x \in [1, \phi) \\ 1.5 & \text{if } x \in [\phi, 2) \\ \lfloor x \rfloor & \text{if } x \in [2, \infty) \end{cases}$$

Note that R is a function such that

$$x \geq R(x) \geq x/\phi \text{ for all } x \geq 1$$

Define $L'_i(f) = \sum_{j:P_j \in f(D_i)} R(\ell_j)$. The motivation for defining $R(x)$ is that the value of $L'_i(f)$ can only be integer multiples of 0.5 whereas $L_i(f)$ can take any arbitrary value depending on the input.

6.3.4 Idea-4: Discretizing Distances

Ideally, we would like to show that each drone in Alg_0 travels more distance than its counterpart in *Opt*. This would show that the total distance travelled by drones in Alg_0 is greater than that of drones in *Opt* which is a contradiction as fewer parcels have been delivered in Alg_0 than in *Opt*. Unfortunately, it turns out that $L_i(Alg_0)$ can be lesser than $L_i(Opt)$. Instead, we show that $L'_i(Opt) \leq L'_i(Alg_0)$ for all $i \in [m]$ and arrive at a similar contradiction by analogous argument.

▶ **Lemma 8.** $L'_i(Alg_0) + \phi - 1 > v_i$ *for all* $i \in [m]$

Proof. From Lemma 7, we get that

$$v_i < \frac{1 + L_i(Algo_0)}{\phi} \leq \frac{1}{\phi} + L'_i(Algo_0) = \phi - 1 + L'_i(Algo_0)$$

The second inequality and the third equality follow from $x/\phi \leq R(x)$ and ϕ being the golden ratio, respectively. ◀

▶ **Lemma 9.** *If a drone D_i has a medium job assigned to it, then $L'_i(Algo_0) + 0.5 > v_i$*

Proof. Let a medium-sized job of size y be assigned to drone D_i. Then,

$$L'_i(Algo_0) = R(y) + \sum_{z \in Algo_0(D_i) \setminus \{y\}} R(z)$$

Since $y \in [\phi, 2)$, we get $R(y) = 1.5 > y - 0.5$, and hence,

$$L'_i(Algo_0) > y - 0.5 + \sum_{z \in Algo_0(D_i) \setminus \{y\}} R(z) \geq y - 0.5 + \frac{L_i(Alg) - y}{\phi}$$

Using Lemma 7 and $y \geq \phi$, we get

$$\begin{aligned} L'_i(Algo_0) &> y - 0.5 + \frac{\phi v_i - 1 - y}{\phi} = y(1 - \frac{1}{\phi}) - 0.5 - \frac{1}{\phi} + v_i \\ &\geq \phi - 1 - \frac{1}{\phi} + v_i - 0.5 = v_i - 0.5 \end{aligned}$$

◀

▶ **Lemma 10.** *$L'_i(Opt) \leq L'_i(Algo_0)$ for all $i \in [m]$*

Proof. Towards a contradiction, let us assume that $L'_j(Opt) > L'_j(Algo_0)$ for some drone D_j. Observe that $L'_j(Opt) \leq L_j(Opt) \leq v_j$ as $T(Opt) = 1$. Using this and Lemma 8 we get,

$$L'_j(Algo_0) < L'_j(Opt) \leq v_j < \phi - 1 + L'_j(Algo_0)$$

As $L'_j(Algo_0)$ and $L'_j(Opt)$ are both integer multiples of 0.5, the only value which satisfies the inequality is, $L'_j(Opt) = L'_j(Algo_0) + 0.5$. Let $L'_j(Algo_0) = k/2$ where k is an integer.

Case 1: $L'_j(Algo_0) = k/2$ is not an integer. This can only happen if D_j contains at least one medium job in $Algo_0$ (as small and large sized jobs have integral values and cannot add to give a non-integer). But then by Lemma 9, we get $L'_j(Opt) = L'_j(Algo_0) + 0.5 > v_j$, which is a contradiction as $L'_j(Opt) \leq v_j$.

Case 2: $L'_j(Algo_0) = k/2$ is an integer. This means that $L'_j(Opt) = (k+1)/2$ is not an integer and hence, D_j has at least one medium job assigned to it in Opt. Therefore, by definition of the $R(\cdot)$ function, we get $L_j(Opt) \geq L'_j(Opt) + \phi - 1.5 = L'_j(Algo_0) + \phi - 1$. Using Lemma 8, we get $L_j(Opt) > v_j$, which is a contradiction as $L_j(Opt) \leq v_j$. ◀

Using Lemma 10, we get that

$$\sum_{i=1}^{m} L'_i(Opt) \leq \sum_{i=1}^{m} L'_i(Algo_0).$$

Observe that $\sum_{i=1}^{m} L'_i(Opt) = \sum_1^n R(\ell_i)$ and $\sum_{i=1}^{m} L'_i(Algo_0) = \sum_1^{n-1} R(\ell_i)$. Substituting this, we get

$$\sum_1^n R(\ell_i) \leq \sum_1^{n-1} R(\ell_i),$$

which is a contradiction. Therefore, our initial assumption must be wrong, implying that $T(Alg) \leq \phi$.

6.4 How much can the above analysis of LPT be improved?

Recall that the special case of DWP when all drones have the same battery life ($d_i = d_j \geq \max_k(\ell_k)$ for all $i, j \in [m]$) is equivalent to USP. It is known that LPT cannot give an approximation better than 1.54 for USP [28]. Therefore, the LPT algorithm can also not give an approximation ratio better than 1.54 for DWP.

7 Future work

A couple of immediate open problems are the following:

1. Can the implementation of the LPT heuristic be done in optimal time, i.e. $O((n+m) \cdot (\log m + \log n)$ time? We believe that it should be possible, since the jobs are known upfront, i.e., it is actually an offline problem.
2. Can the approximation ratio of the LPT heuristic be improved for DWP from ϕ?

In general, delivering parcels from the warehouse using drones is a rich source of scheduling and vehicle routing problems. We mention two general directions:

1. As a concrete setting, consider a warehouse which has a truck and a drone, both of which operate independently. As in the paper, the goal is to assign parcels to the truck and the drone so that time taken to deliver is minimized. The parcels will be delivered by the drone using the same model as in this paper, whereas the truck will deliver using the traditional technique. A generalized version of the problem would involve multiple trucks and drones.
2. Some companies might have multiple warehouses and as such, a parcel can be delivered by any of the warehouses. As a concrete setting, consider the generalization of the DWP studied in this paper to the setting where there are *two* warehouses at different locations. The goal is to perform a two-level partition: first partition the parcels among the warehouses and then partition them among the drones. The goal is to deliver all the parcels as quickly as possible.

References

1 Pankaj K. Agarwal, Ravid Cohen, Dan Halperin, and Wolfgang Mulzer. Maintaining the union of unit discs under insertions with near-optimal overhead. In Gill Barequet and Yusu Wang, editors, *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA*, volume 129 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.SOCG.2019.26.

2 Niels A. H. Agatz, Paul C. Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transp. Sci.*, 52(4):965–981, 2018. doi:10.1287/TRSC.2017.0791.

3 Pasquale Ambrosio and Vincenzo Auletta. Deterministic monotone algorithms for scheduling on related machines. *Theor. Comput. Sci.*, 406(3):173–186, 2008. doi:10.1016/J.TCS.2008.06.050.

4 Nir Andelman, Yossi Azar, and Motti Sorani. Truthful approximation mechanisms for scheduling selfish related machines. *Theory Comput. Syst.*, 40(4):423–436, 2007. doi:10.1007/S00224-006-1316-9.

5 Vincenzo Auletta, Roberto De Prisco, Paolo Penna, and Giuseppe Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In Volker Diekert and Michel Habib, editors, *STACS 2004, 21st Annual Symposium on Theoretical Aspects of Computer Science, Montpellier, France, March 25-27, 2004, Proceedings*, volume 2996 of *Lecture Notes in Computer Science*, pages 608–619. Springer, 2004. doi:10.1007/978-3-540-24749-4_53.

6 François Bodin, Tristan Charrier, Arthur Queffelec, and François Schwarzentruber. Generating plans for cooperative connected uavs. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5811–5813. ijcai.org, 2018. doi:10.24963/IJCAI.2018/846.

7 John L. Bruno, Edward G. Coffman Jr., and Ravi Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974. doi:10.1145/361011.361064.

8 John Gunnar Carlsson and Siyuan Song. Coordinated logistics with a truck and a drone. *Manag. Sci.*, 64(9):4052–4069, 2018. doi:10.1287/MNSC.2017.2824.

9 Shushman Choudhury, Kiril Solovey, Mykel J. Kochenderfer, and Marco Pavone. Coordinated multi-agent pathfinding for drones and trucks over road networks. In *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022*, pages 272–280. IFAAMAS, 2022. doi:10.5555/3535850.3535882.

10 Gregory Dobson. Scheduling independent tasks on uniform processors. *SIAM Journal on Computing*, 13(4):705–716, 1984. doi:10.1137/0213044.

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

12 Donald K. Friesen. Tighter bounds for lpt scheduling on uniform processors. *SIAM Journal on Computing*, 16(3):554–560, 1987. doi:10.1137/0216037.

13 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

14 Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

15 Inge Li Gørtz, Marco Molinaro, Viswanath Nagarajan, and R. Ravi. Capacitated vehicle routing with non-uniform speeds. In Oktay Günlük and Gerhard J. Woeginger, editors, *Integer Programming and Combinatoral Optimization*, pages 235–247, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-20807-2_19.

16 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

17 R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

18 F. Guerriero, R. Surace, V. Loscrí, and E. Natalizio. A multi-objective approach for unmanned aerial vehicle routing problem with soft time windows constraints. *Applied Mathematical Modelling*, 38(3):839–852, 2014.

19 Erez Hartuv, Noa Agmon, and Sarit Kraus. Scheduling spare drones for persistent task performance under energy constraints. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 532–540. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018. URL: http://dl.acm.org/citation.cfm?id=3237463.

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

21 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988. doi:10.1137/0217033.

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

23 Shlomo Karhi and Dvir Shabtay. Online scheduling of two job types on a set of multipurpose machines. *International Journal of Production Economics*, 150:155–162, 2014. doi:10.1016/j.ijpe.2013.12.015.

24 Jonghoe Kim and James R. Morrison. On the concerted design and scheduling of multiple resources for persistent uav operations. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 942–951, 2013.

25 Jonghoe Kim, Byung Duk Song, and James R. Morrison. On the scheduling of systems of uavs and fuel service stations for long-term mission fulfillment. *J. Intell. Robotic Syst.*, 70(1-4):347–359, 2013. doi:10.1007/S10846-012-9727-0.

26 David Klaska, Antonín Kucera, and Vojtech Rehák. Adversarial patrolling with drones. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 629–637, 2020. doi:10.5555/3398761.3398837.

27 Annamária Kovács. Fast monotone 3-approximation algorithm for scheduling related machines. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, volume 3669 of *Lecture Notes in Computer Science*, pages 616–627. Springer, 2005. doi:10.1007/11561071_55.

28 Annamária Kovács. New approximation bounds for lpt scheduling. *Algorithmica*, 57(2):413–433, 2010. doi:10.1007/S00453-008-9224-9.

29 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.

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

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

32 Joseph Y.-T. Leung and C. T. Ng. Fast approximation algorithms for uniform machine scheduling with processing set restrictions. *Eur. J. Oper. Res.*, 260(2):507–513, 2017. doi:10.1016/J.EJOR.2017.01.013.

33 Chung-Lun Li and Kangbok Lee. A note on scheduling jobs with equal processing times and inclusive processing set restrictions. *J. Oper. Res. Soc.*, 67(1):83–86, 2016. doi:10.1057/JORS.2015.56.

34 Chung-Lun Li and Qingying Li. Scheduling jobs with release dates, equal processing times, and inclusive processing set restrictions. *J. Oper. Res. Soc.*, 66(3):516–523, 2015. doi:10.1057/JORS.2014.22.

35 Chung-Lun Li and Xiuli Wang. Scheduling parallel machines with inclusive processing set restrictions and job release times. *Eur. J. Oper. Res.*, 200(3):702–710, 2010. doi:10.1016/J.EJOR.2009.02.011.

36 Amith Manoharan. Strategies for cooperative uavs using model predictive control. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 5196–5197. ijcai.org, 2020. doi:10.24963/IJCAI.2020/738.

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

38 Chase C. Murray and Amanda G. Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

39 Ty Nguyen and Tsz-Chiu Au. Extending the range of delivery drones by exploratory learning of energy models. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1658–1660. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3091395.

40 Marta Niccolini, Mario Innocenti, and Lorenzo Pollini. Multiple uav task assignment using descriptor functions. *IFAC Proceedings Volumes*, 43(15):93–98, 2010. 18th IFAC Symposium on Automatic Control in Aerospace.

41 Alena Otto, Niels A. H. Agatz, James F. Campbell, Bruce L. Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018. doi:10.1002/NET.21818.

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

43 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. doi:10.1016/0022-0000(81)90012-X.

44 Sivakumar Rathinam and Raja Sengupta. Algorithms for routing problems involving uavs. In *Innovations in Intelligent Machines - 1*, volume 70 of *Studies in Computational Intelligence*, pages 147–172. Springer, 2007. doi:10.1007/978-3-540-72696-8_6.

45 Dvir Shabtay and Shlomo Karhi. Online scheduling of two job types on a set of multipurpose machines with unit processing times. *Computers and Operations Research*, 39(2):405–412, 2012. doi:10.1016/j.cor.2011.05.002.

46 Kaarthik Sundar and Sivakumar Rathinam. Algorithms for heterogeneous, multiple depot, multiple unmanned vehicle path planning problems. *J. Intell. Robotic Syst.*, 88(2-4):513–526, 2017. doi:10.1007/S10846-016-0458-5.

47 Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

48 Feng Wu, Sarvapali D. Ramchurn, and Xiaoping Chen. Coordinating human-uav teams in disaster response. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 524–530. IJCAI/AAAI Press, 2016. URL: http://www.ijcai.org/Abstract/16/081.

49 Miao Yu, Viswanath Nagarajan, and Siqian Shen. An approximation algorithm for vehicle routing with compatibility constraints. *Operations Research Letters*, 46(6):579–584, 2018. doi:10.1016/j.orl.2018.10.002.

Circuits, Proofs and Propositional Model Counting

Sravanthi Chede
Indian Institute of Technology Ropar, Rupnagar, India

Leroy Chew
TU Wien, Austria

Anil Shukla
Indian Institute of Technology Ropar, Rupnagar, India

Abstract

In this paper we present a new proof system framework CLIP (Circuit Linear Induction Proposition) for propositional model counting (#SAT). A CLIP proof firstly involves a Boolean circuit, calculating the cumulative function (or running count) of models counted up to a point, and secondly a propositional proof arguing for the correctness of the circuit.

This concept is remarkably simple and CLIP is modular so it allows us to use existing checking formats from propositional logic, especially strong proof systems. CLIP has polynomial-size proofs for XOR-pairs which are known to require exponential-size proofs in MICE [16]. The existence of a strong proof system that can tackle these hard problems was posed as an open problem in Beyersdorff et al. [3]. In addition, CLIP systems can p-simulate all other existing #SAT proofs systems (KCPS(#SAT) [8], CPOG [4], MICE). Furthermore, CLIP has a theoretical advantage over the other #SAT proof systems in the sense that CLIP only has lower bounds from its propositional proof system or if $\mathsf{P}^{\#\mathsf{P}}$ is not contained in P/poly, which is a major open problem in circuit complexity.

CLIP uses unrestricted circuits in its proof as compared to restricted structures used by the existing #SAT proof systems. In this way, CLIP avoids hardness or limitations due to circuit restrictions.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Propositional model counting, Boolean circuits, #SAT, Proof Systems, Certified Partition Operation Graph (CPOG)

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.18

Funding *Leroy Chew*: This work was supported by FWF ESPRIT grant no. ESP 197.

Acknowledgements We thank Olaf Beyersdorff, Tim Hoffmann, Luc Nicolas Spachmann for useful discussions on this topic. We also thank the anonymous reviewers for helpful suggestions on the paper.

1 Introduction

Given a propositional formula, the problem of finding its total number of satisfying assignments (models) is known as the propositional model counting problem #SAT [24]. The problem is known to be #P-complete and is considered one of the hardest problem in the field of computational complexity. In fact, it is known that with a single call to a #SAT-oracle, any problem from polynomial hierarchy can be solved in polynomial time (Toda's Theorem [27]).

Over the last few years, some important proof systems have been developed for #SAT. The knowledge compilation based proof system (KCPS(#SAT)) [8] is the first non-trivial proof system designed for #SAT. A KCPS(#SAT) proof for a CNF represents the proposition as a decision-DNNF (Decomposable Negation Normal Form), with some additional annotations for checking. A decision-DNNF allows for model counting to be easily extracted. However, limitations and lower bounds for KCPS(#SAT) have already been established [2, 8].

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 18; pp. 18:1–18:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The second proof system designed for #SAT is MICE (Model-counting Induction by Claim Extension) [16]. Unlike KCPS(#SAT), it is a line based proof system which computes the model count in a step-by-step fashion using some simple inference rules. Several lower bounds for MICE have been established in the literature [2, 3]. For example, XOR-PAIRS [3], are shown to be hard for the MICE proof system. Recently, an important proof system CPOG [4] was introduced for #SAT. Similar to KCPS(#SAT), CPOG is also based on knowledge compilation. A CPOG proof for a CNF formula ϕ consists of a Partitioned-Operation Graph (POG) G along with a Resolution proof of the fact that $G \equiv \phi$.

The relationship between these three proof systems are now well known. It has been shown in [2], that CPOG is exponentially stronger than KCPS(#SAT) and MICE. On the other hand, KCPS(#SAT) and MICE are incomparable [2, Figure 1]. This means that KCPS(#SAT) and MICE have unconditional lower bounds. For CPOG a lower bound is currently unknown, but its proof complexity is necessarily tied to the limitations of POGs.

In this paper, we introduce a new #SAT proof format CLIP (Circuit Linear Induction Proposition) (Definition 5). A CLIP based proof for a CNF formula ϕ consists of a Boolean circuit calculating the running count of models up to an assignment treated in some fixed lexicographical order. We denote this Boolean circuit as a cumulator (Definition 3). In addition to the cumulator, the CLIP proof also contains a certificate proving the correctness of the cumulator. The CLIP format is similar to CPOG in the sense that instead of a POG, CLIP has a cumulator. Since POG uses restricted versions of AND and OR gates, as compared to cumulators, we believe that CLIP format is much stronger than CPOG. In this direction, we show that CLIP can p-simulate CPOG (Theorem 30). In fact, we show that CLIP has lower bounds only if some major open problems of proof complexity or circuit complexity are solved (Theorem 7).

In MICE and KCPS(#SAT), proofs can grow exponentially because of unsatisfiable formulas that have lower bounds in Resolution. In this direction, for any unsatisfiable formula which is hard in Resolution, it is unclear how large the CPOG proofs will be. However, for unsatisfiable formulas in CLIP, proofs are no bigger than their shortest DRAT proofs (Proposition 35). In addition, we also show that XOR-PAIRS which are known to be hard for existing proof systems are easy for the CLIP format (Theorem 34). We sum up our contributions in the Figure 1. We explain the same in detail in the following subsection.

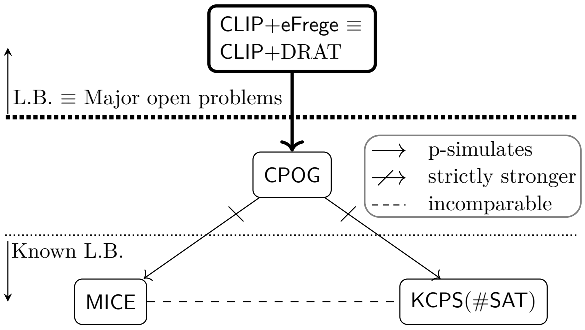


Figure 1 *Hierarchy of #SAT proof systems. New results are shown in bold.*

Our Contributions

1. **Introducing a new proof system framework for #SAT** (CLIP): We present a proof system where proofs are pairs containing a circuit and a propositional proof. The circuit is a multi-output Boolean circuit we call the *cumulator* which takes a complete assignment α and returns the number (in binary) of models of a propositional formula up to α in some fixed lexicographical ordering of assignments. In addition, CLIP proofs also contain a certificate showing the correctness of the cumulator. The certificate here is a propositional proof. This is possible because we can construct a tautology that covers every inductive step for any two consecutive complete assignments. The CLIP format allows us to use any known propositional proof system $\mathcal{P}$ for proving the correctness of the cumulator. In this paper we focus on CLIP+ Extended Frege (CLIP+eFrege). We show that CLIP+eFrege is a powerful proof system in the sense that it has a lower bound only if a super-polynomial lower bound for eFrege is found or it is proven that $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P/poly}$ (Theorem 7). Hence proving lower bounds in CLIP+eFrege will lead to solving major open problems in the fields of proof complexity or circuit complexity. Another way to say this is that CLIP is the first #SAT proof system which is conditionally optimal. Note that such systems already exist in the propositional and QBF worlds, i.e. IPS (Ideal Proof System) [18] and eFrege +∀red [1] system respectively.
2. **A CLIP+eFrege simulation technique for all existing #SAT proof systems**: The CLIP+eFrege simulation technique consists of three parts. The first part consists of extracting a cumulator from any #SAT proof system which is closed under restrictions. The second part establishes that if a #SAT proof system admits easy eFrege proofs of the "properties of restriction" (Definition 15) then it can be p-simulated by CLIP+eFrege. The final part proves that the CPOG system admits all the required properties and hence can be p-simulated by CLIP+eFrege. Let us briefly explain each of them separately.
   a. **Cumulator extraction** (Section 4.1). We show that from any #SAT proof system which is closed under restrictions (Definition 1), there is a simple technique to efficiently extract a cumulator from its proofs. For this, we carefully use the concept of Fenwick trees [15, 26] to introduce and compute the Fenwick assignments (Definition 11). Informally, the Fenwick assignments are a small set of partial assignments that collectively covers all assignments up to a given complete assignment (Definition 9).
   b. **Extended Frege (eFrege) certification of the cumulator** (Section 4.2). Informally, the properties of restriction imply that after restricting a proof with a complete assignment α, the model count will be "one" or "Zero" depending on whether α satisfies the formula or not. Whereas, in the case of restricting a proof with a partial assignment α undefined on some variable x, the model count returned should be the sum of model counts returned when restricting seperately with α_0 and α_1, where $\alpha_b = \alpha \cup \{x = b\}$. These two are the only properties we need to know which when globally combined tell us that the model count under the restriction of a partial assignment is correct. If a #SAT proof system admits easy eFrege proofs for these properties of restriction (Definition 15), we show that it can be p-simulated by CLIP+eFrege (Theorem 19).
   c. **CLIP+eFrege p-simulates CPOG** (Section 5). Using the structure of the POG to form an inductive proof. We explicitly show that the CPOG proof system admits easy eFrege proofs of the properties of restriction (Lemma 29). Thereby, proving that CLIP+eFrege p-simulates CPOG (Theorem 30), which in-turn p-simulates KCPS(#SAT) and MICE [2]. This shows that CLIP+eFrege p-simulates all existing #SAT proof systems.

3. **Upperbounds in CLIP for some hard formulas of existing #SAT systems**:

   a. **XOR-PAIRS**. Since the CLIP framework uses unrestricted Boolean circuits in its proof as compared to other existing #SAT proof systems, CLIP is capable of handling formulas that are hard for other systems. We show this for the family XOR-PAIRS, which are known to be hard for MICE [3], and give an easy proof for the same in the CLIP+eFrege proof system (Theorem 34). For the short proof, we first carefully define a short cumulator for the XOR-PAIRS. Then, using a constant case analysis we certify the correctness of the cumulator in eFrege.

   b. **Unsatisfiable formulas**. We show that any unsatisfiable formula which has an easy eFrege proof, also has an easy CLIP+eFrege proof (Proposition 35). It is already known that for unsatisfiable formulas, MICE and KCPS(#SAT) are p-equivalent to Resolution and regular-Resolution respectively [2, Proposition 5.1, 5.3]. As a result, all unsatisfiable formulas, which are hard for Resolution and easy for eFrege are all hard for MICE and KCPS(#SAT) but easy for CLIP+eFrege. We list three such important counting based unsatisfiable formulas. Namely, the pigeonhole principle (PHP), the clique-coloring principle [22, Definition 7.1] and the Random Parity principle, which are known to be hard for Resolution [19, 22, 9] but are easy for eFrege [13, 6, 7, 10].

2 Preliminaries

For a Boolean variable x, its literals can be x or $\neg x$. We use the notation $\overline{\ell} = \neg x$ when $\ell = x$ and $\overline{\ell} = x$ when $\ell = \neg x$. A clause C is a disjunction of literals and a conjunctive normal form (CNF) formula ϕ is a conjunction of clauses. We denote the empty clause by $\perp$. $vars(\phi)$ is the set of all variables in formula ϕ.

2.1 Assignments

A partial assignment is a partial mapping from a set of propositional variables X to $\{0, 1\}$, when the mapping is defined everywhere we say the assignment is complete. $\langle X \rangle$ is the set of all complete assignments. Consider a totally ordered set of variables X. An **initial assignment** α is a partial assignment to X such that there are no pairs $x, y \in X$ where $x < y$ and x is undefined in α and y is defined. $vars(\alpha)$ are the variables for which α is defined and $|\alpha|$ represents $|vars(\alpha)|$. A partial assignment α can be extended to a total assignment by appending $0/1$ assignment to the variables $X \setminus vars(\alpha)$. Two partial assignments α, β are called non-overlapping, if there does not exist any total assignment γ which can be obtained by extending both α and β.

For a CNF ϕ, $\phi|_\alpha$ (similarly $C|_\alpha$) denotes the restricted formula (or clause) resulting from replacing all occurrences of $vars(\alpha)$ in ϕ (or C) with assignments from α. For a propositional formula F we define the indicator function $\mathbb{1}_F$, this acts on the free variables of F. $\mathbb{1}_F$ is equal to an assignment that corresponds to $\mathbf{0}$ when F is false and $\mathbf{1}$ when F is true.

When variables are ordered as $X =< x_{n-1}, \dots x_0 >$, complete assignments can be seen as binary numbers i.e. $\{x_2 = 1, x_1 = 0, x_0 = 1\}$ represents $\mathbf{5}$.

Let num map assignments to integers using the standard binary encoding ($num(\alpha) = \sum_{i=0}^{i<n} \mathbb{1}_{\alpha(x_i)} \cdot 2^i$), and num^{-1} be its inverse. We also encapsulate arithmetic statements with $|| \cdot ||$ to indicate that we revert this into a proposition. Later we will drop this notation when obvious. We denote $[J]$ to denote numbers $\{\mathbf{1}, \mathbf{2}, \dots, J - \mathbf{1}, J\}$ and $[J_1, J_2]$ to denote numbers $\{J_1, J_1 + \mathbf{1}, \dots, J_2 - \mathbf{1}, J_2\}$. We distinguish numerals $\mathbf{0}$ and $\mathbf{1}$ from Boolean constants 0 and 1 through the use of boldface.

Given a formula ϕ over a set of variables X, a model is an complete assignment to X that satisfies ϕ. The set of models of ϕ is $\mathcal{M}(\phi)$. We denote the total number of models of a CNF ϕ as $\#_{\text{models}}(\phi) = |\mathcal{M}(\phi)|$. #SAT is the computational problem of calculating $\#_{\text{models}}(\phi)$ from a CNF ϕ. The class of languages decidable in polynomial time with an oracle to #SAT are denoted by $P^{\#P}$.

2.2 Circuits

A Boolean *circuit* σ on variables X is a directed acyclic graph, in which the input nodes (with in-degree 0) are Boolean variables $\in X$ and other nodes are the basic Boolean operations: $\vee$ (OR), $\wedge$ (AND) and $\neg$ (NOT) and have in-degree at most 2. Every Boolean circuit σ evaluates a Boolean function whose output is that of the node with out-degree 0 in σ. P/poly is the class of Boolean functions computed by polynomial-sized circuit families.

We refer to a *multi-circuit* when we have multiple nodes with out-degree 0. This is simultaneously many overlapping circuits. Multi-circuits take inputs and outputs of Boolean vectors of fixed length. We denote the Boolean XOR gate with $\oplus$ in the paper. Likewise we use $\leftrightarrow$ (or $=$, or $\equiv$) for bi-equivalence and $A \models B$ to mean that models of A are also models of B. A CNF ϕ can trivially be represented as a Boolean circuit σ as follows: for every $C \in \phi$, σ has $|vars(C)|$ number of OR-gates. Then, σ has $m-1$ AND-gates where m is the number of clauses $\in \phi$.

2.3 Proof Systems

A *proof system* [12] is a polynomial-time function that maps proofs to theorems, where the set of theorems is some fixed language L. A proof system is sound if its image is contained in L and complete if L is contained in its image. A proof system takes in strings as its inputs. Let π be such a proof we denote its *size*, i.e. the string length by $|\pi|$. Given two proof systems f and g for the same language L. We that the f *p-simulates* g, when there is a polynomial time function r that maps g-proofs to f-proofs such that $g(\pi_1) = f(r(\pi_1))$. f and g are said to be *p-equivalent* if they both p-simulate each other,f and g are incomaparable if neither of them p-simulates the other. We say that f is exponentially stronger than g, if f p-simulates g but g does not p-simulate f.

Conventionally we may take L to be the set of propositional tautologies (as in Section 2.3.1). For propositional model counting, we take L as the set of all pairs $(\phi, \#_{\text{models}}(\phi))$, where ϕ is any propositional formula. We refer to a #SAT proof of $(\phi, \#_{\text{models}}(\phi))$ as a proof of ϕ.

▶ **Definition 1** (Closure under restrictions [23]). *A proof system P is closed under restrictions if for every P-proof π of a CNF formula ϕ and any partial assignment α to vars(ϕ), there exists a P-proof π' of $\phi|_\alpha$ such that $|\pi'| \leq p(|\pi|)$ for some polynomial p. In addition, there exists a polynomial time procedure (w.r.t. $|\pi|$) to extract π' from π.*

A similar definition called "closure under conditioning" exists in the knowledge-compilation domain [14, Definition 3]. Precisely, a knowledge representation structure S (like DNNF, POG, etc) is closed under conditioning if from an S structure T and an assignment α to $vars(T)$, another S structure T' can be computed just by replacing all occurrences of free variables by α wherever defined. Additionally T' should be equivalent to $T \wedge \alpha$.

2.3.1 Propositional Proof Systems

Resolution [25] is arguably the most studied propositional proof system. It has the rule $\frac{(C\vee x) \quad (D\vee \overline{x})}{(C\cup D)}$ where C, D are clauses and x is a variable. Resolution refutation ρ of CNF ϕ is a derivation of $\bot$ using the above rule. Resolution is known to be closed under restrictions.

Frege systems [17] are important propositional proof systems. They consist of a sound and complete set of axioms and rules where any variable can be substituted by any formula. All Frege systems are p-equivalent [12]. Figure 2 gives one example of a Frege system.

$$\frac{}{1} \qquad \frac{}{x_1 \to (x_2 \to x_1)} \qquad \frac{}{((x_1 \to 0) \to 0) \to x_1}$$

$$\frac{}{(x_1 \to (x_2 \to x_3)) \to ((x_1 \to x_2) \to (x_1 \to x_3))} \qquad \frac{x_1 \qquad x_1 \to x_2}{x_2}$$

Figure 2 A Frege system for connectives $\to, 0, 1$.

Extended Frege (eFrege) [12] allows the introduction of new variables as well as all Frege rules. Simultaneously we can imagine it as a Frege system where lines are circuits instead of formulas, or as Substitution Frege, where derived tautologies can be generalised. eFrege is also p-equivalent to **DRAT** (Deletion Resolution Asymmetric Tautology) [21], a practical proof-format widely used in certifying SAT solvers. In Figure 3 we show that eFrege sits at the top of the simulation hierarchy of propositional proof systems. In fact eFrege can simulate any proof system as long as there is a short proof of the reflection principle of said proof system [20].

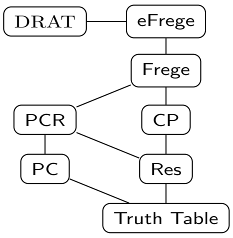


Figure 3 The p-simulation hierarchy of propositional proof systems [12].

When showing that eFrege has short proofs for complicated tautologies, it does not help us to be committed to one strictly defined proof system. Instead, we can use the fact that it can simulate many different proof systems such as Resolution, Cutting planes, Polynomial calculus and Truth Tables. Any tautology that has a short proof in any weaker system will also have a short proof in eFrege.

3 Circuit Linear Induction Proposition (CLIP) Proof Framework

In this section, we define a propositional model counting proof framework (CLIP+$\mathcal{P}$) for any propositional proof system $\mathcal{P}$. Given a CNF ϕ over n variables, a CLIP+$\mathcal{P}$ proof consists of a Boolean circuit ξ (denoted as a *cumulator*) which outputs the total number of models of ϕ from complete assignment $\mathbf{0}$ to a given complete assignment $num(\alpha)$ (denoted as $C_{models}(\phi, \alpha)$, Definition 2). Clearly, when $num(\alpha) = \mathbf{2}^n - \mathbf{1}$, $C_{models}(\phi, \alpha) = \#_{\text{models}}(\phi)$. In addition, the CLIP+$\mathcal{P}$-proof also requires a $\mathcal{P}$-proof of a statement which carefully encodes the correctness of cumulator ξ using the induction-principle (see Definition 5). We need the following definitions.

▶ **Definition 2** ($C_{\mathsf{models}}(\phi, \alpha)$)**.** *Let ϕ be an CNF formula, fix an order among $vars(\phi)$. For any complete assignment α to $vars(\phi)$, the cumulative number of models of CNF ϕ w.r.t. α (denoted by $C_{models}(\phi, \alpha)$) is the number of models of ϕ between assignment $\mathbf{0}$ to assignment $num(\alpha)$. In other words $C_{models}(\phi, \alpha) := \Sigma_{num(\beta) \leq num(\alpha)} \mathbb{1}_{\phi(\beta)}$, where $\mathbb{1}_{\phi(\beta)}$ is the indicator function for when β is a model of ϕ.*

▶ **Definition 3** (Cumulator)**.** *A cumulator for a CNF ϕ over n variables is a Boolean multi-circuit $\xi(\alpha)$ which takes as input a complete assignment α to $vars(\phi)$ (as n binary bits) and calculates the cumulative number of models of ϕ, i.e. $C_{models}(\phi, \alpha)$ outputted as $n + 1$ binary bits. As a result, when α is the last assignment (i.e. $num(\alpha) = \mathbf{2^n} - \mathbf{1}$), $\xi(\alpha)$ outputs the total number of models of ϕ, we denote this as the final output of ξ.*

A trivial cumulator for ϕ would be to keep a counter and given any α, input every assignment from $\mathbf{0}$ to $num(\alpha)$ into the trivial Boolean circuit representing ϕ. If an assignment is a model then increment the counter. This will take $\mathcal{O}(2^{|vars(\phi)|})$ computations in the case of α being the last assignment.

Consider a CNF ϕ and let k be its number of models. Given a cumulator $\xi(\alpha)$ for ϕ, the correctness of the cumulator can be encoded inductively as follows:

For the base case when $num(\alpha) = \mathbf{0}$, we need to verify that the following is satisfied: $(\phi(\alpha) \ \wedge \ ||\xi(\alpha) = \mathbf{1}||) \ \vee \ (\overline{\phi(\alpha)} \ \wedge \ ||\xi(\alpha) = \mathbf{0}||)$. This covers the case that if the first assignment is a model for ϕ then the cumulator should return $\mathbf{1}$, else a $\mathbf{0}$.

For the inductive step when $num(\alpha) = num(\beta) + \mathbf{1}$, the following should be satisfied $(\phi(\alpha) \ \wedge \ ||\xi(\alpha) = \xi(\beta) + \mathbf{1}||) \ \vee \ (\overline{\phi(\alpha)} \ \wedge \ ||\xi(\alpha) = \xi(\beta)||)$. This covers the case that if the next assignment α after β is a model of ϕ, then the cumulator should increment its output by 1. Otherwise, the cumulator should output the same number under both assignments.

For the final case when $num(\alpha) = \mathbf{2}^{|vars(\phi)|} - \mathbf{1}$, it should be true that $\xi(x) = k$. This covers the case that the cumulator computes the correct total number of models of ϕ.

It is clear to see that if all of the above cases are true, the cumulator ξ is proven to be a correct cumulator of ϕ. From the above discussion, one can encode the correctness of ξ as the following statement ($|| \cdot ||$ encloses the arithmetic comparisons needed):
$||num(\alpha) = \mathbf{0}|| \rightarrow \big((\phi(\alpha) \wedge ||\xi(\alpha) = \mathbf{1}||) \vee (\neg\phi(\alpha) \wedge ||\xi(\alpha) = \mathbf{0}||)\big)$
$\wedge ||num(\alpha) = num(\beta) + \mathbf{1}|| \rightarrow \big((\phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta) + \mathbf{1}||) \vee (\neg\phi(\alpha) \wedge ||\xi(\alpha) = \xi(\beta)||)\big)$
$\wedge ||num(\alpha) = \mathbf{2}^{|vars(\phi)|} - \mathbf{1}|| \rightarrow ||\xi(x) = k||$.

To convert this into a purely propositional statement, we need Boolean circuits to implement the arithmetic conditions $||x = y||$ and $||x = y + \mathbf{1}||$ for any integers x, y. We define polynomial sized Boolean circuits for the same as $E(x, y)$ and $T(x, y)$ respectively in Definition 4 below.

▶ **Definition 4.** *Let Z be a set of variables of size n, and let γ and δ be assignments to Z. For pairs of individual variables a, b, use $a = b$ to denote $(\neg a \vee b) \wedge (a \vee \neg b)$. We can encode polynomial size propositional circuits:*

- *$E(\gamma, \delta)$, that denotes $num(\gamma) = num(\delta)$: $E_0(\gamma, \delta) := (\gamma_0 \leftrightarrow \delta_0)$. For $1 \leq i < n$, $E_i(\gamma, \delta) := (\gamma_i \leftrightarrow \delta_i) \wedge E_{i-1}(\gamma, \delta)$. $E(\gamma, \delta) := E_{n-1}(\gamma, \delta)$.*
- *$T(\gamma, \delta)$, that denotes $num(\gamma) = num(\delta) + \mathbf{1}$ using an intermediate definition S that will denote the successor function. For $0 \leq i < n$ and accepting the empty conjunction as true, $S(\delta)_i := \neg(\delta_i \leftrightarrow \bigwedge_{j\geq 0}^{j<i} \delta_j)$. $T(\gamma, \delta) := E(\gamma, S(\delta)) \wedge \bigvee_{i \geq 0}^{i<n} \overline{\delta}_i$.*

▶ **Definition 5** (CLIP+$\mathcal{P}$)**.** *For every propositional proof system $\mathcal{P}$, the* CLIP*+$\mathcal{P}$ system for #SAT is a cumulator ξ for a CNF ϕ along with its correctness presented as a valid $\mathcal{P}$-proof of the following linear induction proposition statement* $\mathsf{lip}(\xi)$.

Let A and B be two disjoint copies of the variables in ϕ. The following is a tautology in the variables of $A \cup B$:

$$\begin{aligned}
&\mathsf{lip}(\xi) := \\
&E\big(A, num^{-1}(\mathbf{0})\big) \rightarrow \Big(\Big(\overline{\phi(A)} \rightarrow E\big(\xi(A), num^{-1}(\mathbf{0})\big)\Big) \wedge \Big(\phi(A) \rightarrow T(\xi(A), num^{-1}(\mathbf{0}))\Big)\Big) \wedge \\
&T(B, A) \rightarrow \Big(\Big(\overline{\phi(B)} \rightarrow E\big(\xi(B), \xi(A)\big)\Big) \wedge \Big(\phi(B) \rightarrow T\big(\xi(B), \xi(A)\big)\Big)\Big) \wedge \\
&E\big(A, num^{-1}(\mathbf{2}^{|vars(\phi)|} - \mathbf{1})\big) \rightarrow E\big(\xi(A), num^{-1}(k)\big).
\end{aligned}$$

The existence of a valid $\mathcal{P}$-proof of $\mathsf{lip}(\xi)$, ensures that ξ is correct and the final output k of ξ is the correct number of models of ϕ. Note that in a technical sense the proof of inductive step (i.e. line 2 in $\mathsf{lip}(\xi)$) is sufficient to verify the cumulator ξ, as the base and final case can be managed in the checker.

▶ **Theorem 6.** *If $\mathcal{P}$ is a propositional proof system then* CLIP+$\mathcal{P}$ *is a propositional model counting proof system.*

Proof. CLIP+$\mathcal{P}$ is sound and complete for #SAT as a trivial cumulator always exists for any ϕ and the propositional proof system $\mathcal{P}$ is sound and complete. Note that for a refutational proof system $\mathcal{P}'$, CLIP+$\mathcal{P}'$ can include the correctness of ξ by including a $\mathcal{P}'$-refutation of $\overline{\mathsf{lip}(\xi)}$ from the above definition. For polynomial time checkability, we perform three steps: a) Verify that ξ is indeed a circuit. b) Using ξ, generate $\mathsf{lip}(\xi)$ once again, to make sure it matches (where $\mathcal{P}$ does not accept circuits a canonical translation, i.e. a Tseitin transformation is needed). c) Verifying the $\mathcal{P}$ proof. ◀

▶ **Theorem 7.** CLIP+eFrege *has a super-polynomial lower bound only if* eFrege *has a super-polynomial lower bound or* $\mathsf{P}^{\#\mathsf{P}} \not\subseteq \mathsf{P/poly}$.

Proof. Suppose there is a family $(\phi_n)_{n\geq 0}$ of propositional formulas that are a super-polynomial lower bound to CLIP+eFrege. Let $f_{n,i}$ be the i^{th} bit of the cumulative function for ϕ_n. $(f_{n,i})^{0\leq i\leq |vars(\phi_n)|}_{n\geq 0}$ is a $\mathsf{P}^{\#\mathsf{P}}$ family. Finding the value of the cumulator at assignment α can be found by adding a constraint to ϕ that the only acceptable models are less than or equal to α and querying for the number of models.

Now suppose $\mathsf{P}^{\#\mathsf{P}} \subset \mathsf{P/poly}$, then there are polynomial size circuits for each $f_{n,i}$ and thus a polynomial size cumulator ξ_n for each ϕ_n. For each n, $\mathsf{lip}(\xi_n)$ is also polynomial size in ϕ_n. Thus the family $(\mathsf{lip}(\xi_n))_{n\geq 0}$ is super-polynomial lower bound for eFrege. ◀

4 CLIP+eFrege simulates existing #SAT proof systems

In this section, we give an important CLIP+eFrege p-simulation technique for any #SAT proof systems which are closed under restrictions and have short eFrege proofs of the properties of restriction (Definition 15). To be precise, we show that CLIP+eFrege can p-simulate any model counting proof system $\mathcal{P}$ which obey the following conditions:

I. The polynomial-time ability to extract circuits θ from a $\mathcal{P}$-proof π of CNF ϕ over n variables $(x_{n-1} \dots x_0)$ that calculate closure under restrictions for any given (partial) assignment α. That is, $\theta : \alpha \rightarrow \#\mathrm{SAT}(\phi|_{\alpha})$ (Definition 8).

II. $\mathcal{P}$ has short eFrege proofs for properties of restriction (Definition 15) that confirm the correctness of closure under restrictions in $\mathcal{P}$.

Let us formally define the circuit θ used in above conditions.

▶ **Definition 8.** *Let ϕ be a CNF on n variables and α be a (partial) assignment of length $n-i$. We define $\theta^i(\alpha)$ to be a Boolean circuit that returns $\#_{models}(\phi|_\alpha)$. Also, $\theta^i_j(\alpha)$ is a circuit that returns the j^{th} bit of $\theta^i(\alpha)$.*

In the upcoming subsections, we give the complete simulation technique. Recall that CLIP+eFrege proof consists of a cumulator ξ and a eFrege-proof of validity of the propositional "lip" statement which encodes the correctness of ξ. Using the condition-I above, in Section 4.1 we derive the cumulator ξ for ϕ (Part 1 of our simulation technique). In Section 4.2, we use the condition-II to derive the eFrege-proof of lip(ξ) (Part 2 of our simulation technique).

4.1 Simulation Technique (Part 1) : Cumulator Extraction

In this section, we give a general framework of extracting efficiently a cumulator from the proofs of existing propositional model counting proof systems. We need the following definition.

▶ **Definition 9** (Disjoint binary partial assignment cover (Cov(J_1,J_2))). *Let J_1, J_2 be integers representing some complete assignments to variables $X :=< x_{n-1}, ..., x_0 >$ in this order. The cover Cov(J_1,J_2) is a set of partial assignments to X which are non-overlapping and together cover the entire assignment space between J_1 and J_2 inclusive of both.*

For instance, let $X := \{x_3, x_2, x_1, x_0\}$, $J_1 := \mathbf{5}$ and $J_2 := \mathbf{15}$. One possible Cov(J_1,J_2):= $\{\{x_3 = 1\}, \{x_3 = 0, x_2 = 1, x_1 = 1\}, \{x_3 = 0, x_2 = 1, x_1 = 0, x_0 = 1\}\}$. Observe that the first partial assignment (i.e. $\{x_3 = 1\}$) is covering all assignments from $[\boldsymbol{8}, \boldsymbol{15}]$. Similarly the second and third partial assignments are covering the assignments $[\boldsymbol{6}, \boldsymbol{7}]$ and $\mathbf{5}$ respectively. Another possible Cov(J_1,J_2):= $\{\{x_2 = 1, x_0 = 1\}, \{x_3 = 1, x_2 = 0\}, \{x_3 = 1, x_2 = 1, x_0 = 0\}, \{x_3 = 0, x_2 = 1, x_1 = 1, x_0 = 0\}\}$ which cover assignments $\{\boldsymbol{5}, \boldsymbol{7}, \boldsymbol{13}, \boldsymbol{15}\}, \{\boldsymbol{8}, \boldsymbol{9}, \boldsymbol{10}, \boldsymbol{11}\}, \{\boldsymbol{12}, \boldsymbol{14}\}$ and $\{\boldsymbol{6}\}$ respectively.

Let us now outline the general extraction technique.

Cumulator Extraction Technique. Let $\mathcal{P} \in \{$MICE, KCPS(#SAT), CPOG$\}$ be a propositional model counting proof system. Consider a CNF ϕ over n variables and its $\mathcal{P}$-proof π. In order to efficiently extract a correct cumulator ξ for ϕ, we follow the following steps:

1. Show that $\mathcal{P}$ is closed under restrictions (see Definition 1). That is, show that $\mathcal{P}$ obeys condition-I from above.
2. For any complete assignment J to $vars(\phi)$, find the set of non-overlapping partial assignments (to $vars(\phi)$) which cover the entire assignment space from assignment-$\mathbf{0}$ up to assignment-J (i.e Cov($\mathbf{0}$,J) see Definition 9).
   Using Fenwick's idea [15, 26], it is easy to compute Cov($\mathbf{0}$,J) for any complete assignment J (Lemma 10). Moreover, |Cov(0,J)|$\leq n$.
3. For each partial assignment $\alpha \in$ Cov($\mathbf{0}$,J), restrict π with α and consider the $\mathcal{P}$-proof π' of CNF $\phi|_\alpha$. Observe that π' and $\theta^i(\alpha)$ agree on the value of $\#SAT(\phi|_\alpha)$ where $i = n - |\alpha|$. Since $\mathcal{P}$ is closed under restrictions, this step takes $\mathcal{O}(|\pi|)$ time for every α.
4. Finally add the number of models returned by all the π' proofs obtained in the above step. (This step will need a full-adder circuit as integers are represented as $(n+1)$-bit numbers).

This process will return $\xi(J)$ which computes $C_{models}(\phi, J)$ and takes $\mathcal{O}(n.|\pi|)$ time.

We prove Step-1 of our simulation technique individually for existing proof system CPOG in Section 5 (Lemma 23). For Step-2, consider the following lemma.

▶ **Lemma 10.** *Given an input size n, and binary integer $\mathbf{0} \leq J < 2^n$. There is a polynomial time algorithm in n that returns a disjoint binary partial assignment cover for $[\boldsymbol{0}, J]$ ($\mathsf{Cov}(\mathbf{0},J)$) with at most n many partial assignments.*

We have proved Lemma 10 by using a deterministic Fenwick-based algorithm in Appendix A.

Note that extracting the cumulator is not enough for the full CLIP simulation, because CLIP proofs also consist of the validity proof of the lip statement. However, with an access to an NP-oracle, the validity of the lip statement can be obtained in one step due to the correctness of our simulation technique. We call such a system as CLIP^{NP}. Thus the efficient cumulator extraction shows that CLIP^{NP} p-simulates any #SAT system which is closed under restrictions. Observe that CLIP^{NP} is a proof system only if P = NP.

In the upcoming sections, we give full CLIP framework simulations of all the existing #SAT proof systems using the powerful eFrege system for validating the lip statement.

Recall that for the cumulator extraction, we used the concepts of Fenwick assignments. In upcoming proofs, we also need some additional results on Fenwick assignments. We finish this subsection with these results before proceeding to the part 2 of our simulation technique.

4.1.1 Formalising Fenwick Assignments

In Lemma 10, we show how to compute the partial assignment cover of a given complete assignment α with $|\mathsf{Cov}(\mathbf{0},\alpha)| \leq |\alpha|$. In this section, we formalise it as a Boolean circuit (Definition 11) which outputs a partial assignment cover for any given complete assignment α. We call the output $\mathsf{Cov}(\mathbf{0},J)$ of the Boolean circuit as Fenwick assignments. We further show that eFrege can handle a few essential properties of Fenwick assignments. We use these in the next section for part-2 of the simulation technique.

▶ **Definition 11** (Fenwick Assignments). *Let α be a complete assignment to n variables with ordering $< x_{n-1} \dots x_0 >$. For every $i, 0 \leq i \leq n$, we define an existence function $e_i(\alpha)$ and the set of initial assignment bits $f_{i,j}$ for $0 \leq i \leq j < n$ as follows:*

$$e_i(\alpha) = \begin{cases} 1 & \alpha(x_i) \wedge \bigvee_{k\geq 0}^{k<i} \overline{\alpha(x_k)} \text{ and } 0 < i < n \\ 1 & \overline{\alpha(x_i)} \wedge \bigwedge_{k\geq 0}^{k<i} \alpha(x_k) \text{ and } 0 \leq i < n \\ 1 & \bigwedge_{k\geq 0}^{k<n} \alpha(x_k) \text{ and } i = n \\ 0 & \text{otherwise} \end{cases} \qquad f_{i,j}(\alpha) = \begin{cases} 1 & \alpha(x_j) \text{ and } j > i \\ 0 & \text{otherwise} \end{cases}$$

We denote for $0 \leq i < n$, $f_i(\alpha) = \{f_{i,j} | i \leq j < n\}$ as the i^{th} partial assignment for α (note that f_n is the empty assignment and needs no variables to be defined). For a complete assignment α, Fenwick assignments are $\{f_i(\alpha) | e_i(\alpha) = 1, 0 \leq i \leq n\}$. Here, $e_i(\alpha)$ can be seen as a single-bit value that indicates if there is an initial assignment defined on $n - i$ variables in the Fenwick assignments of α. Similarly, $f_{i,j}(\alpha)$ is the value of x_j in the i^{th} partial assignment corresponding to $e_i(\alpha)$ in Fenwick assignments of α.

Below we give an example of how we represent Fenwick assignments as in Definition 11.

▶ **Example 12.** Let $n = 4$ and $J = \mathbf{12}$. Let the variables be lexicographical ordered as $< x_3, \dots, x_0 >$. The corresponding $\mathsf{Cov}(\mathbf{0},\mathbf{12})$ is the following set of partial assignments: $\Big\{\{x_3 = 1, x_2 = 1, x_1 = 0, x_0 = 0\},\ \{x_3 = 1, x_2 = 0\},\ \{x_3 = 0\}\Big\}$.

The Fenwick circuits have the following values: $e_0(\mathbf{12}) = 1,\ e_1(\mathbf{12}) = 0,\ e_2(\mathbf{12}) = 1,\ e_3(\mathbf{12}) = 1,\ e_4(\mathbf{12}) = 0$. This indicates that there are 3 partial Fenwick assignments in $\mathsf{Cov}(\mathbf{0},\mathbf{12})$ ending at x_0, x_2 and x_3 respectively. The exact assignments are computed as follows:

$f_{0,3}(\mathbf{12}) = 1,\ f_{0,2}(\mathbf{12}) = 1,\ f_{0,1}(\mathbf{12}) = 0,\ f_{0,0}(\mathbf{12}) = 0 \quad \rightarrow \{x_3 = 1, x_2 = 1, x_1 = 0, x_0 = 0\}$
$f_{2,3}(\mathbf{12}) = 1,\ f_{2,2}(\mathbf{12}) = 0 \quad \rightarrow \{x_3 = 1, x_2 = 0\}$
$f_{3,3}(\mathbf{12}) = 0 \quad \rightarrow \{x_3 = 0\}$

We will be able to prove few properties of how Fenwick assignments change as $num(\alpha)$ increases slowly (see Appendix B). Step 3,4 of our simulation technique require restricting the θ circuit with all the Fenwick assignments of some complete assignment α and adding them up to get $\xi(\alpha)$. Using the formal definition of the Fenwick assignments from Definition 11, we have the following.

▶ **Definition 13.** *For a complete assignment α on n variables, we define the vector of Boolean variables $\xi(\alpha)$ as the following sum:*

$$(e_n(\alpha) \wedge \theta^n(f_n(\alpha))) + (e_{n-1}(\alpha) \wedge \theta^{n-1}(f_{n-1}(\alpha))) + \cdots + (e_0(\alpha) \wedge \theta^0(f_0(\alpha)))$$

This circuit ξ is the required cumulator.

4.2 Simulation Technique (Part 2): eFrege certification of the cumulator

Recall that, for a full CLIP+eFrege simulation, the proof system must have short eFrege proofs of the properties of restriction, i.e. condition-II from Section 4. Let us formally define the properties of restriction in Definition 15 which are based on the following simple observations of partial assignments.

▶ **Observation 14.** *Let ϕ be a CNF on variables $< x_{n-1} \dots x_0 >$ (used in that lexicographic ordering in CLIP). Let α be a partial assignment defined on $x_{n-1} \dots x_i$, undefined on $x_{i-1} \dots x_0$. Given a $\{0,1\}$-value b, let $\alpha_b := \alpha \cup \{x_{i-1} = b\}$. Then, $\#_{models}(\phi|_\alpha) = \#_{models}(\phi|_{\alpha_0}) + \#_{models}(\phi|_{\alpha_1})$. If α is a complete assignment, $\#_{models}(\phi|_\alpha) = \mathbb{1}_\phi(\alpha)$.*

▶ **Definition 15** (Properties of Restriction). *Let S be a propositional model counting proof system and ϕ be a CNF on n variables ($< x_{n-1} \dots x_0 >$). Suppose ϕ has an S-proof π with θ being the associated circuit for restriction. We consider the following properties for all assignments α over $x_{n-1} \dots x_0$.*

1. *If α is a complete assignment: $\theta^0(\alpha) = \mathbb{1}_\phi(\alpha)$.*
2. *If α is a partial assignment defined on $x_{n-1} \dots x_i$: $\theta^i(\alpha) = \theta^{i-1}(\alpha_0) + \theta^{i-1}(\alpha_1)$*

▶ **Observation 16.** *Any propositional model counting proof system $\mathcal{P}$ which is closed under restrictions, satisfies the properties of restriction mentioned in Definition 15.*

Next we prove that short eFrege proofs of the properties of restriction can be used to give short eFrege-proofs of the lip statement from Definition 5.

▶ **Lemma 17.** *Suppose $\mathcal{P}$ is a propositional model counting proof system which is closed under restrictions. Let ϕ be a CNF and ξ be a cumulator obtained by using Fenwick assignments on the $\mathcal{P}$-proof of ϕ. If $\mathcal{P}$ has polynomial-sized eFrege proofs of the properties of restriction, then it has short eFrege proof of lip(ξ).*

Before presenting the detailed proof of Lemma 17, we briefly give the proof idea of it. For a CNF ϕ and any two consecutive assignments $\beta^2 = \beta^1 + 1$, we need to show that $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$. We show this in the following two cases: β^2 being odd or even. In the case of β^1=odd and β^2=even, we use the property of Fenwick assignments (Lemma 37, see Appendix B) that the Fenwick assignments of β^2 are the Fenwick assignment of β^1 and β^2. This directly implies $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$.

In the case of β^1 is even and β^2 is odd, we also relate the Fenwick assignments of the two total assignments. We use the property of Fenwick assignments (Lemma 38, see Appendix B) that the Fenwick assignment of β^2 is some common set of assignments β^* and a single partial assignment γ, γ is the common prefix of β^1 and β^2. We then show that using Observation 16 a linear number of times, we can prove that $\theta(\gamma)$ decomposes so that it implies that $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$.

Proof. Line-1 of the lip statement assumes the assignment $\alpha = \mathbf{0}$. From a simple computation, only $e_0(\mathbf{0}) = 1$ with the corresponding $f_0(\mathbf{0}) = \mathbf{0}$. From Definition 13, $\xi(\mathbf{0}) = \theta^0(\mathbf{0})$. From the properties of restriction $\theta^0(\mathbf{0}) = \mathbb{1}_\phi(\mathbf{0})$.

Line-3 of the lip statement assumes the assignment $\alpha = \mathbf{2^n} - \mathbf{1}$. Similarly, we can compute that only $e_n(\mathbf{2^n} - \mathbf{1}) = 1$ and $f_n(\mathbf{2^n} - \mathbf{1}) = \emptyset$. Then $\xi(\mathbf{2^n} - \mathbf{1}) = \theta^n(\emptyset)$. θ^n contains no restriction, so it is the intended answer for the entire model count.
The main part of CLIP is the inductive step for assignments $\beta^2 = \beta^1 + \mathbf{1}$.

For an even β^2, We can use the cases for short proofs in Lemma 38 (see Appendix B). We can argue that $e_i(\beta^1) \leftrightarrow e_i(\beta^2)$ and $e_i(\beta^1) \rightarrow (f_{i,j}(\beta^1) \leftrightarrow f_{i,j}(\beta^2))$ for $i > 0$. That is, all Fenwick assignments for β^1 and β^2 are the same except the one corresponding to $e_0(\beta^2)$. Additionally that $f_0(\beta^2) = \beta_2$. Along with associativity, this easily leads to $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$ in eFrege.

For an odd β^2, we can use the short proofs from Lemma 38 of the various cases (Appendix B). We take the maximum $p : 0 \leq p \leq n$ such that $\bigwedge_{j<p}^{j\geq 0} \beta^2_j$. We can derive (case-3a,3b) $e_j(\beta^1) \leftrightarrow e_j(\beta^2)$ and $f_{j,k}(\beta^1) \leftrightarrow f_{j,k}(\beta^2)$ for $j > p$. At $j = p$, we have that (case-2a,2b) $f_p(\beta^2) =$ the common prefix of β^1 and β^2 up to p (say γ). For $j < p$, we derive (case-1b) that γ is the prefix of all these Fenwick assignments. Further we show (case-1c,1d) that these assignments extend γ by $p - j - 1$ number of 1s and end with a 0. That is, $\{f_j(\beta^1)\}_{p>j\geq 0} = \gamma_0, \gamma_{10}, \gamma_{110}, \dots, \beta^1$. Using the split property of restrictions for i times, we have $\theta(\gamma) = \sum_{j\leq p}^{j\geq 0}(e_j(\beta^1) \wedge \theta^j(f_j(\beta^1))) + \theta^0(\beta^2)$. Adding $f_{j,k}(\beta^1) \leftrightarrow f_{j,k}(\beta^2)$ for $j > p$ to the above gives us $\xi(\beta^2) = \xi(\beta^1) + \mathbb{1}_\phi(\beta^2)$. ◀

Next, we give a supplementary example of Lemma 17.

▶ **Example 18.** Let a CNF ϕ be defined on $n = 5$ variables and β^1, β^2 be $10010, 10011$ (i.e **18**, **19**) respectively. Let the variables be lexicographical ordered as $< x_4 \dots x_0 >$. The corresponding Cov(**0**,**18**) and Cov(**0**,**19**) are: $\Big\{\{x_4 = 1, x_3 = 0, x_2 = 0, x_1 = 1, x_0 = 0\},\ \{x_4 = 1, x_3 = 0, x_2 = 0, x_1 = 0\},\ \{x_4 = 0\}\Big\}$ and $\Big\{\{x_4 = 1, x_3 = 0, x_2 = 0\},\ \{x_4 = 0\}\Big\}$ respectively.

The first zero of β^2 occurs for the second digit therefore we take $p = 2$. For $j > p$, clearly $e_4(\beta^1) = e_4(\beta^2)$ and $f_4(\beta^1) = f_4(\beta^2)$. There is one partial assignment that is in β^2 but not in β^1, it is $f_2(\beta^2)$ (i.e $\{x_4 = 1, x_3 = 0, x_2 = 0\}$). This is also the common prefix γ of both β^1 and β^2. Using the split property on γ twice, we have the following: $\theta^2(\gamma) = \theta^1(\gamma_0) + \theta^1(\gamma_1) = \theta^1(\gamma_0) + \theta^0(\gamma_{10}) + \theta^0(\gamma_{11})$. Adding $f_4(\beta^1) = f_4(\beta^2)$ to this implies that $\xi(\mathbf{19}) = \xi(\mathbf{18}) + \mathbb{1}_\phi(\mathbf{19})$.

For a model counting proof system $\mathcal{P}$ and a CNF ϕ with it's $\mathcal{P}$-proof π, we have a cumulator ξ from part 1 of our simulation technique. In Lemma 17, we also have an eFrege proof of lip(ξ). Therefore, we have the following.

▶ **Theorem 19.** *CLIP+eFrege p-simulates any model counting proof system which is closed under restrictions and has short eFrege proofs of the properties of restriction.*

In conclusion, for any propositional model counting proof system $\mathcal{P}$, part 2 of our simulation technique consists of the following step:

5. Show that $\mathcal{P}$ has short eFrege-proofs of the two properties of restriction (Definition 15). That is, show that $\mathcal{P}$ obeys condition-II from above.

5 CLIP framework simulation of CPOG

In this section, we apply our simulation technique to the CPOG (Certified Partitioned-Operation Graphs) [4] proof system. That is, we show that CPOG is closed under restrictions (Lemma 23) and admits easy eFrege proofs of the properties of restriction (Lemma 29). For a CNF formula ϕ, CPOG is a propositional weighted-model counting proof system which consists of a POG structure G (Definition 20) along with Resolution proofs of $\phi \leftrightarrow G$. It is well known that model counting is easy for POG [4, p. 16]. However, given a POG G and a CNF ϕ, verifying that $G \equiv \phi$ is hard. In this paper, we only study CPOG for the unweighted (standard) model counting. The missing proofs of this section are included in Appendix C.

The proofs in this section need to prove basic properties of arithmetic in short eFrege proofs which are considered to be academic folklore.

We first define the POG structure (Definition 20) and the CPOG proof system (Definition 22) which is based on POG from [4]. For an example of POG see [5].

▶ **Definition 20** (POG [4]). *A Partitioned-Operation Graph (POG) (say G) is a directed acyclic graph defined on n variables (say X). Each node v in a POG has an associated dependency set $\mathcal{D}(v) \subseteq X$ and a set of models $\mathcal{M}(v)$, consisting of all complete assignments that satisfy the formula represented by the POG rooted at v. The leaf nodes (with outdegree $= 0$) can be of the following:*

- *Boolean constants 0 or 1. Here, $\mathcal{D}(1) = \mathcal{D}(0) = 0$, $\mathcal{M}(0) = \emptyset$ and $\mathcal{M}(0) = \langle X \rangle$.*
- *Literal l for some variable x such that $vars(l) = x \in X$. Here, $\mathcal{D}(l) = x$, $\mathcal{M}(l) = \{\alpha \in \langle X \rangle \mid \alpha(x) \equiv l\}$.*

The rest of the nodes (internal nodes) can be of the following:

- *Decomposable AND-gate ($\wedge^p$) [1] with outgoing edges to $v_1, \dots, v_k$ for $k > 1$. Here, $\mathcal{D}(\wedge^p) = \bigcup_{1 \leq i \leq k} \mathcal{D}(v_i)$ and $\mathcal{M}(\wedge^p) = \bigcap_{1 \leq i \leq k} \mathcal{M}(v_i)$. This node needs to follow the decomposable property namely, $\mathcal{D}(v_i) \cap \mathcal{D}(v_j) = \emptyset$ for every $i, j \in [k]$ and $i \neq j$.*
- *Deterministic OR-gate ($\vee^p$) with outgoing edges to v_1, v_2. Here, $\mathcal{D}(\vee^p) = \mathcal{D}(v_1) \cup \mathcal{D}(v_2)$ and $\mathcal{M}(\vee^p) = \mathcal{M}(v_1) \cup \mathcal{M}(v_2)$. This node needs to follow the deterministic property namely, $\mathcal{M}(v_1) \cap \mathcal{M}(v_2) = \emptyset$.*

The edges of G have an optional polarity to indicate if they need to be negated (polarity $= 1$) or not (polarity $= 0$). Here, $\mathcal{D}(\neg v) = \mathcal{D}(v)$ and $\mathcal{M}(\neg v) = \langle X \rangle - \mathcal{M}(v)$. Every POG has a designated root node r with indegree $= 0$ and models of the POG would be $\mathcal{M}(r)$.

The weighted model counting can be seen as a ring-evaluation problem for a commutative ring over rational numbers $\in [\mathbf{0}, \mathbf{1}]$. The ring evaluation problem takes a weight function $w(x) \in [\mathbf{0}, \mathbf{1}]$ for all variables $x \in X$ and computes the following:

$$R(v, w) = \Sigma_{\alpha \in \mathcal{M}(v)} \ \Pi_{l \in \alpha} \ w(l) \qquad (1)$$

where $w(\overline{x}) = \mathbf{1} - w(x)$. For standard unweighted model-counting (i.e $|\mathcal{M}(v)|$), one can fix $w(x) = \frac{\mathbf{1}}{\mathbf{2}}$ for all $x \in X$ and $|\mathcal{M}(v)| = \mathbf{2}^{|\mathbf{X}|} \cdot R(v, w)$. The following properties of the ring evaluation function are well known.

[1] For simplicity, we use the same notations from [4]. Here, p stands for partitioned-operation formulas.

▶ **Proposition 21** ([4]). *Ring evaluations for operations $\neg$, $\wedge^p$ and $\vee^p$ satisfies the following for any weight function w: (i) $R(\neg v, w) = \mathbf{1} - R(v, w)$, (ii) $R(\bigwedge_{1\leq i\leq k}^{p} v_i, w) = \prod_{1\leq i\leq k} R(v_i, w)$, (iii) $R(v_1 \vee^p v_2, w) = R(v_1, w) + R(v_2, w)$.*

For a CNF ϕ, a CPOG proof π consists of a POG G such that $G \equiv \phi$. However, to make the proof easily verifiable, π explicitly has the proof that G is a POG and is equivalent to ϕ. We present the precise definition from [2] below.

▶ **Definition 22** (CPOG [2, 4]). *A* CPOG *proof π of ϕ is the tuple $(\mathcal{E}(G), \delta, \rho, \psi)$, where*

- *G is a POG such that $G \equiv \phi$ and $\mathcal{E}(G)$ is a clausal encoding of the POG G by defining an extension variable for every internal node of G.*
- *δ is the determinism proof for OR-gates which contains a Resolution proof of $\mathcal{E}(G) \wedge (v_1) \wedge (v_2)$ for every $\vee^p$-gate v with outgoing edges to v_1, v_2.*
- *ρ is the forward implication proof (i.e. $\phi \models G$) consisting of a Resolution proof of $\mathcal{E}(G) \wedge \phi \wedge (\overline{r})$.*
- *ψ is the reverse implication proof (i.e. $G \models \phi$) consisting of a Resolution proof of $\mathcal{E}(G) \wedge (r) \wedge \overline{C}$ for every clause $C \in \phi$.*

Observe that extension variables used in CPOG are restrictive as compared to those in eFrege.

▶ **Lemma 23.** CPOG *is closed under restrictions.*

This proof is fairly simple and we include it in Appendix C. This completes Step 1 of our simulation technique for CPOG. Thus we have the following:

▶ **Corollary 24.** *There is a polynomial time method of extracting a cumulator circuit from a* CPOG *proof.*

Now we are ready to prove that CPOG admits easy eFrege proofs for the properties of restriction. Recall, the weight function w is defined for all the variables (X) of POG G as $\mathbf{\frac{1}{2}}$. In this case, the value of $R(r, w) = \mathbf{2}^{|\mathbf{X}|} \cdot |\mathcal{M}(r)|$. For an assignment α, conditioning the POG with α (Lemma 23) will give G' and the following will hold $R(r', w) = \mathbf{2}^{|\mathbf{X} - |\alpha||} \cdot |\mathcal{M}(r)|_\alpha|$.

Instead of changing the POG structure, we change the weight function as defined below to obtain the same model count as above i.e. $R(r, w_\alpha) = \mathbf{2}^{|\mathbf{X} - |\alpha||} \cdot |\mathcal{M}(r)|_\alpha|$.

▶ **Definition 25.** *Given a CNF ϕ and an initial assignment α defined on $x_{n-1} \ldots x_i$ and undefined on $x_{i-1} \ldots x_0$, we define the weight w_α which weighs variables according to the following*

$$w_\alpha(x_j) = \begin{cases} \mathbf{1} & j \geq i \ \&\ \alpha(x_j) = 1, \\ \mathbf{0} & j \geq i \ \&\ \alpha(x_j) = 0, \\ \mathbf{\frac{1}{2}} & j < i. \end{cases}$$

Next, in Lemma 26, 27, 28, we prove some properties of $R(v, w_\alpha)$ for every node v of the POG recursively. We use the general properties of the ring function from Proposition 21 in these proofs. Informally, in Lemma 26, we show that if α was undefined on x and x is not in the dependency set of v (i.e. $x \notin \mathcal{D}(v)$), the value of R does not change when weight function is changed to w_{α_0} or w_{α_1} where $\alpha_b = \alpha \cup \{x = b\}$. In Lemma 27, we show that if x is in $\mathcal{D}(v)$, the values of R hold a weaker relation of $R(v, w_\alpha) = \mathbf{\frac{1}{2}}(R(v, w_{\alpha_0}) + R(v, w_{\alpha_1}))$. We consider complete assignments α in Lemma 28 and prove that the function $R(v, w_\alpha)$ returns $\mathbf{1}$ if α is a satisfying assignment of the POG rooted at v and $\mathbf{0}$ otherwise. We use these Lemmas to prove that CPOG has easy eFrege proofs of the properties of restriction in Lemma 29. The detailed proofs of these lemmas are pushed to Appendix C.

▶ **Lemma 26.** *Let α be an initial partial assignment defined up to x_i where $i > 0$. We can prove in the structure of the POG that $R(v, w_\alpha) = R(v, w_{\alpha_0}) = R(v, w_{\alpha_1})$ when x_{i-1} is not in the dependency set of v. This proof can be formalised in a short* eFrege *proof.*

▶ **Lemma 27.** *Let α be an initial partial assignment defined up to x_i where $i > 0$. We can prove in the structure of the POG that $R(v, w_\alpha) = \frac{\mathbf{1}}{\mathbf{2}} \cdot (R(v, w_{\alpha_0}) + R(v, w_{\alpha_1}))$. Furthermore we can formalise this in short* eFrege *proofs.*

▶ **Lemma 28.** *For complete assignment α, we can prove using* eFrege *in the structure of the POG that $R(v, w_\alpha) = \mathbb{1}_v(\alpha)$.*

▶ **Lemma 29.** *CPOG has short eFrege proofs of $\theta(\alpha) = \mathbb{1}_\phi(\alpha)$, when α is a complete assignment, and $\theta(\alpha) = \theta(\alpha_0) + \theta(\alpha_1)$, when α is strictly initial and partial.*

Proof. We define $\theta^i(\alpha) = \mathbf{2^i} \cdot R(r, w_\alpha)$. Using Lemma 27 we can show for the root node r that $R(r, w_\alpha) = \frac{\mathbf{1}}{\mathbf{2}} \cdot (R(r, w_{\alpha_0}) + R(r, w_{\alpha_1}))$ when α is initial and strictly partial. This proves the second property of restriction.

Using Lemma 28 we can show that $R(r, \alpha) = \mathbb{1}_r(\alpha)$ when α is complete. Hence $\theta^0(\alpha) = \mathbb{1}_r(\alpha)$. Here for the first property of restriction, we still need to prove that $\mathbb{1}_r(\alpha) = \mathbb{1}_\phi(\alpha)$. For this, we use the Resolution proofs for $r \leftrightarrow \phi$ in the CPOG proof. Since eFrege p-simulates Resolution, these are easily converted to show $\mathbb{1}_r(\alpha) = \mathbb{1}_\phi(\alpha)$. ◀

This proves Step 5 of our simulation technique for CPOG. Therefore from our simulation technique part 1 and 2, we have the following.

▶ **Theorem 30.** CLIP+DRAT *p-simulates* CPOG*.*

In [2], the authors prove that CPOG is strictly stronger than the other existing proof systems (i.e. MICE, KCPS(#SAT)). Therefore we have the following.

▶ **Corollary 31.** CLIP+DRAT *p-simulates* MICE *and* KCPS(#SAT)*.*

6 Exponential Improvement on Existing #SAT proof systems

In this section, we give easy CLIP+eFrege proofs for hard formulas of existing proof systems. Below, we give easy proofs of XOR-PAIRS in CLIP+eFrege system (Theorem 34). These formulas were previously proven to be hard for MICE [3, Theorem 23]. Later in Corollary 36, we give easy CLIP+eFrege proofs of some unsatisfiable formulas which are hard in MICE and KCPS(#SAT).

▶ **Definition 32** (XOR-PAIRS [3])**.** *Let $X = \{x_1, \dots x_n\}$ and $Z = \{z_{1,1}, z_{1,2} \dots, z_{n,n-1}, z_{n,n}\}$.*
$C^1_{ij} = (x_i \vee x_j \vee \bar{z}_{ij}), C^2_{ij} = (\bar{x}_i \vee x_j \vee z_{ij}), C^3_{ij} = (x_i \vee \bar{x}_j \vee z_{ij}), C^4_{ij} = (\bar{x}_i \vee \bar{x}_j \vee \bar{z}_{ij})$
$\phi(X, Z)$ contains $C^1_{ij}, C^2_{ij}, C^3_{ij}, C^4_{ij}$ for $i, j \in [n]$.

The models of XOR-PAIRS are the assignments where $z_{i,j} = (x_i \oplus x_j)$ for all $i, j \in [n]$. Hence, $\#_{\text{models}}$(XOR-PAIRS) $= 2^n$. The family XOR-PAIRS is hard for proof systems MICE [3, Theorem 23]. We will show in Theorem 34 that these formulas are easy in CLIP+eFrege.

▶ **Definition 33.** *Fix an input length n, and let γ and δ be vectors of n variables. For pairs of individual variables a, b, use $a = b$ to denote $(\neg a \vee b) \wedge (\neg b \vee a)$. We can encode polynomial size propositional circuits: $L(\gamma, \delta)$, that denotes $num(\gamma) < num(\delta)$.*

▶ **Theorem 34.** CLIP+eFrege *has short proofs of XOR-PAIRS*

Proof. First we fix that all Z-bits are less significant than all X-bits, otherwise the cumulative function is affected by the variable ordering. We begin by arguing that the cumulative function for XOR-PAIRS is easy to compute. This comes from the fact the truth function itself behaves in a way that makes it amenable to counting, it only ever increases by one, once for each complete assignment to X. There is a function $p : 2^X \to 2^Z$ that maps the binary assignment α on X to the unique assignment in Z such that $\phi(\alpha, p(\alpha))$ for every α. We can construct a multi-output circuit P (a sequence of circuits $P_{i,j}$ for $i, j \in \{|X|\}$) for p, easily through $O(Z)$ many gates: $P_{i,j}(X) = (x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j)$.

We then express the cumulative function in a cumulator circuit that we will use for CLIP.

$$\xi(\alpha, \beta) = \begin{cases} \alpha & \beta < P(\alpha) \\ \alpha + \mathbf{1} & \beta \geq P(\alpha) \end{cases}$$

Note that since $\xi(\alpha, \beta)$ outputs in binary we can actually express each digit as a Boolean circuit.

Now we have to argue why the remaining propositional proof is easy for eFrege. This is basically a number of tautological implications we have to show individually. The idea is to break each implication into a number of cases. Case analysis is typically easy for eFrege as it is just resolving with a disjunction of possibilities.

Base case. If $A_X = \mathbf{0}$, $P(A_X)$ always evaluates to $\mathbf{0}$. If A_Z is also $\mathbf{0}$, $\phi(A_X, A_Z)$ evaluates to true, while $L(A_Z, P(A_X))$ evaluates to false (because of strictness). This makes $\xi(\alpha, \beta)$ evaluate to the integer $\mathbf{1}$ (in other words $\xi(\alpha, \beta)_i = \mathbf{1}$ if and only if $i = n$). Each of these evaluations are shown in eFrege through the extension clauses. These will satisfy the two disjunctions that use the base case.

Inductive Step. Here we firstly argue that $\phi(B_X, B_Z) \leftrightarrow E(B_Z, P(B_X))$ has a short eFrege proof. We show that for each pair i, j the four clauses are implied by $(x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j) \leftrightarrow z_{i,j}$. And then we show the four clauses show the truth table for$(x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j) \leftrightarrow z_{i,j}$. The proof size is linear. If $B_X = A_X$ and $B_Z = A_Z + \mathbf{1}$, we make 3 cases.

1. Let $B_Z = P(B_X)$, we can get a short eFrege proof of $\neg L(B_Z, P(B_X))$ and $L(A_Z, P(A_X))$, and thus a proof of $T(\xi(B_X, B_Z), B_X)$ and $E(\xi(A_X, A_Z), A_X)$. We use $B_X = A_X$ to show $T(\xi(B_X, B_Z), \xi(A_X, A_Z))$. $\phi(B_X, B_Z) \leftrightarrow E(B_Z, P(B_X))$ is a proven tautology.
2. Let $B_Z < P(B_X)$, we get a short eFrege proof that $L(A_Z, P(A_X))$ is true, and thus a proof that $E(\xi(B_X, B_Z), B_X)$ and $E(\xi(A_X, A_Z), A_X)$. We use $B_X = A_X$ to show $E(\xi(B_X, B_Z), \xi(A_X, A_Z))$. $\phi(B_X, B_Z)$ falls into provable contradiction with $L(B_Z, P(B_X))$ by showing a bit must be different.
3. Let $B_Z > P(B_X)$, we can get a short eFrege proof of $\neg L(B_Z, P(B_X))$ and $\neg L(A_Z, P(A_X))$, and thus that $T(\xi(B_X, B_Z) = B_X + \mathbf{1})$ and $T(\xi(A_X, A_Z) = A_X + \mathbf{1})$. We use $B_X = A_X$ to show $E(\xi(B_X, B_Z), \xi(A_X, A_Z))$. $\phi(B_X, B_Z)$ contradicts $L(P(B_X), B_Z)$.

Now consider $||B_X = A_X + \mathbf{1}||$, $||B_Z = \mathbf{0}||$ and $||A_Z = \mathbf{2}^{|\mathbf{Z}|} - \mathbf{1}||$. Part of the trichotomy is impossible. We can prove $L(P(B_X), B_Z)$ fails when $||B_Z = \mathbf{0}||$. For the remaining cases we firstly prove that $\neg||\mathbf{2}^{|\mathbf{Z}|} - \mathbf{1} < P(A_X)||$ which is proven from the fact that one digit must be 0 to be less than. Therefore $\xi(A_X, A_Z) = A_X + \mathbf{1}$ in both cases.

1. Let $B_Z = P(B_X)$ then we can get a short eFrege proof that $L(B_Z, P(B_X))$ is false and so $\xi(B_X, B_Z) = B_X + \mathbf{1} = A_X + \mathbf{1} + \mathbf{1} = \xi(A_X, A_Z) + \mathbf{1}$. We can find an equality proof here. $\phi(B_X, B_Z) \leftrightarrow E(B_Z, P(B_X))$ is a tautology.

2. Let $L(B_Z, P(B_X))$ be true so $\xi(B_X, B_Z) = B_X = A_X + \mathbf{1} = \xi(A_X, A_Z)$. $\phi(B_X, B_Z)$ falls into provable contradiction with $L(B_Z, P(B_X))$.

   For the final case, we use $\neg||\mathbf{2}^{|\mathbf{Z}|} - \mathbf{1} < P(A_X)||$, hence $\xi(\mathbf{2}^{|\mathbf{X}|} - \mathbf{1}, \mathbf{2}^{|\mathbf{Z}|} - \mathbf{1}) = \mathbf{2}^{|\mathbf{X}|}$. ◀

Let us briefly discuss about unsatisfiable formulas. That is, CNF formulas for which the model counts are $\mathbf{0}$. In [2], it has been shown that for unsatisfiable formulas, KCPS(#SAT) is p-equivalent to regular Resolution [2, Proposition 5.1] and MICE is p-equivalent to Resolution [2, Proposition 5.3]. In this paper, we observed the following for the unsatisfiable CNF formulas:

▶ **Proposition 35.** *For unsatisfiable formulas ϕ, if ϕ has an* eFrege *proof π of unsatisfiability, then ϕ has a* CLIP+eFrege *proof of linear size w.r.t. $|\pi|$.*

Proof. For a unsatisfiable CNF ϕ, assume that it has an easy eFrege-proof of unsatisfiability. We can have an easy CLIP+eFrege proof of ϕ as follows: The cumulator ξ for ϕ is a trivial circuit that only outputs "$\mathbf{0}$" for any input. For any two consecutive assignments i.e $\beta_2 = \beta_1 + \mathbf{1}$, the inductive statement of lip encodes that $\xi(\beta_2) = \xi(\beta_1) + \mathbb{1}_\phi(\beta_2)$. Therefore, the eFrege proof of lip statement needs only the unsatisfiability proof of ϕ (i.e. $\mathbb{1}_\phi(\beta_2) = \mathbf{0}$). ◀

This gives more separation results for unsatisfiable formulas which are hard for Resolution but easy for eFrege. That is, we have the following:

▶ **Corollary 36.** *The unsatisfiable formulas, PHP, clique-color and Random Parity have polynomial-size proofs [11, 6, 10] in* CLIP+eFrege *but require exponential-size proofs in [19, 22, 9]* MICE *and* KCPS(#SAT)*.*

Note that the Clique-coloring principle [22, Definition 7.1] is well studied in proof complexity. Informally, it encodes that if a graph G has a clique of size k, then G needs at least k colors. PHP is the famous Pigeon hole principle which encodes that if there are n pigeons and $n - 1$ holes, at least one hole has more than one pigeon in it. Random Parity formulas are contradictions expressing both the parity and non-parity on a set of variables.

7 Conclusion

We have introduced the CLIP framework for propositional model counting. We have demonstrated the advantages CLIP has by having an unrestricted underlying circuit format. Our approach here has been theoretical and no version of CLIP has been implemented.

The main checking task in CLIP proofs can use existing tools in SAT such as DRAT-trim [28]. We have given a p-simulation of all other #SAT proof systems, in theory this can be used to extract CLIP+eFrege (or CLIP+DRAT) proofs from #SAT solvers. However the number of arithmetic lemmas may make the complete programming of the extractor a difficult task. It could be compensated with assistance from a certifying SAT solver.

Future work should take into account weighted and projected model counting.

References

1 Olaf Beyersdorff, Ilario Bonacina, Leroy Chew, and Ján Pich. Frege systems for quantified boolean logic. *J. ACM*, 67(2):9:1–9:36, 2020. doi:10.1145/3381881.

2 Olaf Beyersdorff, Johannes Klaus Fichte, Markus Hecher, Tim Hoffmann, and Kaspar Kasche. The relative strength of #sat proof systems. In *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*, volume 305 of *LIPIcs*, pages 5:1–5:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.SAT.2024.5.

3 Olaf Beyersdorff, Tim Hoffmann, and Luc Nicolas Spachmann. Proof complexity of propositional model counting. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPIcs*, pages 2:1–2:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.2.

4 Randal E Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Certified knowledge compilation with application to verified model counting. In *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*, 2023. doi:10.4230/LIPICS.SAT.2023.6.

5 Randal E. Bryant, Wojciech Nawrocki, Jeremy Avigad, and Marijn J. H. Heule. Supplement to certified knowledge compilation with application to verified model counting, 2023. URL: https://zenodo.org/records/7966174.

6 Sam Buss and Neil Thapen. DRAT and propagation redundancy proofs without new variables. *Log. Methods Comput. Sci.*, 17(2), 2021. URL: https://lmcs.episciences.org/7400.

7 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Log.*, 163(7):906–917, 2012. doi:10.1016/J.APAL.2011.09.009.

8 Florent Capelli. Knowledge compilation languages as proof systems. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 90–99. Springer, 2019. doi:10.1007/978-3-030-24258-9_6.

9 Leroy Chew, Alexis de Colnet, Friedrich Slivovsky, and Stefan Szeider. Hardness of random reordered encodings of parity for resolution and CDCL. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014*, pages 7978–7986. AAAI Press, 2024. doi:10.1609/AAAI.V38I8.28635.

10 Leroy Chew and Marijn J. H. Heule. Sorting parity encodings by reusing variables. In *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2020. doi:10.1007/978-3-030-51825-7_1.

11 Stephen A Cook. A short proof of the pigeon hole principle using extended resolution. *Acm Sigact News*, 8(4):28–32, 1976. doi:10.1145/1008335.1008338.

12 Stephen A Cook and Robert A Reckhow. The relative efficiency of propositional proof systems. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 173–192. ACM, 2023. doi:10.1145/3588287.3588299.

13 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discret. Appl. Math.*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.

14 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001. doi:10.1145/502090.502091.

15 Peter M. Fenwick. A new data structure for cumulative frequency tables. *Softw. Pract. Exp.*, 24(3):327–336, 1994. doi:10.1002/SPE.4380240306.

16 Johannes K Fichte, Markus Hecher, and Valentin Roland. Proofs for propositional model counting. In *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, 2022. doi:10.4230/LIPICS.SAT.2022.30.

17 Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache der reinen Denkens, Halle*. Lubrecht & Cramer, 1879. English translation in: from Frege to Gödel, a source book in mathematical logic (J. van Heijenoord editor), Harvard University Press, Cambridge 1967.

18 Joshua A. Grochow and Toniann Pitassi. Circuit complexity, proof complexity, and polynomial identity testing: The ideal proof system. *J. ACM*, 65(6), November 2018. doi:10.1145/3230742.

19 Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985. doi:10.1016/0304-3975(85)90144-6.

20 Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press, New York, NY, USA, 1995.

21 Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn JH Heule. Extended resolution simulates DRAT. In *Automated Reasoning: 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, pages 516–531. Springer, 2018. doi:10.1007/978-3-319-94205-6_34.

22 Jan Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997. doi:10.2307/2275541.

23 Meena Mahajan and Gaurav Sood. QBF merge resolution is powerful but unnatural. In *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.SAT.2022.22.

24 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

25 John Alan Robinson. Theorem-proving on the computer. *Journal of the ACM*, 10(2):163–174, 1963. doi:10.1145/321160.321166.

26 Boris Ryabko. A fast on-line adaptive code. *IEEE Trans. Inf. Theory*, 38(4):1400–1404, 1992. doi:10.1109/18.144725.

27 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

28 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.

A Missing proof and algorithm from Section 4.1

▶ **Lemma 10.** *Given an input size n, and binary integer $\mathbf{0} \leq J < 2^n$. There is a polynomial time algorithm in n that returns a disjoint binary partial assignment cover for $[\mathbf{0}, J]$ (Cov($\mathbf{0}$,J)) with at most n many partial assignments.*

Proof. Refer to Algorithm 1 for the exact procedure. The correctness of Algorithm 1 stems from the correctness of Fenwick trees [15] which efficiently computes the cumulative sum of numbers stored between any two indices of the array by visiting atmost logarithmic indices in the tree. Algorithm 1 similarly computes Cov($\mathbf{0}$,J) by finding the least number of partial assignments to include such that they cover the entire range from $[\mathbf{0}, J]$ i.e. $log(2^n) = n$. ◀

B Missing lemmas and proofs from Section 4.1.1

▶ **Lemma 37.** *Assume $T(Y, X)$ (i.e. $Y = X + 1$) and $\neg Y_0$ then*

1. $e_0(Y) \wedge \neg e_0(X)$
2. *For $i > 0$, $e_i(X) \leftrightarrow e_i(Y)$*
3. *For $0 < i \leq j < n$, $e_i(X) \rightarrow (f_{i,j}(X) \leftrightarrow f_{i,j}(Y))$*
4. *For $0 \leq j < n$, $f_{0,j}(Y) \leftrightarrow Y_j$*

And we can prove these formally in eFrege *in short proofs even in the case that Y and X are vectors of variables (or extension variables).*

Proof. $\bar{Y}_0 \wedge \bigwedge_{k\geq 0}^{k<0} Y_k$ is true hence $e_0(Y)$ can be shown via definition. Since, $Y_0 = X_0 \oplus \bigwedge_{k\geq 0}^{k<0} X_k$ and $\bigwedge_{k\geq 0}^{k<0} X_k$ is just 1 therefore X_0 is true and so $e_0(X)$ must be false and we can show this through a short derivation.

Algorithm 1 Fenwick tree [15] based algorithm to find Cov(0,J).

```
Require: $J < 2^n$
  function FENWICK-ASSIGNMENTS(int $J$, int $n$)
      int $\alpha := \{\}$, dash:= $\{\}$   /*$\alpha$, dash are each a set of integers*/
      int $indx \leftarrow J + 1$ /*assignments $\in [0, 2^n - 1]$ but the Fenwick tree handles $[1, 2^n]$*/
      while $indx > 0$ do
          parent $= indx - (indx \;\&\; - indx)$   /*& is the bit-wise AND operator*/
          $\alpha$.append(parent)
          dash.append($log(indx$ − parent))  /*records no. of variables to forget from $\alpha$*/
          $indx \leftarrow$ parent
      end while
  return $\alpha'$ = process($\alpha$, dash)
  /*"process" function does the following: for $i \in |\alpha|$, $\alpha'[i]$ is the partial assignment obtained
  from $\alpha[i]$ after discarding "dash[$i$]" number of variables from the right end of the fixed
  ordering of variables*/
  end function
```

Take p to be the maximum such that $\bigwedge_{k\geq 0}^{k<p} X_i$ is true, we can prove such a maximum exists by exhibiting a disjunction. Then for $0 < i < p$, $e_i(X) = 0$. $Y_k = 0$ for $k \leq i$ by definition of T and so $\neg e_i(Y)$ by definition of $e_i(Y)$. For $i = p$, $X_i = 0$ while $\bigwedge_{k\geq 0}^{k<i} X_i$ and $Y_i = 1$ while $\bigvee_{k\geq 0}^{k<i} \bar{Y}_i$ so both $e_i(X)$ and $e_i(Y)$ are true. $f_{p,p}(X) = f_{p,p}(Y) = 0$ because i is not strictly greater than itself. For $j > p$, we have to show that X_j and Y_j are equal. Recall that $Y_j = X_j \oplus \bigwedge_{k\geq 0}^{k<j} X_k$, but since X_p is false, $\bigwedge_{k\geq 0}^{k<j} X_k = 0$ and so $Y_j = X_j$. Hence $f_{p,j}(X) = f_{p,j}(Y)$.

For $i > p$, if $e_i(X)$ is true then $X_i \wedge \bigvee_{k\geq 0}^{k<i} \bar{X}_k$ must be true. $\bigvee_{k\geq 0}^{k<i} \bar{Y}_k$ must also be true because Y_0 is true. Since $X_i = Y_i$ then $Y_i \wedge \bigvee_{k\geq 0}^{k<i} \bar{Y}_k$ is also true and so $e_i(Y)$ can be proven that way. Since $X_j = Y_j$ for $j \geq i$ then by definition $f_{i,j}(X) = f_{i,j}(Y)$.

By definition, for $j > 0$, $f_{0,j}(Y) = Y_j$ and $f_{0,0}(Y) = 0 = Y_0$. ◀

▶ **Lemma 38.** *Assume $T(Y, X)$ (i.e. $Y = X + 1$) and Y_0 then there is some maximum $p : 0 < p \leq n$ such that $\bigwedge_{j\geq 0}^{j<p} Y_j$. Further the following properties are true and have short formal eFrege proofs.*

1. **a.** $e_i(X) \wedge \neg e_i(Y)$ *for* $0 \leq i < p$
   b. $f_{i,j}(X) \leftrightarrow Y_j$ *for* $0 \leq i < p \leq j < n$
   c. $f_{i,j}(X)$ *for* $0 \leq i < j < p \leq n$
   d. $\neg f_{i,p}(X)$ *for* $0 \leq i < p$
2. **a.** $\neg e_p(X) \wedge e_p(Y)$ *if* $p < n$
   b. $f_{p,j}(X) \leftrightarrow f_{p,j}(Y)$ *for* $p \leq j < n$.
   c. $\neg f_{p,p}(X) \wedge \neg f_{p,p}(Y)$ *if* $p < n$
3. **a.** $e_i(X) \leftrightarrow e_i(Y)$ *for* $p < i < n$
   b. $f_{i,j}(X) \leftrightarrow f_{i,j}(Y)$ *for* $p < i \leq j < n$

Proof. Because $Y_0 = 1$, then by definition of T we can prove $X_0 = 0$, hence $\bigvee_{k\geq 0}^{k<i} \bar{X}_k$ is always true for $i > 0$. This means that through the definition of T, $Y_i = X_i$ for $i > 0$. This will be important for many items when $i > 0$.

For $i = 0$ we get $e_0(X)$ and $\neg e_0(Y)$ through definition and use of $Y_0 \wedge \neg X_0$. And for $0 < i < p$, $X_i = 1$ and since $\bigvee_{k\geq 0}^{k<i} \bar{X}_k$ is true $e_i(X)$ is true. However $\bigwedge_{j\geq 0}^{j\leq i} Y_j$ is true so $e_i(Y)$ is false. Through $Y_i = X_i$ we also get that for $0 \leq i < j$, $f_{i,j}(X) = X_j = Y_j$. For $j < p$ we specifically get $Y_j = 1$ and for $j = p$ we get $Y_j = 0$. This completes all cases from 1.

In case 2, $e_p(X)$ is false because $X_p = Y_p = 0$ and $\bigwedge_{k\geq 0}^{k<p} X_k$ is false. Likewise, $e_p(X)$ is true because $Y_p = 0$ and $\bigwedge_{k\geq 0}^{k<p} Y_k$ is true. $f_{p,p}(X) = f_{p,p}(Y) = 0$ by definition and $f_{p,j}(X) = f_{p,j}(Y)$, for $j > p$ because then $X_j = Y_j$.

In case 3, again $f_{i,j}(X) = f_{i,j}(Y)$, for $j \geq i > p$ because $X_j = Y_j$. We can also use $X_j = Y_j$ to show $e_i(X) = e_i(Y)$ because $\bigvee_{k\geq 0}^{k<i} \bar{X}_k$ and $\bigvee_{k\geq 0}^{k<i} \bar{Y}_k$ are now both true.

All these cases can be formalised in eFrege proofs due to their simplicity for each choice of p. One final important step is to create and prove disjunction over all possible p. ◀

C Missing proofs from Section 5

▶ **Lemma 23.** CPOG *is closed under restrictions.*

Proof. For a given CNF ϕ, a CPOG proof consists of a POG G and a Resolution proof of $\phi \leftrightarrow G$. A POG is closed under conditioning as for any partial assignment α: replace the inputs labelled by x with $\alpha(x)$ for every x assigned by α and the resulting structure is still a POG G'. This is because, constants are allowed in POG and the $\wedge^p$-nodes will remain decomposable since we are only reducing variables. Also, the $\vee^p$-gates will remain deterministic because if A and B has disjoint models, so are $A|_\alpha$ and $B|_\alpha$. The Resolution proof witness is closed under restrictions. The CPOG proof of $\phi \leftrightarrow G$ uses Resolution proofs which are closed under restrictions, it implies $\phi|_\alpha \leftrightarrow G|_\alpha$. ◀

▶ **Corollary 24.** *There is a polynomial time method of extracting a cumulator circuit from a* CPOG *proof.*

Proof. With a CPOG proof, given an assignment α, in polynomial time we can calculate the Cov($\mathbf{0}$,α) via Fenwick's method (Lemma 10) and use closure under restrictions to find the values for the sum. By formalising these steps into a circuit as in Definition 13 we get the cumulator circuit. ◀

▶ **Lemma 26.** *Let α be an initial partial assignment defined up to x_i where $i > 0$. We can prove in the structure of the POG that $R(v, w_\alpha) = R(v, w_{\alpha_0}) = R(v, w_{\alpha_1})$ when x_{i-1} is not in the dependency set of v. This proof can be formalised in a short* eFrege *proof.*

Proof. In all cases, except at leaves, the dependency set is the union of the dependency sets of its children.

Boolean leaf: $R(1, w_\alpha) = \mathbf{1}$ and $R(0, w_\alpha) = \mathbf{0}$ independent of α. Therefore the Lemma statement is easily derived in this case.

Variable leaf: Let the variable leaf be x_j. $R(x_j, w)$ takes the value of $w(x_j)$. If $x_{i-1} \notin \mathcal{D}(x_j)$, then either $j \geq i$ or $j < i - 1$. This is formalised in a tautological disjunction in eFrege.

In either case we show equality is easily derived. If $j \geq i$ then $w_\alpha(x_j) = \mathbf{1}$ when $\alpha(j) = 1$ which also extends to $\alpha_0(j) = 1$ and $\alpha_1(j) = 1$ in which case $w_{\alpha_0}(x_j) = \mathbf{1}$ and $w_{\alpha_1}(x_j) = \mathbf{1}$. Similarly all $w_\alpha(x_j) = w_{\alpha_0}(x_j) = w_{\alpha_1}(x_j) = \mathbf{0}$ when $\alpha(j) = 0$. If $j < i - 1$ then $\alpha, \alpha_0, \alpha_1$ are all undefined on x_j, so the weights are all $\frac{1}{2}$.

Negation: Using the induction hypothesis on the child node c, $R(c, w_\alpha) = R(c, w_{\alpha_0}) = R(c, w_{\alpha_1})$ and so $R(\neg c, w_\alpha) = \mathbf{1} - R(c, w_\alpha) = \mathbf{1} - R(c, w_{\alpha_0}) = \mathbf{1} - R(c, w_{\alpha_1})$ therefore $R(\neg c, w_\alpha) = R(\neg c, w_{\alpha_0}) = R(\neg c, w_{\alpha_1})$.

Partition Conjunction: Let C be the set of child nodes for $\wedge^p$. Using the induction hypothesis $R(c,w_\alpha)=R(c,w_{\alpha_0})=R(c,w_{\alpha_1})$ for each child $c\in C$. We get the following equality for the products $\Pi_{c\in C}R(c,w_\alpha)=\Pi_{c\in C}R(c,w_{\alpha_0})=\Pi_{c\in C}R(c,w_{\alpha_1})$. Thus $R(\bigwedge^p_{c\in C},w_\alpha)=R(\bigwedge^p_{c\in C},w_{\alpha_0})=R(\bigwedge^p_{c\in C},w_{\alpha_1})$.

Partition Disjunction: Let the child nodes of $\vee^p$ be c,d. Using the induction hypothesis $R(c,w_\alpha)=R(c,w_{\alpha_0})=R(c,w_{\alpha_1})$ and $R(d,w_\alpha)=R(d,w_{\alpha_0})=R(d,w_{\alpha_1})$.
$R(c\vee^p d,w_\alpha)=R(c,w_\alpha)+R(d,w_\alpha)=R(c,w_{\alpha_0})+R(d,w_{\alpha_0})=R(c\vee^p d,w_{\alpha_0})$. Similarly, it can be derived that $R(c\vee^p d,w_\alpha)=R(c\vee^p d,w_{\alpha_1})$.

Each inductive step involves a bounded application of implications using the definitions hence we get short eFrege proofs. ◀

▶ **Lemma 27.** *Let α be an initial partial assignment defined up to x_i where $i>0$. We can prove in the structure of the POG that $R(v,w_\alpha)=\frac{\mathbf{1}}{\mathbf{2}}\cdot(R(v,w_{\alpha_0})+R(v,w_{\alpha_1}))$. Furthermore we can formalise this in short* eFrege *proofs.*

Proof. If $x_{i-1}\notin\mathcal{D}(v)$, this directly holds from Lemma 26, along with arithmetic properties i.e. $a=\frac{\mathbf{1}}{\mathbf{2}}\cdot(a+a)$. So here we only consider the case that $x_{i-1}\in\mathcal{D}(v)$.

Variable leaf: Let the variable leaf be x_j. $x_{i-1}\in\mathcal{D}(v)$ implies that $j=i-1$, then $R(v,w_\alpha)=\frac{\mathbf{1}}{\mathbf{2}}$. $R(v,w_{\alpha_0})=\mathbf{1}\rightarrow R(v,w_{\alpha_1})=\mathbf{0}$ and $R(v,w_{\alpha_0})=\mathbf{0}\rightarrow R(v,w_{\alpha_1})=1$, in both cases they sum to 1 which is the right identity when multiplied with $\frac{1}{2}$.

Negation: Using the induction hypothesis on the child node c, i.e. $R(c,w_\alpha)=\frac{1}{2}\cdot(R(c,w_{\alpha_0})+R(c,w_{\alpha_1}))$, it implies the following.

$$\begin{aligned}R(\neg c,w_\alpha)&=\mathbf{1}-R(c,w_\alpha)=\mathbf{1}-\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(R(c,w_{\alpha_0})+R(c,w_{\alpha_1}))\\&=\mathbf{1}-\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(\mathbf{1}-R(\neg c,w_{\alpha_0})+\mathbf{1}-R(\neg c,w_{\alpha_1}))\\&=\mathbf{1}-\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(\mathbf{2}-R(\neg c,w_{\alpha_0})-R(\neg c,w_{\alpha_1}))\\&=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(R(\neg c,w_{\alpha_0})+R(\neg c,w_{\alpha_1})).\end{aligned}$$

Partition Conjunction: Let C be the set of child nodes for $\wedge^p$. Observe that, $x_{i-1}\in\mathcal{D}(c^*)$ for exactly one child c^*. Along with multiplicative commutativity and associativity, we know the following from Proposition 21:

$R(\bigwedge\limits_{c\in C}^p c,w_\alpha)=R(c^*,w_\alpha)\cdot\prod_{c\in\{C\setminus c^*\}}R(c,w_\alpha)$.

Using the induction hypothesis on c^* we get

$$=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(R(c^*,w_{\alpha_0})+R(c^*,w_{\alpha_1}))\cdot\prod_{c\in\{C\setminus c^*\}}R(c,w_\alpha).$$

We can use left-distributivity to get

$$=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot R(c^*,w_{\alpha_0})\cdot\prod_{c\in\{C\setminus c^*\}}R(c,w_\alpha)+\tfrac{\mathbf{1}}{\mathbf{2}}\cdot R(c^*,w_{\alpha_1})\cdot\prod_{c\in\{C\setminus c^*\}}R(c,w_\alpha).$$

At this point we know that $x_{i-1}\notin\mathcal{D}(c\in\{C\setminus c^*\})$, using Lemma 26 we get

$$\begin{aligned}&=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot R(c^*,w_{\alpha_0})\cdot\prod_{c\in\{C\setminus c^*\}}R(c,w_{\alpha_0})+\tfrac{\mathbf{1}}{\mathbf{2}}\cdot R(c^*,w_{\alpha_1})\cdot\prod_{c\in\{C\setminus c^*\}}R(c,w_{\alpha_1})\\&=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot R(\bigwedge_{c\in C}^p c,w_{\alpha_0})+\tfrac{\mathbf{1}}{\mathbf{2}}\cdot R(\bigwedge_{c\in C}^p c,w_{\alpha_1})=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(R(\bigwedge_{c\in C}^p c,w_{\alpha_0})+R(\bigwedge_{c\in C}^p c,w_{\alpha_1})).\end{aligned}$$

Partition Disjunction: Let the child nodes of $\vee^p$ be c,d.

$R(c\vee^p d,w_\alpha)=R(c,w_\alpha)+R(d,w_\alpha)$

Using the induction hypothesis on c,d we get

$$=\tfrac{\mathbf{1}}{\mathbf{2}}(R(c,w_{\alpha_0})+R(c,w_{\alpha_1}))+\tfrac{\mathbf{1}}{\mathbf{2}}(R(d,w_{\alpha_0})+R(d,w_{\alpha_1}))$$

We can use additive commutativity and distributivity to get

$$\begin{aligned}&=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(R(c,w_{\alpha_0})+R(d,w_{\alpha_0})+R(c,w_{\alpha_1})+R(d,w_{\alpha_1}))\\&=\tfrac{\mathbf{1}}{\mathbf{2}}\cdot(R(c\vee^p d,w_{\alpha_0})+R(c\vee^p d,w_{\alpha_1}))\end{aligned}$$

Extended Frege can handle the bounded steps in each case of the inductive step. ◀

▶ **Lemma 28.** *For complete assignment α, we can prove using* eFrege *in the structure of the POG that $R(v, w_\alpha) = \mathbb{1}_v(\alpha)$.*

Proof. Again we show the base and inductive cases involve polynomially many basic steps.

Variable leaf: Let the variable leaf be x_i. $R(x_i, w_\alpha) = \mathbf{1}$ or $R(x_i, w_\alpha) = \mathbf{0}$ since $|\alpha| = |X|$ and the value is determined entirely by $\mathbb{1}_{x_i}(\alpha)$.

Negation: Suppose $R(c, w_\alpha) = \mathbf{1}$ then by the induction hypothesis α satisfies c, so α falsifies $\neg c$ and $R(\neg c, w_\alpha) = \mathbf{1} - R(c, w_\alpha) = \mathbf{0}$.
Similarly, it can be derived for $R(c, w_\alpha) = \mathbf{0}$ that $R(\neg c, w_\alpha) = \mathbf{1}$.

Partition Conjunction: Let C be the set of child nodes for $\wedge^p$. If there is some $c^* \in C$ such that α falsifies c^* then by the induction hypothesis $R(c^*, w_\alpha) = \mathbf{0}$. Then we can prove $\prod_{c\in C} R(c, w_\alpha) = \mathbf{0}$ which is the formula for $R(\bigwedge^p_{c\in C} c, w_\alpha)$. Also, α must falsify $\bigwedge^p_{c\in C} c$ as it falsifies c^*.
In the other case, if no $c \in C$ is falsified, α satisfies all of C (we can state and prove this formally as a disjunction). Here, $R(c, w_\alpha) = 1$ for all $c \in C$ and $\prod_{c\in C} R(c, w_\alpha) = \mathbf{1}$. Also, α must satisfy $\bigwedge^p_{c\in C} c$ as it satisfies all $c \in C$.

Partition Disjunction: Let the child nodes of $\vee^p$ be c, d. If α falsifies both c and d then by induction hypothesis, $R(c, w_\alpha) = R(d, w_\alpha) = \mathbf{0}$. By adding these we get $\mathbf{0} = R(c, w_\alpha) + R(d, w_\alpha) = R(c \vee^p d, w_\alpha)$. Also, α must falsify $c \vee^p d$ as it falsifies both c, d.
Suppose α satisfies c, we can prove that α falsifies d using the Resolution proof δ included in the CPOG proof (and this is simulated by eFrege). Hence, $R(c, w_\alpha) = \mathbf{1}$, $R(d, w_\alpha) = \mathbf{0}$. Then $R(c \vee^p d, w_\alpha) = \mathbf{1} + \mathbf{0} = \mathbf{1}$. This can be repeated for when α satisfies d by using left identity instead of right identity. ◀

Quantum Sabotage Complexity

Arjan Cornelissen
Simons Institute for the Theory of Computing, University of California, Berkeley, CA, USA
IRIF – CNRS, Paris, France

Nikhil S. Mande
University of Liverpool, UK

Subhasree Patro
Technische Universiteit Eindhoven, The Netherlands
Centrum Wiskunde en Informatica (QuSoft), Amsterdam, The Netherlands

Abstract

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the goal in the usual query model is to compute f on an unknown input $x \in \{0,1\}^n$ while minimizing the number of queries to x. One can also consider a "distinguishing" problem denoted by f_{sab}: given an input $x \in f^{-1}(0)$ and an input $y \in f^{-1}(1)$, either all differing bits are replaced by a $*$, or all differing bits are replaced by $\dagger$, and an algorithm's goal is to identify which of these is the case while minimizing the number of queries.

Ben-David and Kothari [ToC'18] introduced the notion of randomized sabotage complexity of a Boolean function to be the zero-error randomized query complexity of f_{sab}. A natural follow-up question is to understand the $\mathsf{Q}(f_{\mathsf{sab}})$, the quantum query complexity of f_{sab}. In this paper, we initiate a systematic study of this. The following are our main results for all Boolean functions $f : \{0,1\}^n \to \{0,1\}$.

- If we have additional query access to x and y, then $\mathsf{Q}(f_{\mathsf{sab}}) = O(\min\{\mathsf{Q}(f), \sqrt{n}\})$.
- If an algorithm is also required to output a differing index of a 0-input and a 1-input, then $\mathsf{Q}(f_{\mathsf{sab}}) = O(\min\left\{\mathsf{Q}(f)^{1.5}, \sqrt{n}\right\})$.
- $\mathsf{Q}(f_{\mathsf{sab}}) = \Omega(\sqrt{\mathsf{fbs}(f)})$, where $\mathsf{fbs}(f)$ denotes the fractional block sensitivity of f. By known results, along with the results in the previous bullets, this implies that $\mathsf{Q}(f_{\mathsf{sab}})$ is polynomially related to $\mathsf{Q}(f)$.
- The bound above is easily seen to be tight for standard functions such as And, Or, Majority and Parity. We show that when f is the Indexing function, $\mathsf{Q}(f_{\mathsf{sab}}) = \Theta(\mathsf{fbs}(f))$, ruling out the possibility that $\mathsf{Q}(f_{\mathsf{sab}}) = \Theta(\sqrt{\mathsf{fbs}(f)})$ for all f.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Quantum complexity theory

Keywords and phrases Sabotage complexity, quantum query complexity, Boolean functions, fractional block sensitivity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.19

Related Version *Full Version*: https://arxiv.org/abs/2408.12595

Funding *Subhasree Patro*: This work is co-funded by the European Union (ERC, ASC-Q, 101040624).

Acknowledgements We thank the anonymous reviewers for useful comments and pointers to related results that we had missed.

1 Introduction

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the goal in the standard query complexity model is to compute $f(x)$ on an unknown input $x \in \{0,1\}^n$ using as few queries to x as possible. One can also consider the following distinguishing problem: given $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$, output an index $i \in [n]$ such that $x_i \neq y_i$. This task can be formulated as follows: Consider an arbitrary $x \in f^{-1}(0)$, an arbitrary $y \in f^{-1}(1)$, and then either all indices where

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 19; pp. 19:1–19:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

x and y differ are replaced by the symbol $*$, or all such indices are replaced by the symbol $\dagger$. The goal of an algorithm is to identify which of these is the case, with query access to this "sabotaged" input. A formal description of this task is given below.

Let $f : D \to \{0,1\}$ with $D \subseteq \{0,1\}^n$ be a (partial) Boolean function. Let $D_0 = f^{-1}(0)$ and $D_1 = f^{-1}(1)$. For any pair $(x, y) \in D_0 \times D_1$, define $[x, y, *] \in \{0, 1, *\}^n$ to be

$$[x,y,*]_i = \begin{cases} x_i, & \text{if } x_i = y_i, \\ *, & \text{otherwise.} \end{cases}$$

Similarly, for any pair $(x, y) \in D_0 \times D_1$, define $[x, y, \dagger] \in \{0, 1, \dagger\}^n$ to be

$$[x,y,\dagger]_i = \begin{cases} x_i, & \text{if } x_i = y_i, \\ \dagger, & \text{otherwise.} \end{cases}$$

Let $S_* = \{[x, y, *] \mid (x, y) \in D_0 \times D_1\}$ and $S_\dagger = \{[x, y, \dagger] \mid (x, y) \in D_0 \times D_1\}$. That is, S_* is the set of $*$-sabotaged inputs for f and $S_\dagger$ is the set of $\dagger$-sabotaged inputs for f. Finally, let $f_{\mathsf{sab}} : S_* \cup S_\dagger \to \{0, 1\}$ be the function defined by

$$f_{\mathsf{sab}}(z) = \begin{cases} 0, & z \in S_*, \\ 1, & z \in S_\dagger. \end{cases}$$

That is, f_{sab} takes as input a sabotaged input to f and identifies if the input is $*$-sabotaged or if it is $\dagger$-sabotaged.

Ben-David and Kothari [10] introduced the notion of *randomized sabotage complexity* of a Boolean function f, defined to be $\mathsf{RS}(f) := \mathsf{R}_0(f_{\mathsf{sab}})$, where $\mathsf{R}_0(\cdot)$ denotes randomized zero-error query complexity. It is not hard to see that $\mathsf{RS}(f) = O(\mathsf{R}(f))$ (where $\mathsf{R}(\cdot)$ denotes randomized bounded-error query complexity); this is because a randomized algorithm that succeeds with high probability on both a 0-input x and a 1-input y must, with high probability, query an index where x and y differ. Ben-David and Kothari also showed that randomized sabotage complexity admits nice composition properties. It is still open whether $\mathsf{RS}(f) = \Theta(\mathsf{R}(f))$ for all total Boolean functions f. If true, this would imply that randomized query complexity admits a perfect composition theorem, a goal towards which a lot of research has been done [5, 16, 14, 8, 6, 9, 13, 24]. This motivates the study of randomized sabotage complexity.

In the same paper, they mentioned that one could define $\mathsf{QS}(f) := \mathsf{Q}(f_{\mathsf{sab}})$ (here, $\mathsf{Q}(\cdot)$ denotes bounded-error quantum query complexity), but they were unable to show that it lower bounds $\mathsf{Q}(f)$. In a subsequent work [11], they defined a quantum analog, denoted $\mathsf{QD}(f)$, and called it *quantum distinguishing complexity*. $\mathsf{QD}(f)$ is the minimum number of queries to the input $x \in D$ to produce an output state such that the output states corresponding to 0-inputs and 1-inputs are far from each other. Analogous to their earlier result, they were also able to show that $\mathsf{QD}(f) = O(\mathsf{Q}(f))$ for all total f. Additionally, using $\mathsf{QD}(f)$ as an intermediate measure, they were able to show a (then) state-of-the-art 5th-power relationship between zero-error quantum query complexity and bounded-error quantum query complexity: $\mathsf{Q}_0(f) = \widetilde{O}(\mathsf{Q}(f)^5)$ for all total f. We note here that a 4th-power relationship was subsequently shown between $\mathsf{D}(f)$ and $\mathsf{Q}(f)$ [2], also implying $\mathsf{Q}_0(f) = O(\mathsf{Q}(f)^4)$ for all total f. The proof of this relied on Huang's celebrated sensitivity theorem [19].

Our results

In this paper, we initiate a systematic study of the natural quantum analog of randomized sabotage complexity alluded to in the previous paragraph, which we call *quantum sabotage complexity*, denoted by $\mathsf{QS}(f) := \mathsf{Q}(f_{\mathsf{sab}})$. Slightly more formally, we consider the following variants:

- We consider two input models. In the weak input model, the oracle simply has query access to an input in $z \in S_* \cup S_\dagger$. In the strong input model, the oracle additionally has access to the original inputs $x \in D_0$ and $y \in D_1$ that yield the corresponding input in $S_* \cup S_\dagger$. The model under consideration will be clear by adding either weak or str as a subscript to QS.
- We also consider two different output models: one where an algorithm is only required to output whether the input was in S_* or in $S_\dagger$, and a stronger version where an algorithm is required to output an index $i \in [n]$ with $z_i \in \{*, \dagger\}$. The model under consideration will be clear by adding either no superscript or ind as a superscript to QS.

As an example, $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{weak}}(f)$ denotes the quantum sabotage complexity of f under the weak input model, and in the output model where an algorithm needs to output a differing index.

One can also consider these nuances in defining the input and output models in the randomized setting. However, one can easily show that they are all equivalent for randomized algorithms. We refer the reader to Section 3 for a proof of this, and for a formal description of these models.

An immediate upper bound on $\mathsf{QS}(f)$, for any (partial) Boolean function f, follows directly from Grover's search algorithm [17]. Indeed, for any sabotaged input, we know that at least one of the input symbols must be either a $*$ or $\dagger$. Thus, we can simply use the unstructured search algorithm by Grover to find (the position of) such an element in $O(\sqrt{n})$ queries. This immediately tells us that for Boolean functions where $\mathsf{Q}(f) = \omega(\sqrt{n})$, $\mathsf{QS}(f)$ is significantly smaller than $\mathsf{Q}(f)$.

As mentioned earlier, Ben-David and Kothari left open the question of whether $\mathsf{QS}(f) = O(\mathsf{Q}(f))$, the quantum analog of randomized sabotage complexity being at most randomized query complexity. We first observe that in the strong input model, this holds true.

▶ **Lemma 1.** *Let n be a positive integer, let $D \subseteq \{0,1\}^n$. Let $f : D \to \{0,1\}$ be a (partial) Boolean function. Then, $\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{Q}(f))$.*

The proof idea is simple: consider an input $z \in S^* \cup S^\dagger$ obtained by sabotaging $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$. Run a quantum query algorithm for f on input z, such that:

- Whenever a bit in $\{0,1\}$ is encountered, the algorithm proceeds as normal.
- Whenever a $*$ is encountered, query the corresponding bit in x.
- Whenever a $\dagger$ is encountered, query the corresponding bit in y.

The correctness follows from the following observation: if $z \in S^*$, then the run of the algorithm is exactly that of the original algorithm on x, and if $z \in S^\dagger$, then the run is the same of the original algorithm on y.

This procedure does not return a $*/\dagger$-index, and it is natural to ask if $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(f) = O(\mathsf{Q}(f))$ as well. While we are unable to show this, we make progress towards this by showing the following, which is our first main result.

▶ **Theorem 2.** *Let n be a positive integer, let $D \subseteq \{0,1\}^n$, and let $f : D \to \{0,1\}$ be a (partial) Boolean function. Then, $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(f) = O(\mathsf{Q}(f)^{1.5})$.*

Our result is actually slightly stronger than this; our proof shows that $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(f) = O(\mathsf{QD}(f)^{1.5})$. This implies Theorem 2 using the observation of Ben-David and Kothari that $\mathsf{QD}(f) = O(\mathsf{Q}(f))$ for all (partial) f.

In order to show Theorem 2, we take inspiration from the observation that randomized sabotage complexity is at most randomized query complexity. This is true because with high probability, a randomized query algorithm must spot a differing bit between any pair of inputs with different output values. However, this argument cannot immediately be ported to the quantum setting because quantum algorithms can make queries in superposition. We are able to do this, though, by stopping a quantum query algorithm for f at a random time and measuring the index register. We note here that it is important that we have oracle access not only to a sabotaged input $z \in S_* \cup S_\dagger$, but also the inputs $(x, y) \in D_0 \times D_1$ that yielded the underlying sabotaged input. Using arguments reminiscent of the arguments in the hybrid method [12], we are able to show that the success probability of this is only $1/\mathsf{Q}(f)$. Applying amplitude amplification to this process leads to an overhead of $\sqrt{\mathsf{Q}(f)}$, and yields Theorem 2.

We remark here that this proof idea is reminiscent of the proof of [11, Theorem 1]. However, there are some technical subtleties. At a high level, the main subtleties are the following: in the strong oracle model that we consider, it is easy to check whether or not a terminated run of a Q algorithm actually gives us a $*/\dagger$ index. This is not the case in [11, Proof of Theorem 1]. This allows us to save upon a quadratic factor because we can do amplitude amplification.

Our proof approach modifies a core observation by Ben-David and Kothari [11, Lemma 12]. In their setting, they consider a T-query algorithm $\mathcal{A}$ that computes a function f with high probability. Take two inputs x and y such that $f(x) \neq f(y)$, and let $B \subseteq [n]$ be the indices on which x and y differ. Suppose we run this algorithm on x, interrupt it right before the tth query, and then measure the query register. The probability that we measure an index $i \in B$, we denote by p_t. Then, [11, Lemma 12] shows that

$$\sum_{t=1}^{T} p_t = \Omega\left(\frac{1}{T}\right).$$

We show that by adding the probabilities that come from running the algorithm at input y, we can replace the lower bound of $\Omega(1/T)$ by a much stronger lower bound of $\Omega(1)$. We state this observation more formally in the following lemma.

▶ **Lemma 3.** *Let $x \in \{0,1\}^n$ be an input and let $B \subseteq [n]$. Let $\mathcal{A}$ be a T-query quantum algorithm that accepts x and rejects x_B with high probability, or more generally produces output states that are a constant distance apart in trace distance for x and x_B. Let p_t, resp. p_t^B, be the probability that, when $\mathcal{A}$ is run on x, resp. x_B, up until, but not including, the t-th query and then measured, it is found to be querying a position $i \in B$. Then,*

$$\sum_{t=1}^{T} (p_t + p_t^B) = \Omega(1).$$

▶ Remark 4. We remark here that a stronger statement than that in Lemma 3 has already been shown in [21, Lemma 3.1]. We thank an anonymous reviewer for pointing this out to us.

We find this lemma independently interesting and are confident that it will find use in future research, given the use of such statements in showing quantum lower bounds via the adversary method, for example (see [26, Chapters 11-12] and the references therein). Note that this lemma is very amenable to our "strong" sabotage complexity setup: imagine running an algorithm simultaneously on inputs x and x_B that have different function values. Lemma 3 says that on stopping at a random time in the algorithm, and choosing one of x and x_B at random, the probability of seeing an index in B (i.e., a $*$-index or a $\dagger$-index) is a constant. Indeed, this lemma is a natural quantum generalization of the phenomenon that occurs in the randomized setting: a randomized algorithm that distinguishes x and x_B must read an index in B with constant probability (on input either x or x_B).

We now discuss our remaining results. Given Theorem 2, it is natural to ask if $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(\cdot)$ is polynomially related to $\mathsf{Q}(\cdot)$. Using the positive-weighted adversary lower bound for quantum query complexity [4], we are able to show that $\mathsf{QS}_{\mathsf{str}}(f) = \Omega(\sqrt{\mathsf{fbs}(f)})$ for all total f (and hence the same lower bound holds for $\mathsf{QS}_{\mathsf{weak}}(f)$, and $\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f)$, and $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f)$ as well).

▶ **Theorem 5.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Then $\mathsf{QS}_{\mathsf{str}}(f) = \Omega(\sqrt{\mathsf{fbs}(f)})$.*

Fractional block sensitivity is further lower bounded by block sensitivity, which is known to be polynomially related to $\mathsf{R}(\cdot)$ and $\mathsf{Q}(\cdot)$ [23, 7]. Using the best-known relationship of $\mathsf{Q}(f) = \widetilde{O}(\mathsf{bs}^3(f))$ [1], Theorem 2 and Theorem 5 implies the following polynomial relationship between $\mathsf{QS}_{\mathsf{str}}(f)$, $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f)$ and $\mathsf{Q}(f)$ for all total Boolean functions f.

$$\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{Q}(f)), \qquad \mathsf{Q}(f) = \widetilde{O}(\mathsf{QS}_{\mathsf{str}}(f)^6). \tag{1}$$

$$\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) = O(\mathsf{Q}(f)^{1.5}), \qquad \mathsf{Q}(f) = \widetilde{O}(\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f)^6). \tag{2}$$

In the weakest input model, we have $\mathsf{QS}_{\mathsf{weak}}(f) = O(\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f)) = O(\mathsf{R}(f_{\mathsf{sab}})) = O(\mathsf{R}(f)) = O(\mathsf{Q}(f)^4)$ (where the last inequality follows from [2]). Thus, in the weakest input model for sabotage complexity, Theorem 5 implies the following polynomial relationship with $\mathsf{Q}(f)$:

$$\mathsf{QS}_{\mathsf{weak}}(f) = O(\mathsf{Q}(f)^4), \qquad \mathsf{Q}(f) = \widetilde{O}(\mathsf{QS}_{\mathsf{weak}}(f)^6). \tag{3}$$

$$\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f) = O(\mathsf{Q}(f)^4), \qquad \mathsf{Q}(f) = \widetilde{O}(\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f)^6). \tag{4}$$

It would be interesting to find the correct polynomial relationships between all of these measures. In particular, we suspect that $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) = O(\mathsf{Q}(f))$ for all total f, but we are unable to show this.

As mentioned in the discussion before Theorem 2, it is easy to show that $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) = O(\sqrt{n})$ for all $f : \{0,1\}^n \to \{0,1\}$. Thus, the lower bound in terms of block sensitivity given by Theorem 5 is actually tight for standard functions like And, Or, Parity and Majority. Given this, it is natural to ask if $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) = \Theta(\sqrt{\mathsf{fbs}(f)})$ for all total Boolean f. We show that this is false, by showing that for the Indexing function $\mathsf{IND}_n : \{0,1\}^{n+2^n} \to \{0,1\}$ defined by $\mathsf{IND}_n(x,y) = y_{\mathsf{bin}(x)}$ (where $\mathsf{bin}(x)$ denotes the integer in $[2^n]$ with the binary representation x), $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(\mathsf{IND}_n) = \Theta(\mathsf{fbs}(\mathsf{IND}_n)) = \Theta(n)$.

▶ **Theorem 6.** *Let n be a positive integer. Then, $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(\mathsf{IND}_n) = \Theta(n)$.*

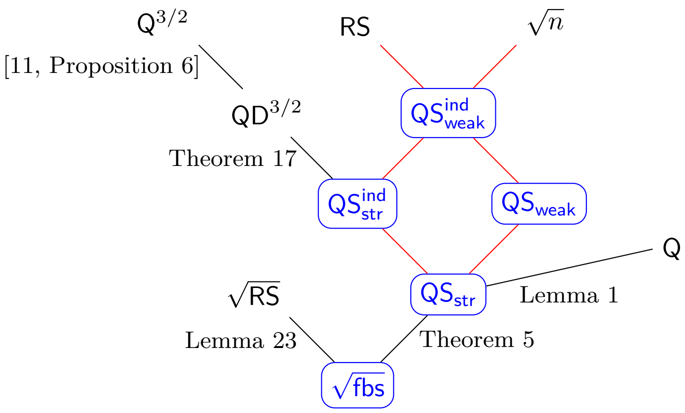
In order to show this, we use a variation of Ambainis' basic adversary method [3, Theorem 5.1], also presented in the same paper [3, Theorem 6.1] (see Lemma 19).

Finally, we summarize the relations we proved in Figure 1.

2 Preliminaries

All logarithms in this paper are taken base 2 unless mentioned otherwise. For a positive integer n, we use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and use $[n]_0$ to denote $\{0, 1, \ldots, n-1\}$. Let $v \in \mathbb{C}^d$, the $\|v\|_2 = \sqrt{\sum_{i=1}^{d} |v_i|^2}$. Let A, B be square matrices in $\mathbb{C}^{d\times d}$. We use $A^\dagger$ to denote the conjugate transpose of matrix A. The operator norm of a matrix A, denoted by $\|A\|$, is the largest singular value of A, i.e., $\|A\| = \max_{v:\|v\|_2=1} \|Av\|_2$. The trace distance between two matrices A, B, denoted by $\|A-B\|_{\mathsf{tr}} = \frac{1}{2}\|A-B\|_1$ where $\|A\|_1 = \mathrm{Tr}(\sqrt{A^\dagger A})$. For two $d \times d$ matrices A, B, $A \circ B$ denotes the Hadamard, or entry-wise, product of A and B.

We refer the reader to [26, Chapter 1] for the relevant basics of quantum computing.



$\mathsf{Q}^{3/2}$
RS
$\sqrt{n}$
[11, Proposition 6]
$\mathsf{QD}^{3/2}$
$\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}$
Theorem 17
$\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}$
$\mathsf{QS}_{\mathsf{weak}}$
Q
$\sqrt{\mathsf{RS}}$
$\mathsf{QS}_{\mathsf{str}}$
Lemma 1
Lemma 23
Theorem 5
$\sqrt{\mathsf{fbs}}$

Figure 1 Overview of the relations proved in this work. If nodes A and B are connected, and A is below B, then $A = O(B)$. All the red edges are reasonably straightforward inclusions, and they are proved in Proposition 16.

Let $D \subseteq \{0,1\}^n$, let R be a finite set, and let $f \subseteq D \times R$ be a relation. A quantum query algorithm $\mathcal{A}$ for f begins in a fixed initial state $|\psi_0\rangle$ in a finite-dimensional Hilbert space, applies a sequence of unitaries $U_0, O_x, U_1, O_x, \ldots, U_T$, and performs a measurement. Here, the initial state $|\psi_0\rangle$ and the unitaries $U_0, U_1, \ldots, U_T$ are independent of the input. The unitary O_x represents the "query" operation, and does the following for each basis state: it maps $|i\rangle\,|b\rangle$ to $|i\rangle\,|b + x_i \mod 2\rangle$ for all $i \in [n]$. The algorithm then performs a measurement and outputs the observed value. We say that $\mathcal{A}$ is a bounded-error quantum query algorithm computing f if for all $x \in D$ the probability of outputting r such that $(x, r) \in f$ is at least $2/3$. The (bounded-error) *quantum query complexity of* f, denoted by $\mathsf{Q}(f)$, is the least number of queries required for a quantum query algorithm to compute f with error at most $1/3$. We use $\mathsf{R}(f)$ to denote the *randomized query complexity* of f, which is the worst-case cost (number of queries) of the best randomized algorithm that computes f to error at most $1/3$ on all inputs.

We recall some known complexity measures.

▶ **Definition 7** (Block sensitivity). *Let n be a positive integer and $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For any $x \in \{0,1\}^n$, a block $B \subseteq [n]$ is said to be sensitive on an input $x \in \{0,1\}^n$ if $f(x) \neq f(x \oplus B)$, where $x \oplus B$ (or x_B) denotes the string obtained by taking x and flipping all bits in B. The* block sensitivity of f on x, *denoted* $\mathsf{bs}(f, x)$, *is the maximum number of pairwise disjoint blocks that are sensitive on x. The* block sensitivity of f, *denoted by* $\mathsf{bs}(f)$, *is* $\max_{x \in \{0,1\}^n} \mathsf{bs}(f, x)$.

The *fractional block sensitivity* of f on x, denoted $\mathsf{fbs}(f, x)$, is the optimum value of the linear program below. We refer the reader to [15, 20] for a formal treatment of fractional block sensitivity and related measures, and we simply state its definition here.

▶ **Definition 8** (Fractional block sensitivity). *Let n be a positive integer and $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Let $x \in \{0,1\}^n$ and let $Y_x = \{z \in \{0,1\}^n : f(x) \neq f(z)\}$. The fractional block sensitivity of f on x, denoted by* $\mathsf{fbs}(f, x)$, *is the optimal value of the following optimization program.*

$$\begin{aligned} \max \quad & \sum_{y \in Y_x} w_y, & \\ \text{subject to} \quad & \sum_{\substack{y \in Y_x \\ x_j \neq y_j}} w_y \leq 1, & \forall j \in [n], \\ & w_y \geq 0, & \forall y \in Y_x. \end{aligned}$$

The fractional block sensitivity of f, denoted by $\mathsf{fbs}(f)$*, is* $\max_{x \in \{0,1\}^n} \mathsf{fbs}(f,x)$.[1]

We also state the non-negative weight adversary bound. It appears in many forms in the literature. We refer to the form mentioned in [18] which is equivalent to the following definition.

▶ **Definition 9** (Non-negative weight adversary bound). *Let n be a positive integer, Σ and Π finite sets, $D \subseteq (\Sigma)^n$, and $f : D \to \Pi$. The adversary bound is the following optimization program.*

$$\begin{aligned} \max \quad & \|\Gamma\| & \\ s.t. \quad & \|\Gamma \circ \Delta_j\| \leq 1, & \forall j \in [n], \\ & \Gamma[x,y] = 0, & \text{if } f(x) = f(y). \end{aligned}$$

Here, the optimization is over all symmetric, entry-wise non-negative adversary matrices $\Gamma \in \mathbb{R}^{D \times D}$. The matrix $\Delta_j \in \{0,1\}^{D \times D}$ has entries $\Delta_j[x,y] = 1$ if and only if $x_j \neq y_j$. The optimal value of this optimization program is denoted by $\mathsf{ADV}^+(f)$*.*

Sometimes, the optimal value of the non-negative weight adversary bound is also written as $\mathsf{ADV}(f)$. However, one can also consider the general adversary bound, in which the entries of the matrix Γ are not constrained to be non-negative. To clearly differentiate between the optimal values of these optimization programs, we distinguish between them by explicitly writing $\mathsf{ADV}^+(f)$ and $\mathsf{ADV}^\pm(f)$.

▶ **Definition 10** (Quantum distinguishing complexity). *Let n be a positive integer. The quantum distinguishing complexity of a (partial) Boolean function $f : D \to \{0,1\}$ (where $D \subseteq \{0,1\}^n$), denoted by* $\mathsf{QD}(f)$*, is the smallest integer k such that there exists a k-query algorithm that on input $x \in D$ outputs a quantum state ρ_x such that*

$$\forall x, y \in D, \quad \|\rho_x - \rho_y\|_{\mathsf{tr}} \geq 1/6,$$

whenever $f(x) \neq f(y)$.

The non-negative weight adversary bound is known to be a lower bound to the quantum distinguishing complexity, which in turn is a lower bound on the quantum query complexity, by [11, Proposition 6]. We state it below.

▶ **Theorem 11** ([11]). *Let n be a positive integer, $D \subseteq \{0,1\}^n$, and $f : D \to \{0,1\}$. Then,*

$$\mathsf{ADV}^+(f) = O(\mathsf{QD}(f)) = O(\mathsf{Q}(f))).$$

The relation $\mathsf{ADV}^+(f) = O(\mathsf{Q}(f))$ holds in the non-Boolean case as well, which follows directly from the definition and known results about the non-negative weight adversary bound, as can be found in [22], for instance.

[1] The block sensitivity of f on x is captured by the integral version of this linear program, where the variable $w_y \in \{0,1\}$ enforces that the blocks must be disjoint.

3 Sabotage complexity

In this section we first define sabotage variants of a Boolean function f that are convenient to work with. Specifically, these variants are useful because they enable us to work with the usual quantum query complexity model in the quantum setting, allowing us to use known results in this setting. After this, we analyze some basic properties of quantum sabotage complexities.

3.1 Formal setup of sabotage complexity

We start by formally defining the sabotage function of f.

▶ **Definition 12** (Sabotage functions and relations). *Let n be a positive integer, $D \subseteq \{0,1\}^n$. Let $f : D \to \{0,1\}$ be a (partial) Boolean function. For any input pair $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$, we define $[x,y,*], [x,y,\dagger] \in \{0,1,*,\dagger\}^n$ by*

$$[x,y,*]_j = \begin{cases} x_j, & \text{if } x_j = y_j, \\ *, & \text{otherwise,} \end{cases} \qquad \text{and} \qquad [x,y,\dagger]_j = \begin{cases} x_j, & \text{if } x_j = y_j, \\ \dagger, & \text{otherwise.} \end{cases}$$

We let $S_* = \{[x,y,*] : x \in f^{-1}(0), y \in f^{-1}(1)\}$, $S_\dagger = \{[x,y,\dagger] : x \in f^{-1}(0), y \in f^{-1}(1)\}$, and we let $D_{\mathsf{sab}} = S_* \cup S_\dagger \subseteq \{0,1,*,\dagger\}^n$.

- *The* sabotage function of f *is defined as $f_{\mathsf{sab}} : D_{\mathsf{sab}} \to \{0,1\}$, where $f_{\mathsf{sab}}(z) = 1$ iff $z \in S_\dagger$.*
- *The* sabotage relation of f *is defined as $f_{\mathsf{sab}}^{\mathsf{ind}} \subseteq D_{\mathsf{sab}} \times [n]$, where $(z,j) \in f_{\mathsf{sab}}^{\mathsf{ind}}$ iff $z_j \in \{*,\dagger\}$.*

For every $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$, we let $(x,y,*)$ denote $((x_j,y_j,z_j))_{j=1}^n$, where $z = [x,y,*]$. Similarly, for every $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$, we let $(x,y,\dagger)$ denote $((x_j,y_j,z_j))_{j=1}^n$, where $z = [x,y,\dagger]$. For $b \in \{*,\dagger\}$, let $S_b^{\mathsf{str}} = \{(x,y,b) : x \in f^{-1}(0), y \in f^{-1}(1)\}$, and $D_{\mathsf{sab}}^{\mathsf{str}} = S_*^{\mathsf{str}} \cup S_\dagger^{\mathsf{str}}$.

- *We define the* strong sabotage function of f *as $f_{\mathsf{sab}}^{\mathsf{str}} : D_{\mathsf{sab}}^{\mathsf{str}} \to \{0,1\}$, where $f_{\mathsf{sab}}^{\mathsf{str}}(w) = 1$ iff $w \in S_\dagger^{\mathsf{str}}$.*
- *We define the* strong sabotage relation of f *as $f_{\mathsf{sab}}^{\mathsf{str,ind}} \subseteq D_{\mathsf{sab}}^{\mathsf{str}} \times [n]$, where $(w,j) \in f_{\mathsf{sab}}^{\mathsf{str,ind}}$ iff $z_j \in \{*,\dagger\}$ where $w = ((x_j,y_j,z_j))_{j=1}^n$.*

If we want to compute f_{sab}, we need to consider how we are given access to the input of f_{sab}. To that end, we consider two input models, the weak and the strong input model. Both can be viewed as having regular query access to the inputs of the function f_{sab} and $f_{\mathsf{sab}}^{\mathsf{str}}$, respectively.

▶ **Definition 13** (Weak sabotage input model). *Let n be a positive integer, $D \subseteq \{0,1\}^n$, and $f : D \to \{0,1\}$ be a (partial) Boolean function. Let D_{sab} be as in Definition 12. In the* weak sabotage input model, *on some input $z \in D_{\mathsf{sab}}$, we are given access to an oracle O_z^{weak} that when queried the jth position returns z_j. In the quantum setting, this means that the oracle performs the mapping*

$$O_z^{\mathsf{weak}} : |j\rangle\,|b\rangle \mapsto |j\rangle\,|(b + z_j) \mod 4\rangle\,, \quad \forall b \in [4]_0, \forall j \in [n],$$

where $*$ is identified with 2 and $\dagger$ is identified with 3.

We also consider a stronger model.

▶ **Definition 14** (Strong sabotage input model). *Let n be a positive integer, $D \subseteq \{0,1\}^n$, and $f : D \to \{0,1\}$ be a (partial) Boolean function. Let $D_{\mathsf{sab}}^{\mathsf{str}}$ be as in Definition 12. In the* strong sabotage input model*, on some input $w = ((x_j, y_j, z_j))_{j=1}^n$, we are given access to an oracle O_w^{str} that when queried the jth position returns the tuple (x_j, y_j, z_j). In the quantum setting, this means that the oracle performs the mapping*

$$O_w^{\mathsf{str}} : |j\rangle |b_x\rangle |b_y\rangle |b_z\rangle \mapsto |j\rangle |b_x \oplus x_j\rangle |b_y \oplus y_j\rangle |(b_z + z_j) \mod 4\rangle ,$$

for all $b_x, b_y \in \{0,1\}$, $b_z \in [4]_0$ and for all $j \in [n]$. As in the previous definition, $*$ is identified with 2 and $\dagger$ is identified with 3.

Note that in the stronger model, we are implicitly also given the information which of the two inputs x and y are the 0- and 1-inputs of f. Indeed, we assume that the bits queried in the first entry of the tuple, always correspond to the 0-input that defined the sabotaged input z. We remark here that we can always remove this assumption if we allow for additive overhead of $O(\mathsf{Q}(f))$, after all we can always run a quantum algorithm that computes f on the first bits from all our queried tuple, to compute $f(x)$, and thus finding out whether it was a 0- or 1-input to begin with.

Having defined the sabotage functions/relations and the input models we now define four different notions of quantum sabotage complexity of f.

▶ **Definition 15** (Sabotage complexity). *Let n be a positive integer, $D \subseteq \{0,1\}^n$, and let $f : D \to \{0,1\}$ be a (partial) Boolean function. Define,*

$$\mathsf{QS}_{\mathsf{weak}}(f) \coloneqq \mathsf{Q}(f_{\mathsf{sab}}), \quad \mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f) \coloneqq \mathsf{Q}(f_{\mathsf{sab}}^{\mathsf{ind}}), \quad \mathsf{QS}_{\mathsf{str}}(f) \coloneqq \mathsf{Q}(f_{\mathsf{sab}}^{\mathsf{str}}), \quad \mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) \coloneqq \mathsf{Q}(f_{\mathsf{sab}}^{\mathsf{str,ind}}).$$

Analogously, define

$$\mathsf{RS}_{\mathsf{weak}}(f) \coloneqq \mathsf{R}(f_{\mathsf{sab}}), \quad \mathsf{RS}_{\mathsf{weak}}^{\mathsf{ind}}(f) \coloneqq \mathsf{R}(f_{\mathsf{sab}}^{\mathsf{ind}}), \quad \mathsf{RS}_{\mathsf{str}}(f) \coloneqq \mathsf{R}(f_{\mathsf{sab}}^{\mathsf{str}}), \quad \mathsf{RS}_{\mathsf{str}}^{\mathsf{ind}}(f) \coloneqq \mathsf{R}(f_{\mathsf{sab}}^{\mathsf{str,ind}}).$$

3.2 Quantum sabotage complexity

In Appendix A we show that the randomized sabotage complexity of a function is (asymptotically) the same in all of the four models we consider. In the quantum case, we have not been able to prove such equivalences. However, we can still prove some bounds between them. We refer the reader to Figure 1 for a pictorial representation of all relationships.

▶ **Proposition 16.** *Let n be a positive integer, $D \subseteq \{0,1\}^n$, and let $f : D \to \{0,1\}$ be a (partial) Boolean function. Then,*

$$\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f)) = O(\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f)) = O(\min\{\mathsf{RS}(f)), \sqrt{n}\}),$$

and

$$\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{QS}_{\mathsf{weak}}(f)) = O(\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f)).$$

Proof. Just as in the randomized case, $\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{QS}_{\mathsf{weak}}(f))$ and $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) = O(\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}})(f)$, because the input model is stronger, i.e., we can simulate a query to the weak oracle with $O(1)$ queries to the strong oracle. Furthermore, $\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f))$ and $\mathsf{QS}_{\mathsf{weak}}(f) = O(\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f))$, because once we have found a $j \in [n]$ where $x_j \neq y_j$, we can query that index with one more query to find figure out whether we have a $*$-input or a $\dagger$-input.

In the strong input model, we can always simply run Grover's algorithm to find a position $j \in [n]$ where $z_j \in \{*, \dagger\}$. This takes $O(\sqrt{n})$ queries. Thus, it remains to show that $\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f) = O(\mathsf{RS}(f))$. We showed in the previous proposition that $\mathsf{RS}(f)$ is the same up to constants to $\mathsf{RS}_{\mathsf{weak}}^{\mathsf{ind}}(f)$, and since the quantum computational model is only stronger than the randomized one, we find $\mathsf{QS}_{\mathsf{weak}}^{\mathsf{ind}}(f) = O(\mathsf{RS}_{\mathsf{weak}}^{\mathsf{ind}}(f)) = O(\mathsf{RS}(f))$. ◀

4 Upper bounds on QS

In this section, we prove upper bounds on the complexity measures introduced in Section 3. We start by showing that the quantum sabotage complexity in the strong model is upper bounded by the regular bounded-error quantum query complexity. Thereby, we prove that $\mathsf{QS}_{\mathsf{str}}(f)$ has the property that was sought for in [10], i.e., in this model computing f_{sab} indeed costs at most as many queries as computing f itself.

▶ **Lemma 1.** *Let n be a positive integer, let $D \subseteq \{0,1\}^n$. Let $f : D \to \{0,1\}$ be a (partial) Boolean function. Then, $\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{Q}(f))$.*

Proof. Let $\mathcal{A}$ be a bounded-error quantum query algorithm that computes f. We construct a quantum query algorithm $\mathcal{B}$ that computes f_{sab} in the strong input model.

Recall that in the strong input model (as in Definition 14), our input is viewed as $w = ((x_j, y_j, z_j))_{j=1}^n$ where $f(x) = 0, f(y) = 1$ and z is the sabotaged input constructed from x and y. A query on the jth position to the oracle O_w^{str} returns a tuple (x_j, y_j, z_j). Now, we define $\mathcal{B}$ to be the same algorithm as $\mathcal{A}$, but whenever $\mathcal{A}$ makes a query, it performs the following operation instead:

1. Query O_w^{str}, denote the outcome by (x_j, y_j, z_j).
2. If $z_j \in \{0,1\}$, return z_j.
3. If $z_j = *$, return x_j.
4. If $z_j = \dagger$, return y_j.

Note that this operation can indeed be implemented quantumly making 2 queries to O_w^{str}. The initial query performs the instructions described above, and the second query uncomputes the values from the tuple we don't need for the rest of the computation. Note here that $(O_w^{\mathsf{str}})^4 = I$, and thus $(O_w^{\mathsf{str}})^3 = (O_w^{\mathsf{str}})^{-1}$, which is what we need to implement for our uncompute operations.

We observe that the above operation always returns x_j whenever we have a $*$-input, and y_j whenever we have a $\dagger$-input. Thus, if we run algorithm $\mathcal{B}$ with this oracle operation (in superposition, including uncomputation), then we output $f(x) = 0$ on a $*$-input, and $f(y) = 1$ on a $\dagger$-input, with the same success probability as that of $\mathcal{A}$. As this query operation can be implemented with a constant number of calls to the query oracle O_w^{str}, we conclude that $\mathsf{QS}_{\mathsf{str}}(f) = O(\mathsf{Q}(f))$. ◀

It is not obvious how we can modify the above algorithm to also output the index where x and y differ. However, we can design such an algorithm using very different techniques, and give an upper bound on $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f)$ in terms of $\mathsf{QD}(f)$. To that end, we first prove a fundamental lemma that is similar to [11, Lemma 12] and [21, Lemma 3.1].

▶ **Lemma 3.** *Let $x \in \{0,1\}^n$ be an input and let $B \subseteq [n]$. Let $\mathcal{A}$ be a T-query quantum algorithm that accepts x and rejects x_B with high probability, or more generally produces output states that are a constant distance apart in trace distance for x and x_B. Let p_t, resp. p_t^B, be the probability that, when $\mathcal{A}$ is run on x, resp. x_B, up until, but not including, the t-th query and then measured, it is found to be querying a position $i \in B$. Then,*

$$\sum_{t=1}^{T}(p_t + p_t^B) = \Omega(1).$$

Proof. We write $y = x_B$, and we let $|\psi_x^t\rangle$ and $|\psi_y^t\rangle$ be the states right before the tth query, when we run $\mathcal{A}$ on inputs x and y, respectively. We also let $|\psi_x\rangle$ and $|\psi_y\rangle$ be the final states of the algorithm run on x and y, respectively. Since $\mathcal{A}$ can distinguish x and y with high probability, we observe that $|\psi_x\rangle$ and $|\psi_y\rangle$ must be far apart, i.e., their inner product must satisfy

$$1 - |\langle\psi_x|\psi_y\rangle| = \Omega(1).$$

For every $j \in [n]$, let $\mathcal{H}_j$ be the subspace of the state space of $\mathcal{A}$ that queries the jth bit of the input. In other words, we let $\mathcal{H}_j$ be the span of all states that pick up a phase of $(-1)^{x_j}$, when the algorithm $\mathcal{A}$ calls the oracle O_x. We let Π_j be the projector on this subspace.

Next, we let $\mathcal{H}_B$ be the subspace that contains all $\mathcal{H}_j$'s with $j \in B$. In other words, we write $\mathcal{H}_B = \oplus_{j\in B}\mathcal{H}_j$. We immediately observe that the projector onto $\mathcal{H}_B$, denoted by Π_B, satisfies $\Pi_B = \sum_{j\in B}\Pi_j$. Note that it is exactly the subspace $\mathcal{H}_B$ on which the oracles O_x and O_y act differently. So, intuitively, if a state $|\psi\rangle$ has a big component in $\mathcal{H}_B$, then $O_x|\psi\rangle$ and $O_y|\psi\rangle$ will be far apart from each other.

Now, we define $p_{x,t} := \left\|\Pi_B|\psi_x^t\rangle\right\|^2$, i.e., the squared overlap of the state $|\psi_x^t\rangle$ with the subspace $\mathcal{H}_B$. Intuitively, if we were to interrupt the algorithm $\mathcal{A}$ run on input x right before the tth query, and we were to measure the query register, then the probability of measuring a $j \in B$ is $p_{x,t}$.

The crucial observation that we make is that

$$\begin{aligned}\left|\langle\psi_x^t|\psi_y^t\rangle\right| - \left|\langle\psi_x^{t+1}|\psi_y^{t+1}\rangle\right| &\leq \left|\langle\psi_x^t|\psi_y^t\rangle - \langle\psi_x^{t+1}|\psi_y^{t+1}\rangle\right| = \left|\langle\psi_x^t|\psi_y^t\rangle - \langle\psi_x^t|O_x^\dagger O_y|\psi_y^t\rangle\right| \\ &= \left|\langle\psi_x^t|(I - O_x^\dagger O_y)|\psi_y^t\rangle\right| = 2\left|\langle\psi_x^t|\Pi_B|\psi_y^t\rangle\right| \leq 2\left\|\Pi_B|\psi_x^t\rangle\right\| \cdot \left\|\Pi_B|\psi_y^t\rangle\right\| \\ &= 2\sqrt{p_{x,t}\cdot p_{y,t}} \leq p_{x,t} + p_{y,t}.\end{aligned}$$

Here, we used the triangle inequality, the Cauchy-Schwarz inequality, and the AM-GM inequality, in order. Finally, we observe that the initial states for algorithm $\mathcal{A}$ run on x and y are the same, and so $|\langle\psi_x^1|\psi_y^1\rangle| = 1$. Thus, identifying $|\psi_x^{T+1}\rangle$ with $|\psi_x\rangle$, and similarly for y, we obtain that

$$1 - |\langle\psi_x|\psi_y\rangle| = \sum_{t=1}^{T} |\langle\psi_x^t|\psi_y^t\rangle| - |\langle\psi_x^{t+1}|\psi_y^{t+1}\rangle| \leq \sum_{t=1}^{T}(p_{x,t} + p_{y,t}). \qquad ◀$$

We now show how the above lemma can be used to prove a connection between $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f)$ and $\mathsf{QD}(f)$.

▶ **Theorem 17.** *Let* $f : \{0,1\}^n \to \{0,1\}$. *We have* $\mathsf{QS}_{\mathsf{str}}^{\mathsf{ind}}(f) = O(\mathsf{QD}(f)^{3/2})$.

Proof. Suppose we have an algorithm $\mathcal{A}$ that distinguishes between input $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$ with high probability in T queries. Then, we construct an algorithm $\mathcal{B}$ that works in the strong sabotage input model, and finds an index $j \in [n]$ where $x_j \neq y_j$.

First, consider the following procedure. We pick an input x or y with probability $1/2$, and we pick a time step $t \in \{1, \ldots, T\}$ uniformly at random. We run $\mathcal{A}$ until right before the tth query, and then we measure the query register to obtain an index $j \in [n]$. From

Lemma 3, we obtain that the probability that $x_j \neq y_j$ is lower bounded by $\Omega(1/T)$.[2] Thus, running this algorithm $O(T)$ times would suffice to find a $j \in [n]$ such that $x_j \neq y_j$, with high probability.

However, we can do slightly better than that. Note that once the algorithm gives us an index $j \in [n]$, it takes just one query (to z) to find out if $z_j \in \{*, \dagger\}$. Thus, we can use amplitude amplification, and use $O(\sqrt{T})$ iterations of the above procedure, to find a $j \in [n]$ such that $z_j \in \{*, \dagger\}$ (equivalently, $x_j \neq y_j$). Each application of the procedure takes $O(T)$ queries to implement in the worst case. Thus, the final query complexity is $O(T^{3/2})$. ◀

Combining the above result with [11, Proposition 6], which states that $\mathsf{QD}(f) = O(\mathsf{Q}(f))$, immediately yields the following theorem.

▶ **Theorem 2.** *Let n be a positive integer, let $D \subseteq \{0,1\}^n$, and let $f : D \to \{0,1\}$ be a (partial) Boolean function. Then, $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(f) = O(\mathsf{Q}(f)^{1.5})$.*

5 $\mathsf{QS}_{\mathsf{str}}(f)$ vs. $\sqrt{\mathsf{fbs}(f)}$

So far we have shown upper bounds on quantum sabotage complexity. In this section we show our lower bounds. We first show that $\mathsf{QS}_{\mathsf{str}}(f)$ (and hence $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(f), \mathsf{QS}_{\mathsf{weak}}(f), \mathsf{QS}^{\mathsf{ind}}_{\mathsf{weak}}(f)$ as well) is bounded from below by $\sqrt{\mathsf{fbs}(f)}$. This is a generalization of the known bound of $\mathsf{Q}(f) = \Omega(\sqrt{\mathsf{bs}(f)})$ [7]. In particular, this already implies that quantum sabotage complexity is polynomially related to quantum query complexity for all total Boolean functions f. Next observe that, unlike in the usual quantum query setting, this $\sqrt{\mathsf{fbs}(f)}$ lower bound is *tight* for standard functions such as Or, And, Majority and Parity because of the Grover-based $O(\sqrt{n})$ upper bound on the quantum sabotage complexity of all functions. This suggests the possibility of the quantum sabotage complexity of f being $\Theta(\sqrt{\mathsf{fbs}(f)})$ for all total f. In the next subsection we rule this out, witnessed by f as the Indexing function, for which we show the quantum sabotage complexity to be $\Theta(\mathsf{fbs}(f))$.

5.1 A general lower bound

In the appendix we show that $\mathsf{RS}(f) = \Omega(\mathsf{fbs}(f))$, as well as the quantum bound of $\mathsf{Q}(f) = \Omega(\sqrt{\mathsf{fbs}(f)})$. We now wish to follow the same approach as in the proof of the latter bound for proving a lower bound on $\mathsf{QS}_{\mathsf{str}}(f)$. To that end, we know that $\mathsf{QS}_{\mathsf{str}}(f) := \mathsf{Q}(f^{\mathsf{str}}_{\mathsf{sab}}) = \Omega(\mathsf{ADV}^{+}(f^{\mathsf{str}}_{\mathsf{sab}}))$ [4], so it remains to show that $\mathsf{ADV}^{+}(f^{\mathsf{str}}_{\mathsf{sab}}) = \Omega(\sqrt{\mathsf{fbs}(f)})$. Thus, we adapt the proof of Lemma 24 in the sabotaged setting.

▶ **Lemma 18.** *Let n be a positive integer, $D \subseteq \{0,1\}^n$ and $f : D \to \{0,1\}$. Then, $\mathsf{ADV}^{+}(f^{\mathsf{str}}_{\mathsf{sab}}) = \Omega(\sqrt{\mathsf{fbs}(f)})$.*

Proof. Let x be the instance for which $\mathsf{fbs}(f) = \mathsf{fbs}(f, x)$, and let $(w_y)_{y \in Y}$ be the optimal weight assignment (see Definition 8). Similar to the proof of Lemma 24, we generate a (non-negative weight) adversary matrix, $\Gamma \in \mathbb{R}^{D^{\mathsf{str}}_{\mathsf{sab}} \times D^{\mathsf{str}}_{\mathsf{sab}}}$. We define Γ to be the all-zeros matrix, except for the instances where $((x, y, *), (x, y', \dagger))$ and $((x, y, \dagger), (x, y', *))$, with $y, y' \in Y$, where we define it to be

$$\Gamma[(x, y, *), (x, y', \dagger)] = \Gamma[(x, y', \dagger), (x, y, *)] = \sqrt{w_y w_{y'}}.$$

[2] It is important to remark here that we have access to our strong sabotage oracle now (Definition 14). Recall that there are four registers: $|j\rangle, |b_x\rangle, |b_y\rangle, |b_z\rangle$. When we say "run $\mathcal{A}$" with this oracle, we mean all operations act as identity on the last register.

Again Γ has a simple sparsity pattern. Only the rows and columns that are labeled by $(x,y,*)$ and $(x,y,\dagger)$, with $y \in Y$, are non-zero. Additionally, observe from the definition of the matrix entries that for any $y, y' \in Y$, we have

$$\Gamma[(x,y,*),(x,y',\dagger)] = \sqrt{w_y w_{y'}} = \Gamma[(x,y,\dagger),(x,y',*)].$$

Thus, in every 2×2-block formed by rows $(x,y,*)$ and $(x,y,\dagger)$ and columns $(x,y',*)$ and $(x,y,\dagger)$, we have that the two diagonal elements are 0, and the two off-diagonal elements are equal. Hence, by removing unimportant rows and columns that are completely zero, we can rewrite Γ as

$$\Gamma = \Gamma' \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \text{where} \qquad \Gamma' \in \mathbb{R}^{Y \times Y}, \qquad \text{with} \qquad \Gamma'[y,y'] = \sqrt{w_y w_{y'}}.$$

It now becomes apparent that Γ' is of rank 1. Indeed, it is the outer product of a vector $\sqrt{w} \in \mathbb{R}^Y$, that contains the entries $\sqrt{w_y}$ at every index labeled by y. From some matrix arithmetic, we now obtain

$$\|\Gamma\| = \|\Gamma'\| \cdot 1 = \left\|\sqrt{w}\sqrt{w}^T\right\| = \left\|\sqrt{w}\right\|^2 = \sum_{y \in Y} w_y = \mathsf{fbs}(f).$$

Thus, it remains to prove that $\|\Gamma \circ \Delta_j\| = O(\sqrt{\mathsf{fbs}(f)})$, for all $j \in [n]$. Indeed, then we can scale our matrix Γ down by $\Theta(\sqrt{\mathsf{fbs}(f)})$ so that it is feasible for the optimization program in Definition 9, and the objective value will then become $\Theta(\sqrt{\mathsf{fbs}(f)})$ as predicted.

Let $j \in [n]$. To compute $\|\Gamma \circ \Delta_j\|$, we look at its sparsity pattern. Observe that whenever we query the jth bit of $(x,y,*)$ and $(x,y',\dagger)$, we obtain the tuples (x_j, y_j, z_j) and (x_j, y_j', z_j'). If $y_j \neq y_j'$, then it is clear that these tuples are not the same. Similarly, if $x_j \neq y_j = y_j'$, then both z_j's will be different, as one will be a $*$ and the other will be a $\dagger$. Thus, the queried tuples are only identical whenever $x_j = y_j = y_j'$, which implies that we can rewrite

$$\Gamma \circ \Delta_j = \Gamma_j' \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{where} \quad \Gamma_j' \in \mathbb{R}^{Y \times Y}, \quad \text{with} \quad \Gamma_j'[y,y'] = \begin{cases} 0, & \text{if } y_j = y_j' = x_j, \\ \sqrt{w_y w_{y'}}, & \text{otherwise.} \end{cases}$$

We now let $Y_0 = \{y \in Y : x_j \neq y_j\}$ and $Y_1 = \{y \in Y : x_j = y_j\}$. We interpret Γ_j' as a 2×2-block matrix, where the first rows and columns are indexed by Y_0 and the last ones are indexed by Y_1. Then, Γ_j' takes on the shape $\Gamma_j' = \begin{bmatrix} A & B \\ B^T & 0 \end{bmatrix}$, with $A \in \mathbb{R}^{Y_0 \times Y_0}$ and $B \in \mathbb{R}^{Y_0 \times Y_1}$, defined as

$$A[y,y'] = \sqrt{w_y w_{y'}}, \qquad \text{and} \qquad B[y,y'] = \sqrt{w_y w_{y'}}.$$

We observe that

$$\|\Gamma \circ \Delta_j\| = \left\|\Gamma_j'\right\| \cdot 1 \leq \|A\| + \|B\|,$$

and hence it remains to compute $\|A\|$ and $\|B\|$.

We now define the vectors $\sqrt{w_0} \in \mathbb{R}^{Y_0}$ and $\sqrt{w_1} \in \mathbb{R}^{Y_1}$, defined by $(\sqrt{w_0})_y = \sqrt{w_y}$, and $(\sqrt{w_1})_y = \sqrt{w_y}$. We observe that $A = \sqrt{w_0}\sqrt{w_0}^T$, and $B = \sqrt{w_0}\sqrt{w_1}^T$. Thus, we obtain

$$\|A\|^2 = \left\|\sqrt{w_0}\sqrt{w_0}^T\right\|^2 = \|\sqrt{w_0}\|^4 = \left[\sum_{y \in Y_0} w_y\right]^2 = \left[\sum_{\substack{y \in Y \\ x_j \neq w_j}} w_y\right]^2 \leq 1,$$

and similarly

$$\|B\|^2 = \left\|\sqrt{w_0}\sqrt{w_1}^T\right\|^2 = \|\sqrt{w_0}\| \cdot \|\sqrt{w_1}\|^2 = \left[\sum_{\substack{y\in Y\\ x_j\neq y_j}} w_y\right] \cdot \left[\sum_{\substack{y\in Y\\ x_j=y_j}} w_y\right] \leq 1 \cdot \mathsf{fbs}(f). \quad ◀$$

The proof of Theorem 5 now follows as a simple corollary from this lemma and the fact that $\mathsf{QS}_{\mathsf{str}}(f) := \mathsf{Q}(f^{\mathsf{str}}_{\mathsf{sab}}) = \Omega(\mathsf{ADV}^+(f^{\mathsf{str}}_{\mathsf{sab}}))$, where the last bound follows from the positive-weighted adversary lower bound of Ambainis [4].

5.2 A stronger lower bound for Indexing

As observed in the discussion following Theorem 2, we have $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{weak}}(f)$ (and hence $\mathsf{QS}_{\mathsf{weak}}(f)$ and $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(f)$ and $\mathsf{QS}_{\mathsf{str}}(f)$) is $O(\sqrt{n})$ for all $f : \{0,1\}^n \to \{0,1\}$. In particular, the $\sqrt{\mathsf{fbs}(f)}$ lower bound for $\mathsf{QS}_{\mathsf{str}}(f)$ is tight for standard functions like AND, OR, Parity, Majority. In view of this it is feasible that $\mathsf{QS}_{\mathsf{str}}(f) = O(\sqrt{\mathsf{fbs}(f)})$ for all total $f : \{0,1\}^n \to \{0,1\}$. In the remaining part of this section, we rule this out, witnessed by the Indexing function.

We use Ambainis' adversary method to prove lower bounds on quantum query complexity [3].

▶ **Lemma 19** ([3, Theorem 6.1]). *Let $f : \{0,1,*,\dagger\}^k \to \{0,1\}$ be a (partial) Boolean function. Let $X, Y \subseteq \{0,1,*,\dagger\}^k$ be two sets of inputs such that $f(x) \neq f(y)$ if $x \in X$ and $y \in Y$. Let $R \subseteq X \times Y$ be nonempty, and satisfy:*

- *For every $x \in X$ there exist at least m_X different $y \in Y$ such that $(x,y) \in R$.*
- *For every $y \in Y$ there exist at least m_Y different $x \in X$ such that $(x,y) \in R$.*

Let $\ell_{x,i}$ be the number of $y \in Y$ such that $(x,y) \in R$ and $x_i \neq y_i$, and similarly for $\ell_{y,i}$. Let $\ell_{\max} = \max_{i\in[k]} \max_{(x,y)\in R, x_i\neq y_i} \ell_{x,i}\ell_{y,i}$. Then any algorithm that computes f with success probability at least $2/3$ uses $\Omega\left(\sqrt{\frac{m_X m_Y}{\ell_{\max}}}\right)$ quantum queries to the input function.

Define the Indexing function as follows. For a positive integer $n > 0$, define the function $\mathsf{IND}_n : \{0,1\}^n \times \{0,1\}^{2^n} \to \{0,1\}$ as $\mathsf{IND}_n(a,b) = b_{\mathsf{bin}(a)}$, where $\mathsf{bin}(a)$ denotes the integer in $[2^n]$ whose binary representation is a.

We first note that the fractional block sensitivity is easily seen to be bounded from below by block sensitivity, and bounded from above by deterministic (in fact randomized) query complexity. Both the block sensitivity (in fact, sensitivity) and deterministic query complexity of IND_n are easily seen to be $n+1$, implying $\mathsf{fbs}(\mathsf{IND}_n) = n+1$.

▶ **Theorem 20.** *Let n be a positive integer. Then,* $\mathsf{QS}_{\mathsf{weak}}(\mathsf{IND}_n) = \Omega(n)$.

Proof. Recall from Definition 15 that $\mathsf{QS}_{\mathsf{weak}}(\mathsf{IND}_n) = \mathsf{Q}(\mathsf{IND}_{n,\mathsf{sab}})$. We construct a hard relation for $f = \mathsf{IND}_{n,\mathsf{sab}}$ and use Lemma 19. Recall that this relation must contain pairs of inputs. For each pair, the function must evaluate to different outputs. For ease of notation we first define the pairs of inputs $(a_1,b_1),(a_2,b_2)$ in the relation, and then justify that these inputs are indeed in S_* and $S_\dagger$, respectively.

Define $(a_1,b_1),(a_2,b_2) \in R$ if and only if all of the following hold true:

1. $(a_1,b_1) \in f^{-1}(0)$ (i.e., $(a_1,b_1) \in S_*$ for IND_n), $(x_2,y_2) \in f^{-1}(1)$ (i.e., $(a_2,b_2) \in S_\dagger$ for IND_n),
2. $|a_1 \oplus a_2| = 2$, i.e., the Hamming distance between a_1 and a_2 is 2,
3. b_1 is all-0, except for the $\mathsf{bin}(a_1)$'th index, which is $*$,
4. b_2 is all-0, except for the $\mathsf{bin}(a_2)$'th index, which is $\dagger$.

First note that $(a_1, b_1) = [(a_1, 0^{2^n}), (a_1, e_{a_1}), *]$, where $e_{a_1} \in \{0,1\}^{2^n}$ is the all-0 string except for the $\mathsf{bin}(a_1)$'th location, which is a 1. Similarly, $(a_2, b_2) = [(a_2, 0^{2^n}), (a_2, e_{a_2}), \dagger]$. Thus, (a_1, b_1) and (a_2, b_2) are in S_* and $S_\dagger$, respectively. In particular, in the language of Lemma 19 we have

$$X = \left\{(a,b) \in \{0,1\}^n \times \{0,1,*\}^{2^n} : b \text{ is all-0 except for the } \mathsf{bin}(a)\text{'th index, which is } *\right\}. \tag{5}$$

Similarly,

$$Y = \left\{(a,b) \in \{0,1\}^n \times \{0,1,\dagger\}^{2^n} : b \text{ is all-0 except for the } \mathsf{bin}(a)\text{'th index, which is } \dagger\right\}. \tag{6}$$

We now analyze the quantities m_X, m_Y and $\ell_{\max}$ from Lemma 19.

1. $m_X = m_Y = \binom{n}{2}$: Consider $(a,b) \in X$. The number of elements $(a',b') \in Y$ such that $((a,b),(a',b') \in R)$ is simply the number of strings a' that have a Hamming distance of 2 from a, since each such a' corresponds to exactly one b' with $((a,b),(a',b') \in R)$, where b' is the all-0 string except for the $\mathsf{bin}(a)'$th location which is a †. Thus $m_X = \binom{n}{2}$. The argument for m_Y is essentially the same.
2. $\ell_{\max} = \min\left\{\binom{n}{2}, (n-1)^2\right\}$: We consider two cases.
   a. $i \in [n]$: Fix $((a,b),(a',b')) \in R$ with $a_i \neq a'_i$. Recall that $\ell_{(a,b),i}$ is the number of $(a'',b'') \in Y$ such that $((a,b),(a'',b'')) \in R$ and $a_i \neq a''_i$. Following a similar logic as in the previous argument, this is simply the number of a'' that have Hamming distance 2 from a and additionally satisfy $a_i \neq a''_i$. There are $n-1$ possible locations for the other difference between a and a'', so $\ell_{(a,b),i} = n-1$ in this case. Essentially the same argument shows $\ell_{(a',b'),i} = n-1$, and so $\ell_{(a,b),i} \cdot \ell_{(a',b'),i} = (n-1)^2$.
   b. $i \in [2^n]$: Fix $((a,b),(a',b')) \in R$ with $b_i \neq b'_i$. By the structure of R, b is the all-0 string except for the $\mathsf{bin}(a)$'th location which is a $*$, and b' is the all-0 string except for the $\mathsf{bin}(a')$'th location which is a †. Thus $i \in \{\mathsf{bin}(a), \mathsf{bin}(a')\}$. Without loss of generality, assume $i = \mathsf{bin}(a)$, and thus $b_{\mathsf{bin}(a)} = *$. For each a'' with Hamming distance 2 from a, we have $((a,b),(a'',b'')) \in R$ where b'' is all-0 except for the a''th index which is †. In particular, $b''_{\mathsf{bin}(a)} = 0$. So we have $\ell_{(a,b),i} = \binom{n}{2}$. On the other hand, we have $b'_{\mathsf{bin}(a)} = 0$. So the only (a'',b'') with $((a'',b''),(a',b')) \in R$ and with $b''_{\mathsf{bin}(a)} \neq b'_{\mathsf{bin}(a)}$ is $(a'',b'') = (a,b)$. Thus $\ell_{(a',b'),i} = 1$.

Lemma 19 then implies

$$\mathsf{QS}_{\mathsf{weak}}(\mathsf{IND}_n) = \mathsf{Q}(\mathsf{IND}_{n,\mathsf{sab}}) = \mathsf{Q}(f) = \Omega\left(\sqrt{\frac{\binom{n}{2}^2}{\min\left\{\binom{n}{2}, (n-1)^2\right\}}}\right) = \Omega(n), \tag{7}$$

proving the theorem. ◀

This proof can easily be adapted to also yield a lower bound of $\mathsf{QS}_{\mathsf{str}}(\mathsf{IND}_n) = \Omega(n)$. We include a proof in the appendix for completeness. As a corollary we obtain the following.

▶ **Corollary 21.** *Let n be a positive integer. Then, $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{str}}(\mathsf{IND}_n) = \Omega(n)$ and $\mathsf{QS}^{\mathsf{ind}}_{\mathsf{weak}}(\mathsf{IND}_n) = \Omega(n)$.*

6 Open questions

In this paper we studied the quantum sabotage complexity of Boolean functions, which we believe is a natural extension of randomized sabotage complexity introduced by Ben-David and Kothari [10]. We note here that in a subsequent work [11] they also defined a quantum analog, but this is fairly different from the notion that we studied in this paper.

We argued, by showing several results, that it makes sense to consider four different models depending on the access to input, and output requirements. While it is easily seen that the randomized sabotage complexity of a function remains (asymptotically) the same regardless of the choice of model (see Proposition 22), such a statement is not clear in the quantum setting. In our view, the most interesting problem left open from our work is to prove or disprove that even the quantum sabotage complexity of a function is asymptotically the same in all of these four models. It would also be interesting to see tight polynomial relationships between the various quantum sabotage complexities and quantum query complexity.

References

1 Scott Aaronson, Shalev Ben-David, and Robin Kothari. Separations in query complexity using cheat sheets. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 863–876. ACM, 2016. doi:10.1145/2897518.2897644.

2 Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of huang's sensitivity theorem. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1330–1342. ACM, 2021. doi:10.1145/3406325.3451047.

3 Andris Ambainis. Quantum lower bounds by quantum arguments. *J. Comput. Syst. Sci.*, 64(4):750–767, 2002. Earlier version in STOC'00. doi:10.1006/JCSS.2002.1826.

4 Andris Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Syst. Sci.*, 72(2):220–238, 2006. doi:10.1016/J.JCSS.2005.06.006.

5 Anurag Anshu, Dmitry Gavinsky, Rahul Jain, Srijita Kundu, Troy Lee, Priyanka Mukhopadhyay, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 93 of *LIPIcs*, pages 10:1–10:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.FSTTCS.2017.10.

6 Andrew Bassilakis, Andrew Drucker, Mika Göös, Lunjia Hu, Weiyun Ma, and Li-Yang Tan. The power of many samples in query complexity. In *47th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 168 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.9.

7 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001. Earlier version in FOCS'98. doi:10.1145/502090.502097.

8 Shalev Ben-David and Eric Blais. A tight composition theorem for the randomized query complexity of partial functions: Extended abstract. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 240–246. IEEE, 2020. doi:10.1109/FOCS46700.2020.00031.

9 Shalev Ben-David, Eric Blais, Mika Göös, and Gilbert Maystre. Randomised composition and small-bias minimax. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 624–635. IEEE, 2022. doi:10.1109/FOCS54457.2022.00065.

10 Shalev Ben-David and Robin Kothari. Randomized query complexity of sabotaged and composed functions. *Theory Comput.*, 14(1):1–27, 2018. Earlier version in ICALP'16. doi:10.4086/TOC.2018.V014A005.

11 Shalev Ben-David and Robin Kothari. Quantum distinguishing complexity, zero-error algorithms, and statistical zero knowledge. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC*, volume 135 of *LIPIcs*, pages 2:1–2:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.TQC.2019.2.

12 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.

13 Sourav Chakraborty, Chandrima Kayal, Rajat Mittal, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. On the composition of randomized query complexity and approximate degree. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 275 of *LIPIcs*, pages 63:1–63:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.APPROX/RANDOM.2023.63.

14 Dmitry Gavinsky, Troy Lee, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity via max-conflict complexity. In *46th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 132 of *LIPIcs*, pages 64:1–64:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.64.

15 Justin Gilmer, Michael E. Saks, and Srikanth Srinivasan. Composition limits and separating examples for some boolean function complexity measures. *Comb.*, 36(3):265–311, 2016. doi:10.1007/S00493-014-3189-X.

16 Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Trans. Comput. Theory*, 10(1):4:1–4:20, 2018. Earlier version in ICALP'17. doi:10.1145/3170711.

17 Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 212–219. ACM, 1996. doi:10.1145/237814.237866.

18 Peter Høyer, Troy Lee, and Robert Spalek. Negative weights make adversaries stronger. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 526–535. ACM, 2007. doi:10.1145/1250790.1250867.

19 Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949–955, 2019.

20 Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chic. J. Theor. Comput. Sci.*, 2016, 2016. URL: http://cjtcs.cs.uchicago.edu/articles/2016/8/contents.html.

21 Sophie Laplante and Frédéric Magniez. Lower bounds for randomized and quantum query complexity using kolmogorov arguments. *SIAM J. Comput.*, 38(1):46–62, 2008. doi:10.1137/050639090.

22 Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 344–353. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.75.

23 Noam Nisan. CREW prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991. Earlier version in STOC'89. doi:10.1137/0220062.

24 Swagato Sanyal. Randomized query composition and product distributions. In *41st International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 289 of *LIPIcs*, pages 56:1–56:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.56.

25 Robert Spalek and Mario Szegedy. All quantum adversary methods are equivalent. *Theory Comput.*, 2(1):1–18, 2006. doi:10.4086/TOC.2006.V002A001.

26 Ronald de Wolf. Quantum computing: Lecture notes, 2023. Version 5. arXiv:1907.09415.

A Randomized sabotage complexity

By comparing the definitions in Definition 15 to those in [10], we observe that $\mathsf{RS} = \mathsf{RS}_{\mathsf{weak}}$. However, we argue that for all functions, the other randomized complexity measures are equal up to constants.

▶ **Proposition 22.** *Let n be a positive integer, and $D \subseteq \{0,1\}^n$. Let $f : D \to \{0,1\}$ be a (partial) Boolean function. Then,*

$$\mathsf{RS}(f) := \mathsf{RS}_{\mathsf{weak}}(f) = \Theta(\mathsf{RS}_{\mathsf{str}}(f)) = \Theta(\mathsf{RS}^{\mathsf{ind}}_{\mathsf{weak}}(f)) = \Theta(\mathsf{RS}^{\mathsf{ind}}_{\mathsf{str}}(f)).$$

Proof. It is clear that we have $\mathsf{RS}_{\mathsf{str}}(f) \leq \mathsf{RS}_{\mathsf{weak}}(f)$, and $\mathsf{RS}^{\mathsf{ind}}_{\mathsf{str}}(f) \leq \mathsf{RS}^{\mathsf{ind}}_{\mathsf{weak}}(f)$ because the input model is strictly stronger. For the opposite directions, suppose that we have an algorithm $\mathcal{A}$ in the strong input model. Let $x \in f^{-1}(0)$ and $y \in f^{-1}(1)$. Note that for every $j \in [n]$ on queries where $x_j = y_j$, the oracle outputs $(0,0,0)$ or $(1,1,1)$. Thus, on these indices, the same algorithm in the weak model would yield the outputs 0 or 1, respectively. Moreover, with high probability $\mathcal{A}$ must query a $j \in [n]$ where $x_j \neq y_j$ at some point, since otherwise it cannot distinguish $[x,y,*]$ and $[x,y,\dagger]$. This gives us the following $\mathsf{RS}^{\mathsf{ind}}_{\mathsf{weak}}$ algorithm: run $\mathcal{A}$ constantly many times (with strong queries replaced by weak queries) until we query a $j \in [n]$ where $x_j \neq y_j$. At that point, we can interrupt the algorithm and we have enough information to solve the sabotage problem, in both the decision and index model. Thus $\mathsf{RS}_{\mathsf{weak}}(f) = O(\mathsf{RS}_{\mathsf{str}}(f))$, and $\mathsf{RS}^{\mathsf{ind}}_{\mathsf{weak}}(f) = O(\mathsf{RS}^{\mathsf{ind}}_{\mathsf{str}}(f))$.

It remains to show that $\mathsf{RS}_{\mathsf{str}}(f) = \Theta(\mathsf{RS}^{\mathsf{ind}}_{\mathsf{str}}(f))$. By the definitions of the models, we have $\mathsf{RS}_{\mathsf{str}}(f) \leq \mathsf{RS}^{\mathsf{ind}}_{\mathsf{str}}(f)$. On the other hand, suppose that we have an algorithm $\mathcal{B}$ that figures out whether we have a $*$- or $\dagger$-input. Then, by the same logic as given in the previous paragraph, with high probability, $\mathcal{B}$ must have encountered at least one $*$ or $\dagger$, so we can read back in the transcript to find out at which position it made that query. Repeating $\mathcal{B}$ a constant number of times this way yields $\mathsf{RS}^{\mathsf{ind}}_{\mathsf{str}}(f) = O(\mathsf{RS}_{\mathsf{str}}(f))$. ◀

Note that we did not do a formal analysis of success probabilities in the above argument, but this is not hard to do. We omit precise details for the sake of brevity.

B Lower bounds on $\mathsf{QS}(f)$ and $\mathsf{RS}(f)$ in terms of $\mathsf{fbs}(f)$

To the best of our knowledge, the only lower bound on randomized sabotage complexity $\mathsf{RS}(f)$ in the existing literature is $\Omega(\mathsf{QD}(f))$ [11, Corollary 11]. We remark that $\mathsf{RS}(f)$ can also be lower bounded by $\mathsf{fbs}(f)$. This result was essentially shown in [10, Theorem 7.2], but we include a proof here for completeness.

▶ **Lemma 23.** $\mathsf{fbs}(f) = O(\mathsf{RS}(f))$.

Proof. From [10, Theorem 3.3], we find that $\mathsf{R}(f_{\mathsf{sab}}) = \Theta(\mathsf{RS}(f))$, and so it suffices to show that $\mathsf{fbs}(f) \leq 10\mathsf{R}(f_{\mathsf{sab}})$. By Yao's minimax principle, it suffices to exhibit a distribution over inputs for which any deterministic algorithm fails to compute f_{sab} correctly with probability at least $2/3$ in less than $\mathsf{fbs}(f)/10$ queries.

Without loss of generality, let $x \in f^{-1}(0)$ be the instance for which fractional block-sensitivity is maximized, let $Y = f^{-1}(1)$, and let $(w_y)_{y \in Y}$ be the optimal weight assignment in the fractional block sensitivity linear program (see Definition 8). We can similarly think of every $y \in Y$ as obtained by flipping the bits in x that belong to a sensitive block B. We denote $y = x_B$, and write $w_y = w_B$.

Let T be a deterministic algorithm for f_{sab} with query complexity less than $\mathsf{fbs}(f)/10$, and let μ be a distribution defined as follows: for all $B \subseteq [n]$ that is a sensitive block for f, let $\mu([x, x_B, *]) = \mu([x, x_B, \dagger]) = w_B/(2\mathsf{fbs}(f))$. Since $\sum_{B \subseteq [n]} w_B = \mathsf{fbs}(f)$ where the sum is over all sensitive blocks for f, μ forms a legitimate probability distribution.

Consider a leaf L of T, and suppose that the output at L is $b \in \{0,1\}$. By the definition of fractional block sensitivity, we know that for any $j \in [n]$, we have $\sum_{B : j \in B} w_B \leq 1$. Thus, by a union bound, the μ-mass of inputs supported by our distribution (of the form $[x, x_B, *]$ or $[x, x_B, \dagger]$) reaching L, and such that no bit of B has been read on this path is at least $1 - \left(\frac{\mathsf{fbs}(f)}{10} \cdot \frac{1}{\mathsf{fbs}(f)}\right) \geq 9/10$. Note that for each such B, both the input $[x, x_B, *]$ and $[x, x_B, \dagger]$ reach L. Since μ allotted equal mass to each such pair, this means an error is made at L for inputs with μ-mass at least $9/20 > 1/3$, concluding the proof. ◀

We now turn to lower bounding $\mathsf{QS}_{\mathsf{str}}(f)$ by $\Omega(\sqrt{\mathsf{fbs}(f)})$. To that end, recall that for any Boolean function f, we have the chain of inequalities $\mathsf{Q}(f) = \Omega(\mathsf{ADV}^+(f)) = \Omega(\sqrt{\mathsf{fbs}(f)})$. The first inequality can be found in [25], for example. While we believe the second inequality is known, we were again unable to find a published proof of it. We include a proof here for completeness.

▶ **Lemma 24.** *Let n be a positive integer, $D \subseteq \{0,1\}^n$, and $f : D \to \{0,1\}$ be a (partial) Boolean function. Then, $\sqrt{\mathsf{fbs}(f)} = O(\mathsf{ADV}^+(f))$.*

Proof. Recall the definition of fractional block sensitivity, Definition 8. Let x be the input to f for which $\mathsf{fbs}(f, x) = \mathsf{fbs}(f)$. Let $(w_y)_{y \in Y}$ be the optimal solution of the linear program. We now define an adversary matrix Γ, that forms a feasible solution to the optimization program in Definition 9. In particular, we define $\Gamma \in \mathbb{R}^{D \times D}$ as the all-zeros matrix, except for the entries (x, y) and (y, x) where $y \in Y$, which we define to be

$$\Gamma[x, y] = \Gamma[y, x] = \sqrt{w_y}.$$

The matrix Γ we constructed is very sparse. It is only non-zero in the row and column that is indexed by x. Moreover, $\Gamma[x, x] = 0$ as well. In particular, this makes computing its norm quite easy, we just have to compute the ℓ_2-norm of the column indexed by x.

For any $j \in [n]$, we thus find

$$\|\Gamma \circ \Delta_j\|^2 = \sum_{\substack{y \in Y \\ x_j \neq y_j}} \Gamma[x, y]^2 = \sum_{\substack{y \in Y \\ x_j \neq y_j}} w_y \leq 1,$$

and so Γ is a feasible solution to the optimization program in Definition 9. Finally, we have

$$\mathsf{ADV}^+(f)^2 \geq \|\Gamma\|^2 = \sum_{y \in Y} \Gamma[x, y]^2 = \sum_{y \in Y} w_y = \mathsf{fbs}(f). \quad ◀$$

C Missing proof

In Theorem 20 we showed using the adversary method that $\mathsf{QS}_{\mathsf{weak}}(\mathsf{IND}_n) = \Omega(n)$. We now show that the same proof can be adapted to show the same lower bound on $\mathsf{QS}_{\mathsf{str}}(\mathsf{IND}_n)$ as well.

▶ **Corollary 25.** *Let n be a positive integer. Then, $\mathsf{QS}_{\mathsf{str}}(\mathsf{IND}_n) = \Omega(n)$.*

Proof. Let $N = n + 2^n$. In the proof of Theorem 20 we constructed a hard relation R for $f = \mathsf{IND}_{n,\mathsf{sab}}$ to show a $\Omega(n)$ lower bound in the weak sabotage model. Using R we construct a hard relation for $\mathsf{IND}^{\mathsf{str}}_{n,\mathsf{sab}}$, which we denote by R^{str} in the strong sabotage model, and use Lemma 19. Define $((x, y, *), (x', y', \dagger)) \in R^{\mathsf{str}}$ if and only if all of the following hold true:[3]

1. $(x, y, *) \coloneqq ((x_j, y_j, z_j))_{j=1}^N \in S^{\mathsf{str}}_*$ for IND_n, $(x', y', \dagger) \coloneqq ((x'_j, y'_j, z'_j))_{j=1}^N \in S^{\mathsf{str}}_\dagger$ for IND_n,
2. $(z, z') \in R$.

Following the language of Lemma 19 we have

$$X^{\mathsf{str}} = \{(x, y, *) \coloneqq ((x_j, y_j, z_j))_{j=1}^N : z \in X \text{ as defined in Equation 5}\}. \tag{8}$$

Similarly,

$$Y^{\mathsf{str}} = \{(x, y, \dagger) \coloneqq ((x_j, y_j, z_j))_{j=1}^N : z \in Y \text{ as defined in Equation 6}\}. \tag{9}$$

We now analyze the quantities $m_{X^{\mathsf{str}}}$, $m_{Y^{\mathsf{str}}}$ and $\ell_{\max}$ from Lemma 19.

1. As described above, the way we construct R^{str} using elements of R ensures that $m_{X^{\mathsf{str}}} \geq m_X$ and $m_{Y^{\mathsf{str}}} \geq m_Y$. Additionally, for every $z \in X \cup Y$ there is exactly one pair in $\mathsf{IND}^{-1}(0) \times \mathsf{IND}^{-1}(1)$ for which z is the sabotaged input. Hence, $m_{X^{\mathsf{str}}} = m_X = \binom{n}{2}$ and $m_{Y^{\mathsf{str}}} = m_Y = \binom{n}{2}$.
2. $\ell_{\max} = \max\left\{\binom{n}{2}, (n-1)^2\right\}$: we consider two cases.
   a. $i \in [n]$: Fix an $(x, y, *), (x', y', \dagger) \in R^{\mathsf{str}}$ differing at an index $i \in [n]$. Recall that $\ell_{(x,y,*),i}$ denotes the number of $(x', y', \dagger) \in Y^{\mathsf{str}}$ such that $(x, y, *), (x', y', \dagger) \in R^{\mathsf{str}}$ and $(x_i, y_i, z_i) \neq (x'_i, y'_i, z'_i)$. The construction of sets $X^{\mathsf{str}}, Y^{\mathsf{str}}$ (which is in turn based on X, Y, respectively) ensures that for all $i \in [n]$ we have $x_i = y_i = z_i$ and $x'_i = y'_i = z'_i$. Hence, directly using the same arguments as in Item 2a, we get $\ell_{(x,y,*),i} = n - 1$ and $\ell_{(x',y',\dagger),i} = n - 1$, hence $\ell_{(x,y,*),i} \cdot \ell_{(x',y',\dagger),i} = (n-1)^2$.
   b. $i \in [n+1, N]$: Fix an $(x, y, *) \coloneqq ((x_j, y_j, z_j))_{j=1}^N, (x', y', \dagger) \coloneqq ((x'_j, y'_j, z'_j))_{j=1}^N \in R^{\mathsf{str}}$ differing at an index $i \in [n+1, N]$. By the structure of R and by extension of R^{str}, for strings $z \coloneqq (a, b)$ and $z' \coloneqq (a', b')$ the string b is all-0 string except for the $\mathsf{bin}(a)$'th location which is a $*$, and string b' is all-0 string except for the $\mathsf{bin}(a')$'th location which is a $\dagger$. Thus the only important case is for $i \in \{n + \mathsf{bin}(a), n + \mathsf{bin}(a')\}$. Without loss of generality, assume $i = n + \mathsf{bin}(a)$, and thus $z_i = b_{\mathsf{bin}(a)} = *$. Moreover, for every $z \in X \cup Y$ there is exactly one pair in $\mathsf{IND}^{-1}(0) \times \mathsf{IND}^{-1}(1)$ that sabotage as z. So, we have $\ell_{z,i} = \binom{n}{2}$ and $\ell_{z',i} = 1$ as $z'_i = b'_{\mathsf{bin}(a')} = 0$. Therefore, $\ell_{z,i} \cdot \ell_{z',i} = \binom{n}{2}$.

Lemma 19 then implies

$$\mathsf{QS}_{\mathsf{str}}(\mathsf{IND}_n) = \mathsf{Q}(\mathsf{IND}^{\mathsf{str}}_{n,\mathsf{sab}}) = \Omega\left(\sqrt{\frac{\binom{n}{2}^2}{\max\left\{\binom{n}{2}, (n-1)^2\right\}}}\right) = \Omega(n). \tag{10}$$

◀

[3] Recall that $(x, y, *)$ denotes $((x_j, y_j, z_j))_{j=1}^N$ where $z = [x, y, *]$ and $(x, y, \dagger)$ denotes $((x_j, y_j, z_j))_{j=1}^N$ where $z = [x, y, \dagger]$; see Definition 12.

The Isomorphism Problem of Power Graphs and a Question of Cameron

Bireswar Das
Indian Institute of Technology Gandhinagar, India

Jinia Ghosh
Indian Institute of Technology Gandhinagar, India

Anant Kumar
Indian Institute of Technology Gandhinagar, India

Abstract

We study the isomorphism problem of graphs that are defined in terms of groups, namely power graphs, directed power graphs, and enhanced power graphs. We design polynomial-time algorithms for the isomorphism problems for the power graphs, the directed power graphs and the enhanced power graphs arising from finite nilpotent groups. In contrast, no polynomial-time algorithm is known for the group isomorphism problem, even for nilpotent groups of class 2.

Our algorithms do not require the underlying groups of the input graphs to be given. A crucial step in our algorithms is to reconstruct the directed power graph from the given power graph or the enhanced power graph. The problem of efficiently computing the directed power graph from a power graph or an enhanced power graph is due to Cameron [IJGT'22]. Bubboloni and Pinzauti [Arxiv'22] gave a polynomial-time algorithm to reconstruct the directed power graph from a power graph. We give an efficient algorithm to compute the directed power graph from an enhanced power graph. The tools and techniques that we design are general enough to give a different algorithm to compute the directed power graph from a power graph as well.

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

Keywords and phrases Graph Isomorphism, Graphs defined on Groups, Power Graphs, Enhanced Power Graphs, Directed Power Graphs, Nilpotent Groups

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.20

Related Version *Full Version*: `https://arxiv.org/pdf/2305.18936` [12]

1 Introduction

Given two graphs as input, the graph isomorphism problem (GI) is to check if the graphs are isomorphic. Despite extensive research, the complexity status of GI is still open. The graph isomorphism problem is in NP but is very unlikely to be NP-hard as it is also in coAM [6].

Efficient algorithms for the graph isomorphism problem are known for several restricted graph classes, for example, graphs with bounded genus [28, 16], graphs with bounded degree [26], graphs with bounded eigenvalue multiplicity [3], graphs with bounded tree-width [5], graphs with bounded rank-width [20, 17].

Group-theoretic tools have played an important role in the design of efficient algorithms for the GI. Some of the early works on GI using the structure of groups non-trivially include the isomorphism algorithm for bounded degree graphs by Luks [26], and the graph canonization framework developed by Babai and Luks [4]. Babai developed sophisticated new techniques to give a quasipolynomial time isomorphism algorithm [2]. Currently, it is the best known isomorphism algorithm for general graphs. Group-theoretic machinery has been used to design faster isomorphism algorithms for bounded degree graphs by Grohe,

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 20; pp. 20:1–20:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Neuen and Schweitzer [18]; for graphs with bounded tree-width by Grohe, Neuen, Schweitzer, Wiebking [19] and Wiebking [36]; for bounded rank-width graphs by Grohe and Schweitzer [20] and graphs excluding small topological subgraphs [32].

In this paper, we study the isomorphism problem of graphs defined on finite groups. More precisely, we study the class of power graphs, directed power graphs, and enhanced power graphs. For two elements x and y in a finite group G, we say that y is a power of x if $y = x^i$ for some integer i. For a group G, the vertex set of the *power graph* $Pow(G)$ of G consists of elements of G. Two vertices x and y are adjacent in $Pow(G)$ if x is a power of y or y is a power of x. We refer to G as the *underlying group* of $Pow(G)$. The definition of directed power graphs and enhanced power graphs can be found in Section 2.

Kelarev and Quinn defined the concept of directed power graphs of semigroups [24]. Power graphs were defined by Chakrabarty et al. [11] again for semigroups. Cameron in [9] discusses several graph classes defined in terms of groups and surveys many interesting results on these graphs. Kumar et al. [25] gave a survey on the power graphs of finite groups. Questions related to isomorphism of graphs defined on groups have also been studied [34, 13, 30, 14].

Our motivation for studying the isomorphism of graphs defined in terms of groups is to explore if the group structure can be exploited to give efficient algorithms for the isomorphism problems of these graphs. There are two versions of the isomorphism problem for each class of graphs defined on groups. Let us consider the case for the class of power graphs. In the first version of the problem, two groups G_1 and G_2 are given by their Cayley tables, and the task is to check if $Pow(G_1)$ is isomorphic to $Pow(G_2)$. In the second version, two power graphs Γ_1 and Γ_2 are given and we need to check if Γ_1 is isomorphic to Γ_2. Note that in the second version, the underlying groups are not given as input.

In the first version of the problem, it is tempting to use the isomorphism of the underlying groups in the hope that it might yield an easier [1] quasipolynomial time algorithm because, unlike graphs, the quasipolynomial time algorithm for groups attributed to Tarjan by Miller [29] is technically and conceptually much easier. However, we note that it is not enough to check the isomorphism of the underlying groups, as two nonisomorphic groups can have isomorphic power graphs[2].

The second version looks more challenging since we do not have the underlying groups. As one of our *main results*, we show that the isomorphism problem of power graphs of nilpotent groups can be tested in polynomial-time even in the second version of the isomorphism problem (Section 6). Thus, we do not need the underlying groups to be given. In contrast, the group isomorphism problem for nilpotent groups, even for class 2, is still unresolved and is considered to be a bottleneck for the group isomorphism problem [15].

Our algorithm for solving the isomorphism problem of power graphs works by first computing the directed power graphs of the input power graphs. Next, we use the algorithm for the isomorphism problem of directed power graphs for nilpotent groups that we design in Section 6.

The question of efficiently computing the directed power graph from the power graph (or the enhanced power graph) was asked by Cameron [9]: “Question 2: Is there a simple algorithm for constructing the directed power graph or the enhanced power graph from the power graph, or the directed power graph from the enhanced power graph?” Recently,

[1] compared to Babai’s quasipolynomial time isomorphism algorithm.

[2] To see this, we can take the elementary abelian group of order 27 and the non-abelian group of order 27 with exponent 3 ([10]). In general, consider the power graphs of two nonisomorphic groups of order p^i for any $i \geq 2$ and exponent p for some prime p. One can check that the power graphs are isomorphic while the groups are not.

Bubboloni and Pinzauti [7] gave a polynomial-time algorithm to reconstruct the directed power graph from the power graph. We give a different solution to the problem of efficiently computing the directed power graph from the power graph. Moreover, we also design an efficient algorithm to compute the directed power graph from the enhanced power graph. This fully resolves Cameron's question.

Cameron [8], and Zahirović et al. [37] proved that for two finite groups, the power graphs are isomorphic if and only if the directed power graphs of the groups are isomorphic if and only if the enhanced power graphs of the groups are isomorphic. The algorithms to solve Cameron's question provide a complete algorithmic proof of this result.

This paper makes contributions in three algebraic and combinatorial techniques, which form the foundation of our algorithms. Firstly, we introduce a simple yet effective concept of a certain type of generating sets of a group which we call covering cycle generating sets (CCG-sets). In essence, these are defined in terms of the set of maximal cyclic subgroups (Section 3). Secondly, we present a set of results concerning the structure of closed-twins within specific subgraphs of power graphs. While Cameron previously explored the structure of closed-twins in a power graph [8], we extend this investigation to focus on the subgraph induced by the closed neighbourhood of a vertex (Section 4). These structures form the basis of our algorithm to determine whether a vertex in a power graph can be part of a CCG-set. Lastly, we introduce a series of reduction rules that facilitate the simplification of the structure of a directed power graph while preserving its isomorphism-invariant properties (Section 6).

Related work on Cameron's question. The algorithm to reconstruct the directed power graph from a power graph by Bubboloni and Pinzauti works by first considering the notion of *plain* and *compound* closed-twin classes in a power graph[3]. The other notion is that of *critical* closed-twin classes. Depending on whether a closed-twin class is critical or non-critical they design efficient tests to determine if the class is plain or compound. Once this is done, they put directions to the edges of the power graph to reconstruct the directed power graph. The details can be found in [7]. In contrast, our algorithm identifies a CCG-set in the power graph by using the properties and algorithms associated with the graph reductions that we give in this paper.

2 Preliminaries

For a simple graph X, the vertex set of X is denoted by $V(X)$, and the edge set of X is denoted by $E(X)$. For basic definitions and notations from graph theory, an interested reader can refer to any standard textbook (for example, [35]). A *subgraph* of X is a graph Y, where $V(Y) \subseteq V(X)$ and $E(Y) \subseteq E(X)$. The subgraph with the vertex set $S \subseteq V(X)$, and all such edges in $E(X)$ whose both endpoints are in S, is called the *induced subgraph* of X on S, and it is denoted by $X[S]$.

The set of vertices adjacent to a vertex u in an undirected graph X is called the open neighborhood of u in X and is denoted by $N_X(u)$. The cardinality of $N_X(u)$ is called the *degree* of u in X, denoted by $deg_X(u)$. The *closed neighborhood* of a vertex u in X is denoted by $N_X[u]$ and defined by $N_X[u] = N_X(u) \cup \{u\}$. Two vertices in X are called *closed-twins*[4] in X if their closed neighborhoods in X are the same.

[3] These notions are similar to the closed-twin classes that Cameron calls type (a) and type (b) (Proposition 5, [8]).

[4] In the previous related literature, both the terms *'false twins'* and *'closed twins'* are used. However, in this paper, we follow the terminology used by Cameron and other authors in their works on graphs defined on groups.

For a directed graph X (with no multiple edges), the *out-neighborhood* of a vertex u in X is the set $\{v \in V(X) \ : \ (u,v) \ \in E(X)\}$ and *out-degree* of u in X, denoted by $\textit{out-deg}_X(u)$, is the size of the out-neighborhood of u in X. Similarly, the *in-neighborhood* of a vertex u in X is the set $\{v \in V(X) \ : \ (v,u) \ \in E(X)\}$ and *in-degree* of u in X, denoted by $\textit{in-deg}_X(u)$, is the size of the in-neighborhood of u in X.[5] Two vertices in a directed graph X are called the *closed-twins* in X if their closed-out-neighborhoods in X are the same and also the closed-in-neighborhoods in X are the same. An edge of the form (u,u) in a directed graph is called a *self-loop.*

In any graph X, the *closed-twin-class* of a vertex u in X is the set of all closed-twins of u in X. In this paper, if the underlying graphs are colored, then by isomorphism we mean color preserving isomorphism only.

A graph is called *prime* with respect to strong product if it cannot be represented as a strong product of two non-trivial graphs.

▶ **Definition 1.** *Two graphs X and Y are called* isomorphic *if and only if there exists a bijection f from $V(X)$ to $V(Y)$ such that $\{u,v\} \in E(X)$ if and only if $\{f(u), f(v)\} \in E(Y)$. Moreover, if X and Y are vertex-colored, then an isomorphism f is called a* color preserving isomorphism *if for all $u \in V(X)$, the color of u and the color of $f(u)$ are the same.*

▶ **Definition 2** (see for example [22])**.** *Let X and Y be two directed graphs. The* strong product *$(X \boxtimes Y)$ of X and Y is the graph with the vertex set $V(X) \times V(Y)$, where there is an edge from (u,u') to a distinct vertex (v,v') in $X \boxtimes Y$ if and only if one of the following holds: (i) $u = v$ and there is an edge from u' to v' in Y. (ii) $u' = v'$ and there is an edge from u is to v in X. (iii) There is an edge from u to v in X and an edge from u' to v' in Y.*

▶ **Definition 3** (see e.g., [23])**.** *Vertex identification of a pair of vertices v_1 and v_2 of a graph is the operation that produces a graph in which the two vertices v_1 and v_2 are replaced with a new vertex v such that v is adjacent to the union of the vertices to which v_1 and v_2 were originally adjacent. In vertex identification, it doesn't matter whether v_1 and v_2 are connected by an edge or not.*

All the groups considered in this paper are finite. The basic definitions and properties from group theory can be found in any standard book (see, for example, [33]). A subset H of a group G is called a *subgroup* of G if H forms a group under the binary operation of G; it is denoted by $H \leq G$. The number of elements in G is called the *order* of the group and it is denoted by $|G|$. The *order* of an element g in G, denoted by $o(g)$, is the smallest positive integer m such that $g^m = e$, where e is the identity element. The set $\{g, g^2, g^3, \ldots, g^{m-1}, e\}$ is the set of all group elements that are *generated* by g, where $m = o(g)$. Moreover, this set forms a subgroup of G and is called the *cyclic subgroup* generated by g and denoted by $\langle g \rangle$. The number of generators of a cyclic subgroup $\langle g \rangle$ is $\phi(o(g))$, where ϕ is the Euler's totient function. A group G is called *cyclic* if $G = \langle g \rangle$, for some $g \in G$. In a finite cyclic group G, for any factor m of $|G|$, G has a unique subgroup of order m (known as the converse of Langrance's theorem).

A group G is called a *p-group* if the order of each element is some power of p, where p is a prime. For a prime p, if p^m is the highest power of p such that p^m divides $|G|$, then a subgroup $H \leq G$ with the property $|H| = p^m$ is called a *Sylow p-subgroup* of G. The *direct product* of two groups G and H, denoted by $G \times H$, is the group with elements (g,h) where

[5] When the graph is clear from the context, we drop the suffixes.

$g \in G$ and $h \in H$. The group operation of $G \times H$ is given by $(g_1, h_1)(g_2, h_2) = (g_1g_2, h_1h_2)$, where the co-ordinate wise operations are the group operations of G and H respectively. A finite group is called a *nilpotent group* if it is a direct product of its Sylow p-subgroups.

We now give the definitions of graphs defined on groups that we discuss in this paper (see [9]).

▶ **Definition 4.** *The* directed power graph *of a group G ($DPow(G)$), is a directed graph with vertex set G, and edge set $E = \{(x, y) : y = x^m$ for some integer m $\}$. The* power graph *of a group G, denoted by $Pow(G)$, is the undirected version $DPow(G)$.*

If (x, y) is an edge in $DPow(G)$, then $o(y)$ divides $o(x)$ whereas if $\{x, y\}$ is an edge in $Pow(G)$, then $o(x)|o(y)$ or $o(y)|o(x)$. Let $\mathcal{DP}ow$ denote the set $\{DPow(G)$: G is a finite group $\}$. Let $\mathcal{P}ow$ denote the set $\{Pow(G)$: G is a finite group $\}$.

▶ **Definition 5.** *The enhanced power graph of a group G, denoted $EPow(G)$, is an undirected graph with vertex set G, in which two vertices x and y are adjacent if and only if they are in a common cyclic subgroup of G, i.e., there exists z in G such that $x, y \in \langle z \rangle$.*

Let $\mathcal{EP}ow$ denote the set $\{EPow(G)$: G is a finite group $\}$.

3 Cyclic cover of a group and organization of the paper

In this section, we introduce the notion of minimal cyclic cover and covering cycle generating set. We start with the following definitions.

▶ **Definition 6.** *We say that a proper cyclic subgroup C of G is a* maximal cyclic subgroup *if for all cyclic subgroups C', $C \leq C'$ implies $C = C'$ or $C' = G$.*

▶ **Definition 7.** *Let G be a finite group. Let $\{C_1, \ldots, C_m\}$ be a set of the cyclic subgroups of G. We say that $\{C_1, \ldots, C_m\}$ is a* minimal cyclic cover (MCC) *if $G = \cup_{i=1}^m C_i$ and $\cup_{i \neq j} C_i \neq G$ for all $j = 1, \ldots, m$.*

It is not hard to see from the following lemma that the set of all maximal cyclic subgroups forms the minimum cyclic cover of a non-cyclic group.

▶ **Lemma 8.** *For a cyclic group G the only MCC is $\{G\}$. For non-cyclic groups the set of all maximal cyclic subgroups forms the unique minimal cyclic cover.*

▶ **Definition 9.** *Let $\{C_1, \ldots, C_m\}$ be the minimum cyclic cover (MCC) of G. Each C_i is called a* covering cycle. *For a cyclic group C, let $gen(C)$ be the set of generators of C. An element in $\cup_{i=1}^m gen(C_i)$ is called a covering cycle generator or CC-generator. We call a set $\{g_1, g_2, \ldots, g_m\}$ a* covering cycle generating set *(CCG-set) if $\{\langle g_1 \rangle, \langle g_2 \rangle, \ldots, \langle g_m \rangle\} = \{C_1, C_2, \ldots, C_m\}$.*

The above definition includes the case when $m = 1$, i.e., G is cyclic.

Organization of the paper. With the notion of CCG-set defined above, we are now ready to describe the organization of the paper. For the sake of clarity we give the organization in a somewhat nonlinear manner.

How to identify a CCG-set in a power graph or in an enhanced power graph when the underlying group is not given? We design algorithms in Section 5 to solve this problem. These are iterative algorithms that take one of the potential vertices and decide if that vertex can be safely marked as a member of the CCG-set.

The correctness of the algorithm, in the case of power graphs, crucially depends on the structure of closed-twins in the subgraph induced by the closed neighbourhood of the potential vertex that the algorithm examines in each iterative step. In Section 4, we derive a collection of results that characterizes these structures.

In Section 6, we define a set of reduction rules that simplifies the structure of the directed power graph of a group G while retaining all its isomorphism-invariant properties. There are four reductions, Reduction 1, 2, 3, and 4, and they are applied one after the other in that order. The graph obtained after i-th reduction is denoted by $R_i(G)$. We also show that these reductions can be efficiently reversed. At the end of Section 6 we design an isomorphism algorithm for the directed power graphs of nilpotent groups using the structure of R_3.

In Section 7, we show how to obtain the reduced graph R_4 from an input power graph (or an enhanced power) graph given along with a CCG-set.

Combining the above, we have an efficient way of going from a power graph (or an enhanced power graph) to R_4 to R_3 to the directed power graph. This answers Cameron's question positively.

The isomorphism of the power graphs (or the enhanced power graphs) of nilpotent groups can be done as follows: compute the directed power graphs from the input graphs and apply the algorithm developed in Section 6.

4 Structure of closed-twins in a power graph

The structure of closed-twins in a power graph has been studied by Cameron [8], and by Bubboloni and Pinzauti [7]. In this section, we explore the structures of closed-twins in the subgraph of a power graph induced by the closed neighborhood of a vertex. We show in Section 5.1 that these structures can be used to find a CCG-set of a group from the corresponding power graph, even when the group is not given.

First, we note an easy fact about the closed-twins in any graph.

▶ **Lemma 10.** *Let X be a graph and let $v \in V(X)$. Suppose x and y are closed-twins in X. If $x \in N[v]$, then $y \in N[v]$ and x and y are closed-twins in $X[N[v]]$.*

In a group G, an element $x \in G$ and any generator of $\langle x \rangle$ are closed-twins in $\Gamma = Pow(G)$. Therefore, by Lemma 10, we have the following corollary.

▶ **Corollary 11.** *Let $v \in V(\Gamma)$. If $x \in N[v]$, then all the generators of $\langle x \rangle$ are in $N[v]$. Moreover, they are closed-twins in Γ_v, where $\Gamma_v = \Gamma[N[v]]$.*

Now consider a vertex $v \in V(\Gamma)$, where $\Gamma \in \mathcal{P}ow$ and the subgraph $\Gamma_v = \Gamma[N[v]]$ induced on the closed neighborhood of v. For any vertex x in Γ_v, $o(x)|o(v)$ or $o(v)|o(x)$. We partition $V(\Gamma_v)$ according to the order of the vertices in the following way:

$$U_v = \{x \in V(\Gamma_v) : o(x) > o(v)\},\ E_v = \{x \in V(\Gamma_v) : o(x) = o(v)\},$$
$$L_v = \{x \in V(\Gamma_v) : o(x) < o(v)\}$$

For a vertex $x \in U_v$, we have $o(v)|o(x)$ and for a vertex $x \in L_v$, we have $o(x)|o(v)$.

▶ **Definition 12.** *For a prime p, an element x in a group is called* a p-power element *if $o(x) = p^i$ for some $i \geq 0$, x is a* nontrivial p-power element *if $i > 0$.*

▶ **Lemma 13.** *Suppose $v \in V(\Gamma)$ is not a p-power element for any prime p and $x \in U_v$ is a closed-twin of v in Γ_v. Then, $deg_\Gamma(x) > deg_\Gamma(v)$.*

Proof. In this case, there exists prime q and positive integer s such that $q^s|o(x)$ but $q^s \nmid o(v)$. Then, x has a neighbor $z = x^{\frac{o(x)}{q^s}}$ of order q^s (by the converse of Lagrange's theorem in finite cyclic groups). Note that z is not a neighbor of v as $o(z) \nmid o(v)$ and also $o(v) \nmid o(z)$. The latter is true as $o(v)$ is divisible by at least two distinct primes. ◀

The proofs of the next lemma is in Section A.1.

▶ **Lemma 14.** *Let $v \in V(\Gamma)$ be a CC-generator such that $o(v)$ is not a prime power. Let $u \in V(\Gamma_v)$. If $u = e$ or u is a generator of $\langle v\rangle$, then the closed-twins of u in Γ_v are exactly the generators of $\langle v\rangle$ and e; otherwise, the closed-twins of u in Γ_v are exactly the generators of $\langle u\rangle$.*

▶ Remark 15. If $a|b$, $a \neq b$, then (1) $\phi(a)|\phi(b)$, (2) $\phi(a) \leq \phi(b)$ and the equality holds only when $b = 2a$ where a is an odd number.

If $v \in V(\Gamma)$ is a CC-generator, it is easy to see that $o(v) = deg(v) + 1 = |\Gamma_v|$. Let $o(v)$ be not a prime power. Then, using Lemma 14, the set of dominating vertices in Γ_v is the set of generators of $\langle v\rangle$ and identity. Thus, the size of the closed-twin-class of v in Γ_v is $\phi(o(v)) + 1$, i.e., $\phi(|\Gamma_v|) + 1$. Also, for all divisors, $1 < k < o(v)$, of $o(v)$, there exists a closed-twin-class of size $\phi(o(k))$ in Γ_v. The proof of the following corollary can be found in the full version of the paper [12].

▶ **Corollary 16.** *Let $v \in V(\Gamma)$ be a CC-generator and $o(v)$ be not a prime power. Then the following holds: (1) The size of the closed-twin-class of v in Γ_v, i.e., the set of dominating vertices in Γ_v, is $\phi(o(v)) + 1$. (2) For each divisor k of $o(v)$, $1 < k < o(v)$, there is a closed-twin-class of size $\phi(k)$ in Γ_v. Moreover, $\phi(k)$ divides $\phi(o(v))$. (3) There are at most two closed-twin-classes of size greater or equal to $\phi(o(v))$.*

The following theorem is a well-known result [11].

▶ **Theorem 17** ([11]). *Let G be a finite group. Then, $\Gamma = Pow(G)$ is complete if and only if G is cyclic of prime power order.*

From the above theorem, the following corollary is immediate.

▶ **Corollary 18.** *Let $v \in V(\Gamma)$ be a p-power element for some prime p. Then, $\Gamma[E_v \cup L_v]$ is a complete graph. Moreover, the elements of $E_v \cup L_v$ are closed-twins of v in Γ_v.*

▶ **Lemma 19.** *Let $v \in V(\Gamma)$ be a nontrivial p-power element and not a CC-generator. Suppose for all $u \in U_v$ such that u is a closed-twin of v in Γ_v, $deg_\Gamma(u)$ is at most $deg_\Gamma(v)$. Let y be a closed-twin of v in Γ_v with maximum order and S denote the set $\{x \in V(\Gamma_v) : o(y)|o(x)$ and $o(x) \neq o(y)\}$. Then, (1) The closed-twins of v are exactly the elements in $\langle y\rangle$. (2) $V(\Gamma_v) = \langle y\rangle \sqcup S$, where $\sqcup$ denotes the disjoint union.*

(3) Moreover, if $o(y) = p^j$ where $j \geq 2$, then p divides $|V(\Gamma_v)|$.

Proof. Suppose $o(v) = p^i$. From Corollary 18, we know that the elements in E_v and L_v are closed-twins of v. Observe that these elements have order p^r for some $r \leq i$. Next, we show that all closed-twins of v in U_v have orders of the form p^l for some $l > i$. Suppose not, then let u be a closed-twin of v in U_v. As $u \in U_v$, p^i divides $o(u)$. Then $o(u) = k \cdot p^i$, where $k > 1$ and $gcd(k, p) = 1$. Since u is a closed-twin of v, $|\Gamma_v| - 1 = deg_{\Gamma_v}(u) = deg_{\Gamma_v}(v) = deg_\Gamma(v)$. Now, $\langle u\rangle$ has an element of order k and this element cannot be a neighbor of v. So, $deg_\Gamma(u) > deg_{\Gamma_v}(u) = deg_\Gamma(v)$. Therefore, $deg_\Gamma(u) > deg_\Gamma(v)$. This is a contradiction. Hence, $o(u) = p^l$, for some $l > i$.

Given that y is the closed-twin of v in Γ_v with maximum order, say p^j. Suppose $z \in \langle y \rangle$. If $y \in E_v \cup L_v$, then clearly $\langle y \rangle = \langle v \rangle$ (because y cannot be in L_v). If $y \in U_v$, then noting that $deg_\Gamma(y) \leq deg_\Gamma(v)$ and y is a closed-twin of v in Γ_v, we can say that z is in Γ_v. In both the cases $\langle y \rangle \subseteq V(\Gamma_v)$. We show that every vertex $w \in V(\Gamma_v)$ is adjacent to z. Since $w \in V(\Gamma_v)$ and y is a closed-twin of v, there is an edge between w and y. So, $o(y)|o(w)$ or $o(w)|o(y)$. In the first case, $z \in \langle y \rangle \subseteq \langle w \rangle$. On the other hand, if $o(w)|o(y)$, then $w \in \langle y \rangle$. So either $z \in \langle w \rangle$ or $w \in \langle z \rangle$ as $\langle y \rangle$ is a cyclic group of prime power order. In any case, $\{w, z\}$ is an edge. So, any element z in $\langle y \rangle$ is a closed-twin of v.

Let z be a closed-twin of v. If $z \in \langle v \rangle$ then $z \in \langle y \rangle$. On the other hand, if $z \notin \langle v \rangle$, then $z \in U_v$. Therefore, $o(z)$ is a power of p. As y is a closed-twin of v, there is an edge between y and z. Therefore, $z \in \langle y \rangle$ or $y \in \langle z \rangle$. If $y \in \langle z \rangle$, we must have $\langle y \rangle = \langle z \rangle$ as $o(z) \leq o(y)$ (as both are p-power order closed-twins of v and y is with maximum order). This forces $\langle z \rangle = \langle y \rangle$. Thus, $z \in \langle y \rangle$ in both cases. This completes the proof of part (1).

Now, we prove part (2). Let $x \in V(\Gamma_v) \setminus \langle y \rangle$. In this case, $x \notin E_v \cup L_v$ by Corollary 18. Since y is a closed-twin of v, $\{x, y\}$ is an edge. Therefore, $x \in \langle y \rangle$ or $y \in \langle x \rangle$. However, by assumption, $x \notin \langle y \rangle$. So, $y \in \langle x \rangle$. Therefore, $o(y)|o(x)$. So, $o(x) = p^j \cdot k$ for some $k > 1$.

Therefore, $V(\Gamma_v) = \langle y \rangle \sqcup \ \{x \in V(\Gamma_v) : p^j|o(x) \text{ and } o(x) \neq p^j\}$, i.e., $V(\Gamma_v) = \langle y \rangle \sqcup S$.

To prove part (3), we define an equivalence relation $\equiv$ on S as follows: $x_1 \equiv x_2$, if and only if $\langle x_1 \rangle = \langle x_2 \rangle$. Note that if $x \in S$, then generators of $\langle x \rangle$ are in S by Corollary 11. Therefore, the equivalence class of any vertex $x \in S$ is of size $\phi(o(x))$. Recall that, $o(x) = o(y) \cdot k = p^j \cdot k$. Now, as $p^j \geq p^2$, so p divides $\phi(o(x))$. Therefore, p divides $|V(\Gamma_v)|$, as claimed. ◀

5 Finding a CCG-set of a group from its power graph and enhanced power graph

For a directed power graph, even if the underlying group is not given, the set of vertices corresponding to a CCG-set $\{g_1, \ldots, g_m\}$ of the underlying group G can be readily found in the graph. The scenario changes when the input graph is a power graph or an enhanced power graph and the underlying group is *not* given as input. Then, it is not possible to recognise these vertices exactly in the input graph as we can not distinguish two closed-twins g_i and g'_i in $Pow(G)$ (or $EPow(G)$). For example, if we take $\mathbb{Z}_{p^m}$ for some prime p and integer m, then $Pow(\mathbb{Z}_{p^m})$ is a clique (Theorem 17). If the vertices of $Pow(\mathbb{Z}_{p^m})$ are named arbitrarily, then it is not possible to distinguish a generator of $\mathbb{Z}_{p^m}$ from any other vertex. Fortunately, the fact that the underlying group is $\mathbb{Z}_{p^m}$ can be concluded just from the graph by Theorem 17.

Therefore, we aim to do the following: Given a power graph (or an enhanced power graph) Γ, mark a set $\{g_1, g_2, \ldots, g_m\}$ of vertices such that (1) each g_i is a CC-generator or g_i is a closed-twin of a CC-generator g'_i in the graph Γ, and (2) $\{h_1, h_2, \ldots, h_m\}$ is a CCG-set where $h_i = g_i$, if g_i is a CC-generator; otherwise, $h_i = g'_i$.

5.1 Finding a CCG-set of a group from its power graph

The next theorem states that given a power graph Γ as input, we can essentially compute a CCG-set corresponding to Γ, even if the underlying group is not given.

▶ **Theorem 20.** *There is a polynomial-time algorithm that, on input a power graph $\Gamma \in \mathcal{P}ow$, outputs a set $\{g_1, g_2, \ldots, g_m\}$ where g_i is a CC-generator or g_i is a closed-twin of a CC-generator g'_i in the graph Γ such that $\{h_1, h_2, \ldots, h_m\}$ is a CCG-set where $h_i = g_i$, if g_i is a CC-generator, otherwise, $h_i = g'_i$.*

Hence, without loss of generality, we call the set $\{g_1, g_2, \ldots, g_m\}$ as CCG-set and g_i's as CC-generators. Before we give the proof of the above theorem, we need the following definition that is required in the algorithm.

▶ **Definition 21.** *Let d be a positive integer. Let $\Gamma \in \mathcal{P}ow$ and v be a vertex in Γ. Let $T_1, \ldots, T_r$ be the partition of $V(\Gamma_v)$ into closed-twin classes of Γ_v. Similarly, let $S_1, \ldots, S_{r'}$ be the closed-twin classes of $Pow(\mathbb{Z}_d)$. We say that Γ_v* closed-twin-partition-wise matches *with $Pow(\mathbb{Z}_d)$ if (1) the closed-twin class containing the dominating vertices[6] of both the graphs have the same size, and (2) $r = r'$ and there is some permutation $\pi \in Sym(r)$ such that $|T_i| = |S_{\pi(i)}|$.*

If v is a CC-generator and $o(v) = d$ is not a prime power then, Γ_v twin-partition-wise-matches with $Pow(\mathbb{Z}_d)$, by Corollary 16.

It is not hard to see that testing if Γ_v closed-twin-partition-wise matches with $Pow(\mathbb{Z}_d)$ can be done in polynomial-time. Also, when d is not prime power, the size of the closed-twin class containing v has size $\phi(d)$+1.

Proof of Theorem 20. The process of finding a CCG-set of the underlying group of a given power graph is described in Algorithm 1.

Algorithm 1 Algorithm to mark a CCG-set in a finite power graph.

Input: $\Gamma \in \mathcal{P}ow$

- First, isolate the case when the power graph Γ is a complete graph using Theorem 17. Then, return a singleton set, consisting of any vertex, as the CCG-set.
- If Γ is not a clique, then mark any of the Dominating vertices as the identity. Next, all vertices except the identity are stored in a list L in decreasing order of their degrees.
- During the algorithm, we use the labels: U (undecided), CC (a CC-generator) and NC (not a CC-generator). To start with, mark all the vertices U in the list. Note that identity is not marked with any label.
- The algorithm marks the vertices further in phases. In each phase, pick the first U marked vertex, say v, in the list L and do the following: Let $a = deg(v) + 1$

  [Rule 1a] *If* a is a prime power and $\Gamma_v = \Gamma[N[v]]$ is complete, then mark v as CC and mark all its neighbors NC.

  [Rule 1b] *Else if* a is a prime power and Γ_v is not complete, then mark v as NC.

  [Rule 2a] *Else if* a is not a prime power and if v has a closed-twin w in Γ_v such that w has been marked NC, then mark v as NC.

  [Rule 2b] *Else* (i.e., a is not a prime power and v does not have a NC marked closed-twin in Γ_v)

  ```
  If Γ_v closed-twin-partition-wise matches with Pow(Z_d), where d = deg(v) + 1
      Mark v as CC and all its neighbors NC.
  Else
      Mark v as NC.
  ```
- Return the set of vertices marked CC.

A vertex picked at any phase is either marked CC or NC, thereby reducing the number of vertices marked U. Therefore, Algorithm 1 terminates in $O(n)$ phases.

[6] *Dominating vertex* in a graph is a vertex that is adjacent to all other vertices in the graph.

We prove the correctness of the algorithm by induction on the number of phases. In each phase, a set of vertices is relabeled using one of the 4 rules. We prove that this labelling is correct. In phase i, we assume that up to phase $(i-1)$, all the labellings were done correctly. In base case, this means that all the vertices are still labelled U.

If Rule 1a is applied: If v is not a CC-generator, then v is contained in at least one covering cycle. If v is contained in two covering cycles, say $\langle g_1 \rangle$ and $\langle g_2 \rangle$, then Γ_v is not complete, as the CC-generators g_1 and g_2 are not adjacent. Now consider the case when v is contained in exactly one covering cycle $\langle x \rangle$. Then $N_{\Gamma_v}(v) \subseteq N_{\Gamma_v}(x)$. So, if $deg_{\Gamma}(x) > deg_{\Gamma}(v)$, then x or one of its closed-twins has already been marked as CC in some previous phase, and then v would have been marked as NC. Now if $deg_{\Gamma}(x) = deg_{\Gamma}(v)$, then v and x are closed-twins and thus v is also a CC-generator. This is a contradiction.

If Rule 1b is applied: If v is a CC-generator, then Γ_v is a complete graph. Thus, this step works correctly.

If Rule 2a is applied: If v is a CC-generator, then by Lemma 14 its closed-twins in Γ_v are exactly e (identity) and generators of $\langle v \rangle$. So, if any of the closed-twins is marked NC, it must have been because some other closed-twin is already marked CC in some previous phase $t \leq i-1$ of the algorithm. In phase t, the algorithm would have also marked v as NC.

If Rule 2b is applied: If v is a CC-generator, then Γ_v closed-twin-partition-wise matches with $Pow(\mathbb{Z}_d)$. Now if none of v's closed-twins in Γ_v are already marked CC, then v can be marked CC.

On the other hand, suppose that v is not a CC-generator. We first consider the case when v is contained in only one covering cycle, say generated by x. The proof of the next claim is in Section A.1.

▷ Claim 22. $deg_{\Gamma}(x) > deg_{\Gamma}(v)$.

By the above claim, the algorithm considers x and other generators of $\langle x \rangle$ before v. Then, by the induction hypothesis, one of these generators would be marked CC, and v would not be labelled U.

Now we consider the case when v is contained in at least two covering cycles, say $\langle g_1 \rangle$ and $\langle g_2 \rangle$. We prove that if v is not a CC-generator, then Γ_v cannot closed-twin-partition-wise match with $Pow(\mathbb{Z}_d)$. This case is divided into two subcases.

In the 1^{st} subcase, we assume that $o(v)$ is not a prime power. Now we count the closed-twins of v in Γ_v present in each of the sets U_v, E_v and L_v.

If $x \in U_v$ is a closed-twin of v in Γ_v, then by Lemma 13, $deg_{\Gamma}(x) > deg_{\Gamma}(v)$. So, the algorithm must have considered x before v. At that phase, the algorithm either marked x as NC or CC. If x was marked as NC, v would not satisfy the condition of Rule 2b (i.e., no closed-twin of v in Γ_v is marked NC). Moreover, if x was marked CC, the algorithm would have marked v as NC. So, v has no closed-twin in U_v.

The number of closed-twins of v in Γ_v which are present in E_v is $\phi(o(v))$. By noting that $\Gamma_v[E_v \sqcup L_v] = Pow(\langle v \rangle)$ and using Lemma 14 on $Pow(\langle v \rangle)$, we see that the only closed-twin of v in L_v is the identity. Therefore, the total number of closed-twins of v in Γ_v is $\phi(o(v)) + 1$.

Now CC-generators g_1 and g_2 have distinct [7] closed-twin-classes of size at least $\phi(o(g_1))$ and $\phi(o(g_2))$. But, $\phi(o(g_i)) \geq \phi(o(v))$ for $i = 1, 2$ by Remark 15. This is a contradiction since $Pow(\mathbb{Z}_d)$ can have at most two closed-twin-classes of size greater than or equal to $\phi(o(v))$, by (3) of Corollary 16.

[7] g_1 and g_2 are not adjacent.

In the 2^{nd} subcase, we assume that $o(v)$ is a prime power, say $o(v) = p^i$ for some prime p and some integer $i > 0$. Consider y and S as in Lemma 19. Note that $deg_\Gamma(y) \leq deg_\Gamma(v)$. Since otherwise, the algorithm would have marked y as NC or CC. In both cases, the algorithm would not satisfy the conditions of Rule 2b.

Subsubcase 1: $o(y) = p^j, j \geq 2$. In this case, by (3) of Lemma 19, p divides $|V(\Gamma_v)|$. Therefore, Γ_v must have a closed-twin-class of size $p - 1$ for it to closed-twin-partition-wise match with $Pow(\mathbb{Z}_d)$ (because of (2) of Corollary 16). By (1) of Lemma 19, if $x \in \langle y \rangle$, then the number of closed-twins of x in Γ_v is $|\langle y \rangle| = p^j > p - 1$. Also, if $x \in S$, then number of closed-twins of x in $\Gamma_v \geq \phi(o(x)) \geq \phi(o(y)) \geq \phi(p^2) > p - 1$. Therefore, there is no closed-twin-class of size $p - 1$.

Subsubcase 2: $o(y) = p$. Recall that $\langle g_1 \rangle$ and $\langle g_2 \rangle$ are covering cycles containing v. Since y and v are closed-twins in Γ_v, we can see that $y \in \langle g_1 \rangle \cap \langle g_2 \rangle$. Now by (1) of Lemma 19, the size of the closed-twin-class of v is p. Since $o(y)\big|o(g_1)$ and $o(y)\big|o(g_2)$, the size of the closed-twin-class of both g_1 and g_2 is at least $p - 1$. This is not possible by (3) of Corollary 16. ◀

5.2 Finding a CCG-set of a group from its enhanced power graph

▶ **Lemma 23.** *If v is a CC-generator of a group G, then $N_{EPow(G)}[v] \subseteq N_{EPow(G)}[u]$ for all $u \in \langle v \rangle$.*

Algorithm 2 performs the task of finding a CCG-set. The next theorem ensures the correctness of the algorithm.

▶ **Theorem 24.** *Algorithm 2 on input an enhanced power graph[8] $\Gamma \in \mathcal{EP}ow$ outputs a set $\{g_1, g_2, \ldots, g_m\}$ where g_i is a CC-generator or g_i is a closed-twin of a CC-generator g_i' in the graph Γ such that $\{h_1, h_2, \ldots, h_m\}$ is a CCG-set where $h_i = g_i$ if g_i is a CC-generator, otherwise, $h_i = g_i'$.*

As before, we call the set $\{g_1, g_2, \ldots, g_m\}$ as CCG-set and g_i's as CC-generators.

Proof. The proof of correctness of Algorithm 2 is by induction on the number of iterations. In any iteration, the first unmarked vertex is marked as CC and its neighbors in the graph are marked as NC. Our goal is to prove that this marking process is correct.

For the base case, $x = v_1$. By Lemma 23, v_1 is either a CC-generator or $v_1 \in \langle g_1 \rangle$, where g_1 is a CC-generator and v_1 is a closed-twin of g_1 in Γ. Since $N[v_1]$ corresponds to $\langle g_1 \rangle$ by Lemma 23, we can safely mark the vertices adjacent to v_1 as NC.

In phase i, we assume that up to iteration $(i-1)$, all the markings were done correctly. Let us pick the first unmarked vertex, say x, in A. It is easy to see that x does not belong to any covering cycle marked till the $(i-1)^{th}$ iteration, i.e., x does not belong to the neighborhood of any CC marked vertex till the $(i-1)^{th}$ iteration. So, again using the same argument given in the base case, it can be seen that the markings done in the i^{th} iteration are correct. ◀

[8] Recall that the underlying group is not given.

Algorithm 2 Algorithm to mark a CCG-set of G in a finite enhanced power graph.

```
Input: $\Gamma \in \mathcal{EP}ow$
1. Sort the vertices of $\Gamma$ by their degree in increasing order. Let the sorted array be
   $A = \{v_1, v_2, \ldots, v_m\}$.
2. Pick the first unmarked element $x$ of $A$ and mark it CC.
   Mark all the elements of $N[x]$ as NC.
3. Pick the next unmarked element in $A$ and repeat Step 2 till all elements of $A$ are marked.
```

6 Isomorphism of directed power graphs

The isomorphism problems of power graphs, directed power graphs, and enhanced power graphs are equivalent (see [9, 8, 37]). Thus, an algorithm for the isomorphism problem of directed power graphs automatically gives an isomorphism algorithm for power graphs (or enhanced power graphs), provided we can obtain the directed power graph from the power graph (respectively, the enhanced power graph). This is done in Section 7. In the currect section, we focus on the isomorphism problem of directed power graphs. In the last part of this section, we discuss a necessary result that is used in Section 7 for obtaining the directed power graph of an input power graph (or an enhanced power graph).

We perform several reductions on a directed power graph that are isomorphism invariant. The out-degree of a vertex in $DPow(G)$ is the order of the element in the group G, i.e., for a vertex u, $out\text{-}deg(u) = o(u)$. Therefore we can color the vertices by their out-degrees. We call the colored graph $CDPow(G)$. We emphasise that here the colors are numbers, and hence we can perform arithmetic operations on these colors and use the natural ordering of integers inherited by these colors. We recall that by isomorphism we mean color preserving isomorphism when the graphs are colored.

Two vertices u and v are closed-twins in $CDPow(G)$ (in $DPow(G)$ also) if and only if $\langle u\rangle = \langle v\rangle$ in G, i.e., u and v are two generators of the same cyclic subgroup in G. There are $\phi(o(u))$ generators of $\langle u\rangle$ in G. So, for each vertex $u \in CDPow(G)$, there are exactly $\phi(col(u))$ closed-twins in $CDPow(G)$. By the converse of Lagrange's theorem, in each cyclic subgroup of order n, for each divisor k of n, there are exactly $\phi(k)$ generators. So, for each k in the color set of $CDPow(G)$, there are $\phi(k)$ closed-twins in the graph. Observe that u and v are closed-twins in $CDPow(G)$, if and only if $(u, v) \in E(CDPow(G))$ and $col(u) = col(v)$.

Reduction rule 1: Closed-twin Reduction. If there are two closed-twins u and v in $CDPow(G)$, then do a vertex identification of u and v and color the identified vertex with $col(u) = col(v)$. Let $R_1(G)$ denote the reduced graph after applying Reduction rule 1 to $CDPow(G)$.

From the discussion above, the next lemma follows easily.

▶ **Lemma 25.** *$CDPow(G) \cong CDPow(H)$ if and only if $R_1(G) \cong R_1(H)$.*

▶ Remark 26. It is easy to see that we can get back an isomorphic copy of $CDPow(G)$ from $R_1(G)$, by adding $\phi(col(u))$ closed-twins at each vertex u in $R_1(G)$.

Since each vertex has a self-loop, for the purpose of isomorphism we can delete these self-loops. One can check that $R_1(G)$ is a transitively closed directed graph.

Reduction rule 2: Edge-deletion. Let us consider $R_1(G)$. Do the following steps: (1) Delete all self-loops. (2) For all a, b, c, if (a, b) and (b, c) are edges, then mark (a, c) as a transitive edge. Then, delete all edges that are marked as transitive edges. Let $R_2(G)$ denote the resulting graph. Since $R_1(G)$ is the reflexive and transitive closure of $R_2(G)$, we have the following lemma:

▶ **Lemma 27.** *$R_1(G) \cong R_1(H)$ if and only if $R_2(G) \cong R_2(H)$.*

Due to space constraints we omit the proof of the following lemma (see the full version for a proof [12]).

▶ **Lemma 28.** *The reduced graph $R_2(G)$ satisfies the following properties: (1) Vertices with in-degree zero in $R_2(G)$ form a CCG-set of G, (2) If (u, v) is an edge in $R_2(G)$, then $col(u) > col(v)$ and $col(u) = col(v) \cdot p$ for some prime p, (3) $R_2(G)$ is a directed acyclic graph.*

Note that using (1) of Lemma 28, we can easily find a set of vertices, say $\{g_1, g_2, \ldots, g_m\}$, that form a covering cycle generating set (CCG-set) of G.

Reduction rule 3: Removing the direction. Remove the direction of the edges in $R_2(G)$ to obtain an undirected colored graph $R_3(G)$.

Note that the CCG-set of G can still be identified easily in $R_3(G)$: A vertex g is a CC-generator if and only if all its neighbours have smaller orders (or colors).

The following result is an easy consequence of (2) of Lemma 28.

▶ **Lemma 29.** *$R_2(G) \cong R_2(H)$ if and only if $R_3(G) \cong R_3(H)$.*

▶ **Definition 30.** *A path $u_1u_2 \ldots u_l$ in $R_3(G)$ is said to be a* descendant path *if $col(u_i) > col(u_{i+1})$. The vertices in the graph reachable from u using descendant path are called descendant reachable vertices from u. We denote the set of descendant reachable vertices from u in $R_3(G)$ by $Des(u)$.*

Observe that $Des(u)$ in $R_3(G)$ is same as the closed out-neighborhood of u in $R_1(G)$. The colors of the vertices of $Des(u)$ in $R_3(G)$ form the set of all divisors of $col(u)$. Also, no two vertices of $Des(u)$ in R_3 have the same color.

▶ **Theorem 31.** *If G is a finite p-group, then $R_3(G)$ is a colored tree.*

Proof. Let $|G| = p^\alpha$. Any edge in $R_3(G)$ is of the form $\{u, v\}$ where by using (2) of Lemma 28 we can assume without loss of generality that $col(u) = p^t$ and $col(v) = p^{t-1}$, for some $t \in \{1, 2, \ldots, \alpha\}$. Suppose the graph contains a cycle $u_0u_1u_2 \ldots u_nu_0$. By (2) of Lemma 28, we can assume without loss of generality that the colors of the vertices form the following sequence: $p^tp^{t-1}p^{t-2} \ldots p^{t-(i-1)}p^{t-i}p^{t-(i-1)} \ldots p^{t-1}p^t$ for some i. Now, $u_1, u_n \in Des(u_0)$ such that $col(u_1) = col(u_n) = p^{t-1}$. This is a contradiction since no two vertices of $Des(u)$ for any vertex u have the same color. Hence, our assumption is wrong and $R_3(G)$ has no cycle. ◀

Since the isomorphism of trees can be tested in linear time (see, for example, [1]), the isomorphism of the directed power graphs of p-groups can also be tested in polynomial-time.[9] Now we extend our algorithm to check the isomorphism of directed power graphs of finite nilpotent groups. For that, we use the following two results.

[9] It can actually be done in linear time.

▶ **Lemma 32** ([31]). *Let G_1 and G_2 be two finite groups such that $|G_1|$ and $|G_2|$ are co-prime to each other. Then, $DPow(G_1 \times G_2) = DPow(G_1) \boxtimes DPow(G_2)$, where $\boxtimes$ denotes the strong product of two graphs.*

▶ **Lemma 33** ([27]). *There exists a unique prime factor decomposition of a simple connected* [10] *directed graph with respect to strong product and the uniqueness is up to isomorphism and ordering of the factors.*

It is easy to verify that the next lemma follows from the above two lemmas. However, we can prove Lemma 34 without using Lemma 33 and the proof is given in Section A.1.

▶ **Lemma 34.** *Let $G = G_1 \times \cdots \times G_k$ and $H = H_1 \times \cdots \times H_k$ where $|G_i| = |H_i|$ for all $1 \leq i \leq k$. Suppose $gcd(|G_i|, |G_j|) = gcd(|H_i|, |H_j|) = 1$, for all $1 \leq i < j \leq k$. Then, $DPow(G) \cong DPow(H)$ if and only if $DPow(G_i) \cong DPow(H_i)$, for all $1 \leq i \leq k$.*

We are now ready to present one of the main results of the paper. Namely, we show that the isomorphism of the directed power graphs of nilpotent groups can be tested in polynomial-time. Let $\mathcal{DP}ow_{nil} = \{DPow(G) \ : \ G \text{ is a finite nilpotent group}\}$.

Theorem 31 and Lemma 34 suggest that obtaining the directed power graphs corresponding to the factor groups might be useful. One approach would be to decompose an input directed power graph into prime factors with respect to strong product in polynomial-time using the algorithm by Hellmuth et al. [21]. Note that, in a general setting, the prime graphs in the strong product decomposition may not correspond to the directed power graphs of the direct factors of the underlying group. We are also not sure if the Sylow-p subgroups of a nilpotent group generate prime graphs. If not, then just applying the algorithm of Hellmuth et al. is not enough and we need to regroup the prime factors properly to apply Theorem 31. Fortunately, all these complications can be easily avoided as shown in the next theorem.

▶ **Theorem 35.** *There is an efficient polynomial-time algorithm that on inputs $\Gamma_1, \Gamma_2 \in \mathcal{DP}ow_{nil}$ checks if Γ_1 and Γ_2 are isomorphic.*

Proof. We know that a finite nilpotent group is the direct product of its Sylow subgroups. Since the orders of the Sylow subgroups are coprime with each other, by Lemma 34, Γ_1 and Γ_2 are isomorphic if and only if for each prime p dividing $|V(\Gamma_1)|$ (which is same as the order of the underlying group), the directed power graphs of the Sylow p-subgroups of the underlying groups of Γ_1 and Γ_2 are isomorphic. Therefore, if we can find the directed power graphs of the Sylow subgroups associated with each prime divisor, we can test the isomorphism of Γ_1 and Γ_2.

While the underlying groups are not given as input, we can still compute the directed power graph of a Sylow p-subgroup of an input graph by finding the set V_p of all vertices whose order in the underlying group is p^i for some $i \geq 0$. More precisely, the subgraph induced by the set V_p is the directed power graph associated with the Sylow p-subgroup. Note that the order of a vertex (which is also an element in the underlying group) is the out-degree of the vertex in the directed power graph. ◀

We show that all the isomorphism invariant information of $R_3(G)$ is captured by a) the CCG-set of G in $R_3(G)$ along with their colors, and b) elements corresponding to their pairwise common neighborhood along with their colors. For this, we do a further reduction. The results in the rest of this section are required in Section 7.

[10] Here *connected directed graph* means that the underlying undirected graph is connected.

We define a new simple undirected colored graph $HD[n] = (V, E)$ for any natural number n, where $V = \{d : d|n\}$. The name of each vertex is treated as its color, i.e., here $col(v) = v$. The edge set is $E = \{\{u, v\} : v = u \cdot p \text{ or } u = v \cdot p \text{ for some prime } p\}$. One can see that $HD[n]$ is the Hasse diagram of the Poset defined over the set of all divisors of n with respect to the divisibility relation. Moreover, $HD[n]$ is also isomorphic to $R_3(\mathbb{Z}_n)$ (as a consequence of (2) of Lemma 28).

▶ Remark 36. (1) It is easy to see that $R_3(G)[Des(g_i)]$ is isomorphic to $HD[col(g_i)]$ for all $1 \le i \le m$. We can see that the isomorphism is unique as in each of these graphs, there is only one vertex with a particular color.
(2) Note that $\{y, y'\} \in E(R_3(G))$ if and only if (a) y, $y' \in Des(g_i)$ for some $1 \le i \le m$ and (b) $col(y) = p \cdot col(y')$ or $col(y') = p \cdot col(y)$ for some prime p.

Let $\bar{I}(i, j)$ denote the vertex in $R_3(G)$ that is of maximum color among the common descendant reachable vertices from both g_i and g_j. It is not hard to see that in the group G, $col(\bar{I}(i, j)) = |\langle g_i\rangle \cap \langle g_j\rangle|$. Note that for two distinct pairs (i, j) and (i', j'), $\bar{I}(i, j)$ and $\bar{I}(i', j')$ can be the same vertex in $R_3(G)$. It is not hard to see the proof of the following claim.

▷ Claim 37. In $R_3(G)$, $gcd(col(\bar{I}(i, j)), col(\bar{I}(s, j)))$ divides $col(\bar{I}(i, s))$.

Reduction rule 4. Consider $R_3(G)$. Recall that, in $R_3(G)$ a CCG-set $\{g_1, g_2, ..., g_m\}$ of G can be readily found. We make a new graph $R_4(G)$ as follows:
(1) Introduce the vertices $g_1, g_2, \ldots, g_m$ with their colors. (2) For each pair (i, j), $1 \le i < j \le m$, do the following: Find the vertex $\bar{I}(i, j)$ that is of maximum color among the descendant reachable vertices from both g_i and g_j. We add a vertex $I(i, j)$ in $R_4(G)$ and color it with $col(\bar{I}(i, j))$. Add edges $\{g_i, I(i, j)\}$ and $\{g_j, I(i, j)\}$.

Note that $R_4(G)$ is a bipartite graph where one part is a CCG-set and another part contains vertices marked as $I(i, j)$ for all (i, j). In $R_4(G)$, for distinct pairs (i, j) and (i', j'), $I(i, j)$ and $I(i', j')$ are distinct vertices, while in $R_3(G)$, $\bar{I}(i, j)$ and $\bar{I}(i', j')$ may be the same vertex. In other words, $R_4(G)$ may have several copies of vertex $\bar{I}(i, j)$.

We now present an algorithm to get back an isomorphic copy of $R_3(G)$ from $R_4(G)$.

Idea of the algorithm. In $R_4(G)$, we have a set of colored CC-generators. Also, there exist vertices $I(i, j)$ corresponding to each pairwise intersection of maximal cyclic subgroups $\langle g_i\rangle$ and $\langle g_j\rangle$ in G. $I(i, j)$ is the only common neighbor of g_i and g_j in $R_4(G)$. Using this information, we construct $R_3(G)$ in an iterative manner. First, we describe a sketch of the idea behind the process. There are m iterations in the process. In the 1^{st} iteration, we introduce $HD[col(g_1)]$. One can easily verify that $R_3(G)[Des(g_1)]$ is isomorphic to $HD[col(g_1)]$ (by (2) of Remark 36). In the 2^{nd} iteration, we introduce $HD[col(g_2)]$. As we know the color of $I(1, 2)$, we have information about the set of vertices common to both $HD[col(g_1)]$ and $HD[col(g_2)]$. Let u and v be the vertices with color $col(I(1, 2))$ in $HD[col(g_1)]$ and $HD[col(g_2)]$ respectively. We identify (via vertex-identification) the vertices with the same colors in $Des(u)$ (which is in $HD[col(g_1)]$) and $Des(v)$ (which is in $HD[col(g_2)]$)[11]. One can see that the resulting graph is isomorphic to the induced subgraph of $R_3(G)$ on $Des(g_1) \cup Des(g_2)$. Inductively the algorithm introduces $Des(g_1) \cup Des(g_2) \cup \ldots \cup Des(g_{j-1})$ at the end of the $(j-1)^{th}$ iteration. In the j^{th} iteration, we introduce $HD[col(g_j)]$. It is easy to note that the set of

[11] Since $HD[col(g_i)]$ is isomorphic to $R_3(G)[Des(g_i)]$, we can use the concept of Des in the graph $HD[col(g_i)]$ for all i.

vertices in $HD[col(g_j)]$ that are contained in $Des(g_j) \cap Des(g_s)$ for all $s \leq j-1$ has already been introduced. So, we need to identify the vertices introduced by the algorithm earlier with the corresponding subset of vertices in $HD[col(g_j)]$. We get the information of such vertices using the color of $I(s, j)$ for $s \leq j-1$. The details and correctness of the algorithm (Algorithm 3) are given in Section A.2.

7 Reconstruction Algorithms

Cameron asked the following question: "Question 2 [9]: Is there a simple algorithm for constructing the directed power graph or enhanced power graph from the power graph, or the directed power graph from the enhanced power graph?" Bubboloni and Pinzauti [7] gave an algorithm to reconstruct the directed power graph from the power graph. In this section, we show that with the tools we have developed, we can readily design algorithms to reconstruct the directed power graph from both the enhanced power graph and the power graph.

Suppose we are given a power graph (or an enhanced power graph) of some finite group G as input, i.e., $\Gamma = Pow(G)$ (or, $\Gamma = EPow(G)$). However, the group G is not given. As discussed in Section 5, we can find a CCG-set for G from the input graph. Next, we describe how to obtain a graph isomorphic to $R_4(G)$ from the CCG-set. From the vertices corresponding to a CCG-set of G, say $\{g_1, g_2, \dots, g_m\}$, we get the information about their degree in Γ and the pairwise common neighborhood of g_i and g_j in the respective graph. This immediately gives us $R_4(G)$. From $R_4(G)$, we know how to get back an isomorphic copy of $DPow(G)$ using the results in Section 6. All the steps in the process can be performed in polynomial-time.

For any two vertices u and v we can easily decide when to put an edge between them in the enhanced power graph by looking into the corresponding directed power graph: there is an edge $\{u, v\}$ in the enhanced power graph, if and only if both u and v belong to the closed-out-neighbourhood of some vertex in the directed power graph. In this way, we can construct the enhanced power graph from an input directed power graph. Therefore, we get a complete solution to Cameron's question.

References

1 Alfred V. Aho and John E. Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

2 László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

3 László Babai, D. Yu Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 310–324, 1982. doi:10.1145/800070.802206.

4 László Babai and Eugene M Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183, 1983. doi:10.1145/800061.808746.

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

6 Ravi B. Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987. doi:10.1016/0020-0190(87)90232-8.

7 Daniela Bubboloni and Nicolas Pinzauti. Critical classes of power graphs and reconstruction of directed power graphs. *arXiv preprint arXiv:2211.14778*, 2022.

8 Peter J. Cameron. The power graph of a finite group, ii. *Journal of Group Theory*, 13(6):779–783, 2010.

9 Peter J. Cameron. Graphs defined on groups. *International Journal of Group Theory*, 11(2):53–107, 2022.

10 Peter J. Cameron and Shamik Ghosh. The power graph of a finite group. *Discrete Mathematics*, 311(13):1220–1222, 2011. doi:10.1016/J.DISC.2010.02.011.

11 Ivy Chakrabarty, Shamik Ghosh, and M. K. Sen. Undirected power graphs of semigroups. In *Semigroup Forum*, volume 78, pages 410–426. Springer, 2009.

12 Bireswar Das, Jinia Ghosh, and Anant Kumar. The isomorphism problem of power graphs and a question of cameron. *arXiv preprint arXiv:2305.18936*, 2023. doi:10.48550/arXiv.2305.18936.

13 Min Feng, Xuanlong Ma, and Kaishun Wang. The full automorphism group of the power (di) graph of a finite group. *European Journal of Combinatorics*, 52:197–206, 2016. doi:10.1016/J.EJC.2015.10.006.

14 Valentina Grazian and Carmine Monetta. A conjecture related to the nilpotency of groups with isomorphic non-commuting graphs. *Journal of Algebra*, 633:389–402, 2023.

15 Joshua A. Grochow and Youming Qiao. On p-group isomorphism: search-to-decision, counting-to-decision, and nilpotency class reductions via tensors. In *36th Computational Complexity Conference (CCC 2021)*, volume 200, 2021. doi:10.4230/LIPICS.CCC.2021.16.

16 Martin Grohe and Sandra Kiefer. A linear upper bound on the Weisfeiler-Leman dimension of graphs of bounded genus. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

17 Martin Grohe and Daniel Neuen. Isomorphism, canonization, and definability for graphs of bounded rank width. *Communications of the ACM*, 64(5):98–105, 2021. doi:10.1145/3453943.

18 Martin Grohe, Daniel Neuen, and Pascal Schweitzer. A faster isomorphism test for graphs of small degree. *SIAM Journal on Computing*, pages FOCS18–1, 2020.

19 Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. An improved isomorphism test for bounded-tree-width graphs. *ACM Transactions on Algorithms (TALG)*, 16(3):1–31, 2020. doi:10.1145/3382082.

20 Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1010–1029. IEEE, 2015. doi:10.1109/FOCS.2015.66.

21 Marc Hellmuth and Tilen Marc. On the cartesian skeleton and the factorization of the strong product of digraphs. *Theoretical Computer Science*, 565:16–29, 2015. doi:10.1016/J.TCS.2014.10.045.

22 Wilfried Imrich, Sandi Klavzar, and Douglas F. Rall. *Topics in graph theory: Graphs and their Cartesian product.* CRC Press, 2008.

23 G. James Oxley. *Matroid Theory.* Oxford graduate texts in mathematics. Oxford University Press, 2006.

24 Andrei V. Kelarev and Stephen J. Quinn. A combinatorial property and power graphs of groups. *Contributions to general algebra*, 12(58):3–6, 2000.

25 Ajay Kumar, Lavanya Selvaganesh, Peter J. Cameron, and T. Tamizh Chelvam. Recent developments on the power graph of finite groups–a survey. *AKCE International Journal of Graphs and Combinatorics*, 18(2):65–94, 2021. doi:10.1080/09728600.2021.1953359.

26 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

27 Ralph McKenzie. Cardinal multiplication of structures with a reflexive relation. *Fundamenta Mathematicae*, 70(1):59–101, 1971.

28 Gary Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 225–235, 1980. doi:10.1145/800141.804670.

29 Gary L. Miller. On the nlog n isomorphism technique (a preliminary report). In *Proceedings of the tenth annual ACM symposium on theory of computing*, pages 51–58, 1978. doi:10.1145/800133.804331.

30 Dave Witte Morris, Joy Morris, and Gabriel Verret. Isomorphisms of cayley graphs on nilpotent groups. *New York J. Math*, 22:453–467, 2016.

31 Himadri Mukherjee. Hamiltonian cycles of power graph of abelian groups. *Afrika Matematika*, 30:1025–1040, 2019.

32 Daniel Neuen. Isomorphism testing for graphs excluding small topological subgraphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1411–1434. SIAM, 2022. doi:10.1137/1.9781611977073.59.

33 Joseph J. Rotman. *An introduction to the theory of groups*, volume 148. Springer Science & Business Media, 2012.

34 V. Vikraman Arvind and Peter J. Cameron. Recognizing the commuting graph of a finite group. *arXiv preprint*, 2022. doi:10.48550/arXiv.2206.01059.

35 Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, September 2000.

36 Daniel Wiebking. Graph isomorphism in quasipolynomial time parameterized by treewidth. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

37 Samir Zahirović, Ivica Bošnjak, and Rozália Madarász. A study of enhanced power graphs of finite groups. *Journal of Algebra and Its Applications*, 19(04):2050062, 2020.

A Appendix

A.1 Omitted Proofs

Proof of Lemma 14. Let $o(v) = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}$, where $k \geq 2$. The case when $u = e$ or u is a generator of $\langle v \rangle$ is easy as $N[u] = V(\Gamma_v)$ for any such element. Otherwise, since v is a CC-generator, $\langle u \rangle \lneq \langle v \rangle$. For u and z to be closed-twins, we must have $u \in \langle z \rangle$ or $z \in \langle u \rangle$. We show that for z to be a closed-twin of u, its order must be the same as that of u. We consider the case when $z \in \langle u \rangle$. The other case can be handled similarly. In this case, we have $o(z) | o(u)$.

Suppose both u and z are p-power elements for some prime $p \in \{p_1, p_2, \dots, p_k\}$. Moreover, without loss of generality, assume that $o(u) = p_1^{s_1}$ and $o(z) = p_1^{s_1'}$ where $s_1 > s_1'$. Note that $r_1 \geq s_1$. In this case, there is an element in $V(\Gamma_v)$ of order $p_1^{s_1'} p_2$ which is adjacent to z but not to u. More precisely, this element is an element in $\langle v \rangle$ of order $p_1^{s_1'} p_2$. So, in this case, u and z are not closed-twins in Γ_v.

Now suppose $o(u)$ is not a prime power. We first take z to be non-identity. Then, let $o(u) = p_1^{s_1} \dots p_k^{s_k}$, where $k \geq 2$. Let $o(z) = p_1^{s_1'} \dots p_k^{s_k'}$, where $s_j \geq s_j'$. Assume without loss of generality that $s_1 > s_1'$. As $o(u)$ is not a prime power order, we can take $s_2 \neq 0$. Now if $s_2' = 0$, consider an element x of order p_2 in Γ_v. Then x is a neighbor of u, but not of z. On the other hand, if $s_2' \neq 0$, we take an element y of order $p_1^{s_1}$. Again y is a neighbor of u but not of z. So, in this case also, u and z are not closed-twins in Γ_v.

Now suppose $o(u)$ is not a prime power, i.e., $o(u) = p_1^{s_1} p_2^{s_2} \dots p_k^{s_k}$ where $k \geq 2$ and z is the identity. We recall that since u is not a generator $\langle v \rangle$, there exists i such that $r_i > s_i$. We take an element x of order $p_i^{r_i}$ in Γ_v. One can check that x is adjacent to z but not to u. So, here also u and z are not closed-twins in Γ_v.

Note that if $o(u) = o(z)$, then they are closed-twins in Γ_v. ◀

▷ Claim 22. $deg_\Gamma(x) > deg_\Gamma(v)$.

Proof. As v is contained in only one covering cycle, we have $N_\Gamma(v) \subseteq N_\Gamma(x)$. This implies $deg_\Gamma(x) \geq deg_\Gamma(v)$. Moreover in Γ_v, the vertices x and v are closed-twins. If $o(x) = p^i$, then $deg(x) + 1 = p^i$. The graph $\Gamma_x = \Gamma[N[x]]$ is complete. So, $deg(v) + 1 = p^i$. Therefore, this case cannot arise. On the other hand, if $o(x)$ is not a prime power, we can apply [12] Lemma 14 and since $v \neq e$ and v is not a CC-generator, we can see that v and x are not closed-twins in Γ_x. Thus,$deg_\Gamma(x) > deg_\Gamma(v)$. ◁

Proof of Lemma 34. It is enough to prove the lemma for $k = 2$. Let $f_1 : V(DPow(G_1)) \to V(DPow(H_1))$ and $f_2 : V(DPow(G_2)) \to V(DPow(H_2))$ be two isomorphisms from $DPow(G_1)$ to $DPow(H_1)$ and from $DPow(G_2)$ to $DPow(H_2)$ respectively. Let us define $f : V(DPow(G)) \to V(DPow(H))$ as $f((u_1, u_2)) = (f_1(u_1), f_2(u_2))$. Since f_1 and f_2 are bijections, so is f. We show that f preserves the edge relations between $DPow(G)$ and $DPow(H)$. Let us consider an edge $((u_1, u_2), (v_1, v_2))$ from $E(DPow(G)) = E(DPow(G_1) \boxtimes DPow(G_2))$ (This equality follows from Lemma 32.). Now from Definition 2 and the facts that f_1 and f_2 are isomorphisms from $DPow(G_1)$ to $DPow(H_1)$ and from $DPow(G_2)$ to $DPow(H_2)$ respectively, we have the following three scenarios:

1. $u_1 = v_1$ and $(u_2, v_2) \in E(DPow(G_2))$. In this case, $f_1(u_1) = f_1(v_1)$ and $(f_2(u_2), f_2(v_2)) \in E(DPow(H_2))$.
2. $u_2 = v_2$ and $(u_1, v_1) \in E(DPow(G_1))$. In this case, $f_2(u_2) = f_2(v_2)$ and $(f_1(u_1), f_1(v_1)) \in E(DPow(H_1)$.
3. $(u_1, v_1) \in E(DPow(G_1))$ and $(u_2, v_2) \in E(DPow(G_2))$. In this case, $(f_1(u_1), f_1(v_1)) \in E(DPow(H_1))$ and $(f_2(u_2), f_2(v_2)) \in E(DPow(H_2))$.

In all the three scenarios, by Definition 2, we have $((f_1(u_1), f_2(u_2)), (f_1(v_1), f_2(v_2))) \in E(Dpow(H_1) \boxtimes DPow(H_2))$. Therefore by Lemma 32, $(f((u_1, u_2)), f((v_1, v_2))) \in E(DPow(H))$.

For the other direction, let $f : V(DPow(G)) \to V(DPow(H))$ be an isomorphism between $DPow(G)$ and $DPow(H)$. Consider the sets $A_i = \{(u, v) \in V(DPow(G)) : \textit{out-deg}((u, v))$ divides $|G_i|\}$ and $B_i = \{(u', v') \in V(DPow(H)) : \textit{out-deg}((u', v'))$ divides $|H_i|\}$ for $i = 1, 2$. Recall that here the out-degree of a vertex is the order of the element and $o((u, v)) = o(u) \cdot o(v)$. Since $|G_1| \times |G_2| = |G|$ and $gcd(|G_1|, |G_2|) = 1$, it is easy to see that A_i indeed corresponds to $V(DPow(G_i))$ for $i = 1, 2$. Also, the subgraph of $DPow(G_1 \times G_2)$ induced by A_i corresponds to $DPow(G_i)$ for $i = 1, 2$. Similarly, we can see that B_i corresponds to $V(DPow(H_i))$ and the subgraph induced by B_i corresponds to $DPow(H_i)$ for $i = 1, 2$. Now the isomorphism f preserves the out-degrees of the vertices. We denote the restriction of f on A_i by f_i. Then it is easy to see that f_i is a bijection from A_i to B_i. Also, there is only one element, namely the identity element, of out-degree 1 (self-loop) and common in both A_1 and A_2. Also, that element is unique in $DPow(G)$. One can see that $f_i : V(DPow(G_i)) \to V(DPow(H_i))$ is an isomorphism between $DPow(G_i)$ and $DPow(H_i)$, for all $i = 1, 2$. ◀

A.2 Algorithm to construct an isomorphic copy of Reduction graph

Here, we give a detailed description of the algorithm to construct $R_3(G)$ from $R_4(G)$ discussed in Section 6.

As indicated in the idea behind the algorithm in Section 6 and in Line 12 of Algorithm 3, vertices in the old graph and $HD[col(g_j)]$ are identified. In Claim 40 we show that these vertices can be identified without conflict.

[12] Here x and v are to be treated as the variables v and u in Lemma 14.

Algorithm 3 To construct an isomorphic copy of $R_3(G)$ from $R_4(G)$.

```
    Input: $R_4(G)$
 1: $X_1 \leftarrow HD[col(g_1)]$
 2: $j \leftarrow 2$
 3: while $j \leq m$ do
 4:     Introduce $Y_j = HD[col(g_j)]$
 5:     $s \leftarrow 1$
 6:     $h_{j,0} \leftarrow \emptyset$                         ▷ Mapping for vertex identification
 7:     while $s \leq j-1$ do
 8:         Consider $I(s,j)$.
 9:         $h_{j,s} \leftarrow h_{j,s-1} \cup$
            $\{(u,v) : col(u) = col(v)$ where $u \in Des(g_s) \subseteq V(X_{j-1})$ s.t $col(u) | col(I(s,j))$ and
            $v \in V(Y_j)\}$
10:         $s \leftarrow s+1$
11:     end while
12:     For all $(u,v) \in h_{j,j-1}$ vertex-identify $u$ and $v$ and color the new vertex with $col(u)$.
13:     $X_j \leftarrow$ The graph obtained after the above vertex identification of $X_{j-1}$ and $Y_j$.
14:     $j \leftarrow j+1$
15: end while
16: Return $X_m$
```

▶ **Lemma 38.** *The graph X_m returned by Algorithm 3 is isomorphic to $R_3(G)$.*

Proof. We show by induction on j that the constructed graph up to the j^{th} step is isomorphic to the subgraph of $R_3(G)$ induced on $Des(g_1) \cup Des(g_2) \cup \ldots \cup Des(g_j)$. This shows that after the m^{th} iteration, we can get an isomorphic copy of $R_3(G)$.

▷ Claim 39. $X_j \cong R_3(G)[Des(g_1) \cup Des(g_2) \cup \cdots \cup Des(g_j)]$, $\forall 1 \leq j \leq m$.

Proof of claim. For simplicity, we denote $R_3(G)[Des(g_1) \cup Des(g_2) \cup \cdots \cup Des(g_j)]$ by $R_3(j)$ in the remaining part of the proof. With this, $R_3(1)$ denotes $R_3(G)[Des(g_1)]$.

By Remark 36, $X_1 = HD[col(g_1)]$ is isomorphic to $R_3(1)$ by a unique isomorphism, say f_1. If we take f_0 to be the empty map, then f_1 extends f_0.

We prove by induction on j that X_j is isomorphic to $R_3(j) = R_3[Des(g_1) \cup Des(g_2) \cup \cdots \cup Des(g_j)]$ via a map f_j that extends the isomorphism f_{j-1}.

By induction hypothesis, let us assume that $X_{j-1} \cong R_3(G)[Des(g_1) \cup \cdots \cup Des(g_{j-1})]$ and f_{j-1} is an isomorphism between X_{j-1} and $R_3(j-1)$ derived by extending f_{j-2}. We show that f_j is an extension of f_{j-1} and f_j is an isomorphism between X_j and $R_3(j)$.

However, before we go into the details of the inductive case, we address the following important issue.

In the j^{th} iteration of the outer while loop and just after the execution of Line 4 of Algorithm 3, the current graph is the disjoint union of X_{j-1} and Y_j. Now to get X_j, some vertices of X_{j-1} and Y_j are vertex-identified using the tuples stored in $h_{j,j-1}$ as described in Line 12 of Algorithm 3. Observe that two vertices in Y_j cannot be identified with the same vertex in X_{j-1}, because in $Y_j = HD[col(g_j)]$, no two vertices have the same color. However, there is a possibility that two or more vertices of X_{j-1} are assigned to be identified with the same vertex of Y_j. We show that this case does not arise. To do this, we first define the following sets:

$$
\begin{aligned}
Y_{j,1} &= \{v \in V(Y_j) \;:\; col(v)\big|col(I(1,j))\}\\
Y_{j,s} &= Y_{j,s-1} \cup \{v \in V(Y_j) \;:\; col(v)\big|col(I(s,j))\}, \quad s = 2,\dots,j-1\\
X_{j-1,1} &= \{u \in V(X_{j-1}) \;:\; u \in Des(g_1) \text{ and } col(u)\big|col(I(1,j))\}\\
X_{j-1,s} &= X_{j-1,s-1} \cup \{u \in V(X_{j-1}) \;:\; u \in Des(g_s) \text{ and } col(u)\big|col(I(s,j))\}, \quad s = 2,\dots,j-1
\end{aligned}
$$

Now $h_{j,j-1}$ is updated from $h_{j,0} = \emptyset$ by the following rule: $h_{j,s} = h_{j,s-1} \cup \{(u,v) \mid col(u) = col(v)$ where $u \in X_{j-1,s}$ and $v \in Y_{j,s}\}$ (as described in Line 9 in Algorithm 3)[13]. Since there is a unique vertex of any particular color in Y_j, we can see $h_{j,s}$ as a well-defined function from $X_{j-1,s}$ to $Y_{j,s}$. Now to show that $h_{j,j-1}$ gives a conflict-free vertex identification process, we show that $h_{j,s}$ is a bijection and an extension of $h_{j,s-1}$. Since $h_{j,s-1} \subseteq h_{j,s}$, it is enough to prove the following claim:

▷ Claim 40. The map $h_{j,s} : X_{j-1,s} \to Y_{j,s}$ is a bijection, for all $1 \le s \le j-1$.

Proof of claim: First, we show that $h_{j,s}$ is onto for all $s = 1,\dots,j-1$. For this, take a vertex v from $Y_{j,s}$. Then $col(v)|col(I(i,j))$ for some $i \le s$. So,[14] there exists a vertex $u \in Des(g_i)$ in $X_{j-1,s}$ such that $col(u) = col(v)$ and $h_{j,s}(u) = v$.

Now we prove that $h_{j,s}$ is one-to-one using induction on s. For the base case, it is easy to see that $h_{j,1} : X_{j-1,1} \to Y_{j,1}$ is a bijection since $X_{j-1,1}$ and $Y_{j,1}$ contains colored vertices corresponding to each divisor of $col(I(1,j))$ and color of each vertex is distinct. By induction hypothesis we assume that $h_{j,s-1} : X_{j-1,s-1} \to Y_{j,s-1}$ is a bijection. Now for the inductive case, we consider $h_{j,s} : X_{j-1,s} \to Y_{j,s}$. We need to prove that $h_{j,s}$ is one-one. Suppose that $u \in X_{j-1,s}$ is paired with $v \in Y_{j,s}$ to be stored at $h_{j,s}$ in the s^{th} iteration of the inner while loop (Line 9 of Algorithm 3). We need to argue that the pairing does not violate the one-to-one condition. We do this in two cases.

Case 1: The vertex v was not encountered in any of the previous iterations, i.e., $v \notin Y_{j,s-1}$. So by definition of $X_{j-1,s-1}$, there is no vertex of color $col(v)$ in $X_{j-1,s-1}$. Since $col(u) = col(v)$, we have $u \in X_{j-1,s} \setminus X_{j-1,s-1}$. So, (u,v) is added to $h_{j,s}$ in the s^{th} iteration only, where v is in $Y_{j,s}$. Therefore, $X_{j-1,s}$ contains exactly one vertex of color $col(u)$. This implies that v cannot be paired with any vertex except u.

Case 2: The vertex v was encountered before the s^{th} iteration, and $i \le (s-1)$ is the most recent such iteration. This means that there exists u' in the old graph (*i.e.*, $X_{j-1,s-1}$) such that $h_{j,s-1}(u') = v$. Since $h_{j,s-1}$ is a bijection by induction hypothesis, u' is the only preimage of v under $h_{j,s-1}$. We show that $u = u'$.

Observe that there is a vertex $w \in Des(g_s)$ in X_{j-1} such that $col(w) = col(I(s,j))$. By the algorithm, $col(u)|col(I(s,j))$. So, $u \in Des(w)$. Similarly, there is a vertex $w' \in Des(g_i)$ in X_{j-1} such that $col(w') = col(I(i,j))$ and by the algorithm $col(u')|col(I(i,j))$. So $u' \in Des(w')$. Since $col(u) = col(u')$, $col(u')|col(I(i,j))$ and $col(u)|col(I(s,j))$, we conclude that $col(u)$ divides $gcd(col(I(i,j)), col(I(s,j)))$. So, by Claim 37, $col(u)|col(I(i,s))$.

Now we consider the subgraph of X_{j-1} induced by $Des(g_i) \cap Des(g_s)$. If $x \in Des(g_i) \cap Des(g_s)$ is the vertex with color $col(I(i,s))$, then this subgraph is formed by the descendants of x. Since the descendants of x are exactly the vertices in $Des(g_i)$ and $Des(g_s)$ with colors as factors of $col(I(i,s))$, both u and u' are in $Des(x)$. Now, $Des(x)$ has a unique vertex of a particular color. Therefore, as u and u' have the same color, $u = u'$.

[13] Note that when $u \in X_{j-1,j-1}$ is identified with $v \in Y_{j,j-1}$, we color it with $col(u)$ and for simplicity we name the new vertex as u.

[14] Since $X_{j-1} \cong R_3(j-1)$, the concept of descendant reachability can also be defined in X_{j-1}. Therefore, it makes sense to use $Des(g)$ in X_{j-1} for any vertex g.

Hence, we have proved that $h_{j,s}$ is one-one in both the cases. Therefore, we can conclude that $h_{j,s} : X_{j-1,s} \to Y_{j,s}$ is a bijection for all $1 \leq s \leq j-1$. ◁

From the above claim, we can conclude that in the j^{th} iteration of the outer while loop, the identification process done in Line 12 in Algorithm 3 via the mapping $h_{j,j-1}$ is correct. Next, we show that the graph X_j (output in Line 13), derived after the identification process on X_{j-1} and Y_j, is indeed isomorphic to $R_3(j)$.

For $j \geq 2$, we define $f_j : V(X_j) \to V(R_3(j))$ in the following manner:

$$f_j(x) = \begin{cases} f_{j-1}(x) & \text{if } x \in V(X_{j-1}) \\ y & \text{otherwise where } y \in V(R_3(j)) \setminus V(R_3(j-1)) \\ \text{ and } col(y) = col(x). \end{cases} \tag{1}$$

To show that f_j is well defined, it is enough to argue that for each $x \in V(X_j) \setminus V(X_{j-1})$, there exists a unique $y \in V(R_3(j)) \setminus V(R_3(j-1))$ such that $col(y) = col(x)$. Observe that $V(X_j) \setminus V(X_{j-1})$ is the set of vertices of $Y_j = HD[col(g_j)]$ that have not been identified in the j^{th} iteration. So, for any vertex $x \in V(X_j) \setminus V(X_{j-1})$, $col(x)$ divides $col(g_j)$ but $col(x)$ does not divide $col(I(i,j))$ for any $i < j$. This means, for each such x, there exists y in $V(R_3(j)) \setminus V(R_3(j-1))$ with color $col(x)$ and this y is unique since $V(R_3(j)) \setminus V(R_3(j-1))$ contains the vertices of $Des(g_j)$ that are not descendant reachable from any g_i where $i < j$. The uniqueness of colors in $Y_j = HD[col(g_j)]$ also implies that f_j is a bijection.

Now to show that f_j is an isomorphism between X_j and $R_3(j)$, it remains to show that f_j preserves edge relations between X_j and $R_3(j)$.

Here, we want to emphasize that it might happen that two vertices x, x' in X_{j-1} are not adjacent to each other, but after the vertex identification process in the j^{th} iteration, there is an edge between x and x' in X_j. Moreover, through the following claim, we want to show that this incident has a correspondence in $R_3(j)$.

▷ Claim 41. Let x, x' be two vertices in the old graph (i.e., X_{j-1}) that take part in the vertex identification process in the j^{th} iteration, i.e., $x, x' \in X_{j-1,j-1}$. Then, $\{x, x'\} \notin E(X_{j-1})$, but $\{x, x'\} \in E(X_j)$ if and only if $\{f_{j-1}(x), f_{j-1}(x')\} \notin E(R_3(j-1))$, but $\{f_j(x), f_j(x')\} \in E(R_3(j))$.

Proof of claim. As f_{j-1} is an isomorphism between X_{j-1} and $R_3(j-1)$, we have $\{x, x'\} \notin E(X_{j-1})$ if and only if $\{f_j(x), f_j(x')\} \notin E(R_3(j-1))$.

Now, assume that $\{x, x'\} \notin E(X_{j-1})$ but $\{x, x'\} \in E(X_j)$. Since $x, x' \in X_{j-1,j-1}$, the vertices x, x' get identified with some elements z, z' respectively in Y_j such that $\{z, z'\} \in E(Y_j)$. Also, $col(x) = col(z) = col(f_j(x))$ and $col(x') = col(z') = col(f_j(x'))$. Since $Y_j = HD[col(g_j)]$ and $\{z, z'\} \in E(Y_j)$, by definition either $col(z) = col(z') \cdot p$ or $col(z') = col(z) \cdot p$ for some prime p. Therefore, either $col(f_j(x)) = col(f_j(x')) \cdot p$ or $col(f_j(x')) = col(f_j(x)) \cdot p$ for some prime p. Moreover, $f_j(x), f_j(x') \in Des(g_j)$. Hence, by (2) of Remark 36, $\{f_j(x), f_j(x')\} \in E(R_3(j))$.

Conversely, assume that $\{f_j(x), f_j(x')\} \in E(R_3(j))$. Since $x, x' \in X_{j-1,j-1}$, x and x' must have been identified with some vertices z and z' in Y_j respectively such that $col(x) = col(z)$ and $col(x') = col(z')$. Now, because of (2) of Remark 36, $\{f_j(x), f_j(x')\} \in E(R_3(j))$ implies either $col(f_j(x)) = col(f_j(x')) \cdot p$ or $col(f_j(x')) = col(f_j(x)) \cdot p$ for some prime p. Therefore, either $col(z) = col(z') \cdot p$ or $col(z') = col(z) \cdot p$. So, $\{z, z'\} \in E(Y_j)$. Hence, after the vertex identification, $\{x, x'\} \in E(X_3(j))$. ◁

Now to show the preservation of edge relations, we consider the following cases, not necessarily disjoint:

(a) Let $x, x' \in V(X_{j-1})$, i.e., both the vertices are from the graph obtained in the previous iteration of the outer while loop. Then, by definition of f_j in 1, $f_j(x) = f_{j-1}(x)$ and $f_j(x') = f_{j-1}(x')$. Since by induction hypothesis f_{j-1} is an isomorphism between X_{j-1} and $R_3(j-1)$, $\{x, x'\} \in E(X_{j-1}) \iff \{f_{j-1}(x), f_{j-1}(x')\} \in E(R_3(j-1))$. The remaining case is covered by Claim 41.

(b) Let x, x' be two vertices in X_j that appear in the "Y_j-part" of X_j. More precisely, x, x' belong to the disjoint union of $V(X_j) \setminus V(X_{j-1})$ (which is the set of vertices which are newly introduced in the j^{th} iteration of the outer while loop but not identified in the same) and $X_{j-1,j-1}$ (which corresponds to the set of vertices that are the result of vertex identification of $X_{j-1,j-1}$ and $Y_{j,j-1}$ in the j^{th} iteration). Since $Y_j = HD[col(g_j)] \cong R_3(G)[Des(g_j)]$ by Remark 36, $\{x, x'\} \in E(X_j) \iff \{f_j(x), f_j(x')\} \in E(R_3(j))$.

(c) Let x be a vertex from the old graph X_{j-1} which has not been identified in the j^{th} iteration, i.e., $x \in V(X_{j-1}) \setminus X_{j-1,j-1}$. Let x' be a newly added vertex that has not been identified in the j^{th} iteration, i.e., $x' \in V(X_j) \setminus V(X_{j-1})$. It is not hard to see that $\{x, x'\}$ is not an edge of the disjoint union of X_{j-1} and Y_j (before the identification process). Since none of x and x' has taken part in the identification process in this iteration, we have $\{x, x'\} \notin E(X_j)$. Now as f_j is a bijection, we also have the following: $f_j(x) \in V(R_3(j-1)) \setminus Des(g_j)$ and $f_j(x') \in V(R_3(j)) \setminus V(R_3(j-1))$. Since $f_j(x)$ and $f_j(x')$ are not in same $Des(u)$ for any vertex u in $R_3(j)$, $\{f_j(x), f_j(x')\}$ is not an edge in $R_3(j)$. Thus, it is proved that f_j is an isomorphism between X_j and $R_3(j)$. So, we can conclude that $X_m \cong R_3(m)$. It is easy to see that $R_3(m)$ is $R_3(G)$. This concludes the proof of Claim 39. ◁

Hence, the algorithm is correct, and we can return an isomorphic copy of $R_3(G)$ from $R_4(G)$. ◀

A Myhill-Nerode Style Characterization for Timed Automata with Integer Resets

Kyveli Doveri
University of Warsaw, Poland

Pierre Ganty
IMDEA Software Institute, Pozuelo de Alarcón, Madrid, Spain

B. Srivathsan
Chennai Mathematical Institute, India
CNRS IRL 2000, ReLaX, Chennai, India

Abstract

The well-known Nerode equivalence for finite words plays a fundamental role in our understanding of the class of regular languages. The equivalence leads to the Myhill-Nerode theorem and a canonical automaton, which in turn, is the basis of several automata learning algorithms. A Nerode-like equivalence has been studied for various classes of timed languages.

In this work, we focus on timed automata with integer resets. This class is known to have good automata-theoretic properties and is also useful for practical modeling. Our main contribution is a Nerode-style equivalence for this class that depends on a constant K. We show that the equivalence leads to a Myhill-Nerode theorem and a canonical one-clock integer-reset timed automaton with maximum constant K. Based on the canonical form, we develop an Angluin-style active learning algorithm whose query complexity is polynomial in the size of the canonical form.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Timed languages, Timed automata, Canonical representation, Myhill-Nerode equivalence, Integer reset

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.21

Related Version *Full Version*: `https://arxiv.org/abs/2410.02464` [9]

Funding *Kyveli Doveri*: Supported by the ERC grant INFSYS, agreement no. 950398. Results partly obtained when previously affiliated with the IMDEA Software Institute.
Pierre Ganty: This publication is part of the grant PID2022-138072OB-I00, funded by MCIN/AEI/10.13039/501100011033/FEDER, UE.

1 Introduction

A cornerstone in our understanding of regular languages is the Myhill-Nerode theorem. This theorem characterizes regular languages in terms of the Nerode equivalence $\sim_L$: for a word w we write $w^{-1}L = \{z \mid wz \in L\}$ for the *residual language* of w w.r.t. L; and for two words u, v we say $u \sim_L v$ if $u^{-1}L = v^{-1}L$.

▶ **Theorem 1.1** (Myhill-Nerode theorem). *Let L be a language of finite words.*

- *L is regular iff the Nerode equivalence has a finite index.*
- *The Nerode equivalence is coarser than any other monotonic L-preserving equivalence.*

An equivalence is said to be *monotonic* if $u \approx v$ implies $ua \approx va$ for all letters a and is L-preserving if each equivalence class is either contained in L or disjoint from L. [1] An equivalence over words being monotonic makes it possible to construct an automaton with

[1] The exact term would be “right monotonic” because it only considers concatenation to the right of the word. Throughout the paper we simply write monotonic to keep it short but we mean right monotonic.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 21; pp. 21:1–21:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

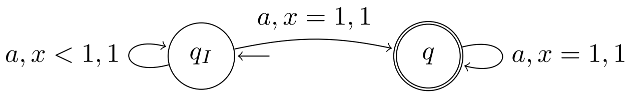


Figure 1 Automaton accepting $L = \{(t_1 \cdot a) \dots (t_n \cdot a) \mid t_1 + \cdots + t_n = 1\}$ with alphabet $\Sigma = \{a\}$.

states being the equivalence classes. The Nerode equivalence being the coarsest makes the associated automaton the minimal (and a canonical) deterministic automaton for the regular language. Our goal in this work is to obtain a similar characterization for certain subclasses of timed languages.

Timed languages and timed automata were introduced by Alur and Dill [1] as a model for systems with real-time constraints between actions. Ever since its inception, the model has been extensively studied for its theoretical aspects and practical applications. In this setting, words are decorated with a delay between consecutive letters. A *timed word* is a finite sequence $(t_1 \cdot a_1)(t_2 \cdot a_2) \cdots (t_n \cdot a_n)$ where each $t_i \in \mathbb{R}_{\geq 0}$ and each a_i is a letter taken from a finite set Σ called an *alphabet*. A timed word associates a time delay between letters: a_1 was seen after a delay of t_1 from the start, the next letter a_2 appears t_2 time units after a_1, and so on. Naturally, a timed language is a set of timed words. A timed automaton is an automaton model that recognizes timed languages. Figures 1 and 2 present some examples (formal definitions appear later). In essence, a timed automaton makes use of *clocks* to constrain time between the occurrence of transitions. In Figure 1, the variable x denotes a clock. The transition labels are given by triples comprising a letter (e.g. a), a clock constraint (e.g. $x = 1$), and a multiplicative factor (0 or 1) for the clock update. Intuitively, the semantics of the transition from q_I to q is as follows: the automaton reads the letter a when the value of the clock held in x is exactly 1 and updates the clock value to $1 \times x$. If the third element of the transition label is 0, then the transition updates the value of x to $0 \times x = 0$. We refer to the second element of the transition label as the transition *guard* and the third element as the *reset*. It is worth mentioning that the transition guards feature constants given by integer values, meaning that a guard like $x = 0.33$ is not allowed. Next, we argue how challenging it is to define a Nerode-style equivalence for timed languages.

Challenge 1. *The Nerode equivalence lifted as it is has infinitely many classes.* For example, the timed automaton of Figure 1 accepts a timed word $(t_1 \cdot a) \dots (t_n \cdot a)$ as long as $t_1 + \cdots + t_n = 1$. The timed language L of that automaton has infinitely many quotients. Indeed let $0 < t_1 < 1$, we have that $(t_1 \cdot a)^{-1} L = \{(t_2 \cdot a) \dots (t_n \cdot a) \mid t_2 + \cdots + t_n = 1 - t_1\}$. Observe that different values for t_1 yield different quotients, hence L has uncountably many quotients.

Challenge 2. *Two words with the same residual languages may never go to the same control state in any timed automaton.* Figure 2 gives an example of a timed language that exhibits this challenge. Consider the words $u = (0.5 \cdot a)$ and $v = (1.5 \cdot a)$. The residual of both these words is the singleton language $\{(0.5 \cdot b)\}$. Suppose both u and v go to the same control state q in the timed automaton. After reading u (resp. v), clocks which are possibly reset will be 0, whereas the others will be 0.5 (resp. 1.5). Suppose v is accepted via a transition sequence $q_I \to q \to q_F$. Since guards contain only integer constants, the guard on $q \to q_F$ should necessarily be of the form $x = 2$ for some clock x which reaches q with value 1.5. The same transition can then be taken from u to give $u(2 \cdot b)$ or $u(1.5 \cdot b)$ depending on the value of x after reading u. A contradiction. This example shows there is no hope to identify states of a timed automaton through quotients of a Nerode-type equivalence. The equivalence that we are aiming for needs to be stronger, and further divide words based on some past history.

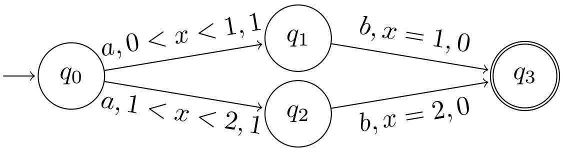


Figure 2 Automaton accepting $L_2 = \{(t_1 \cdot a)(t_2 \cdot b) \mid \text{either } 0 < t_1 < 1 \text{ and } t_1 + t_2 = 1, \text{ or } 1 < t_1 < 2 \text{ and } t_1 + t_2 = 2\}$ with alphabet $\Sigma = \{a, b\}$.

Challenge 3. *The Nerode-style equivalence should be amenable to a timed automaton construction.* In the case of untimed word languages, monotonicity of the Nerode equivalence immediately led to an automaton construction. We need to find the right notion of monotonicity for the class of automata that we want to build from the equivalence.

A machine independent characterization for deterministic timed languages has been studied by Bojańczyk and Lasota [6]. They circumvented the above challenges by considering a new automaton model *timed register automata* that generalizes timed automata. This automaton model makes use of registers to store useful information, for instance for the language in Figure 2, a register stores the value 0.5 after reading $(0.5 \cdot a)$ and $(1.5 \cdot a)$. This feature helps in resolving Challenge 2. For the question of finiteness mentioned in Challenge 1, timed register automata are further viewed as a restriction of a more general model of automata that uses the abstract concept of *Frankel-Mostowski* sets in its definition. Finiteness is relaxed to a notion of *orbit-finiteness*.

The work of An et al. [3] takes another approach to these challenges by considering a subclass of timed languages which are called *real-time languages*. These are languages that can be recognized using timed automata with a single clock that is reset in every transition. Therefore, after reading a letter, the value of the clock is always 0. This helps in solving the challenges, resulting in a canonical form for real-time languages.

Our work. As we have seen, to get a characterization which also lends to an automaton construction, either the automaton model has been modified or the characterization is applied to a class of languages where the role of the clock is restricted to consecutive letters. Our goal is to continue working with the same model as timed automata and apply a characterization to a different subclass.

In this work, we look at languages recognized by timed automata with integer resets (IRTA). These are automata where clock resets are restricted to transitions that contain a guard of the form $x = c$ for some clock x and some integer c [17]. The class of languages recognized by IRTA is incomparable with real-time languages. Moreover, it is known that IRTA can be reduced to 1-clock-deterministic IRTA [15], or 1-IRDTA for short. The proof of this result effectively computes, given an IRTA, a timed language equivalent 1-IRDTA. Here is our main result which gives a Myhill-Nerode style characterization for IRTA languages.

▶ **Theorem 1.2.** *Let L be a timed language.*

- *L is accepted by a timed automaton with integer resets iff there exists a constant K such that $\approx^{L,K}$ is K-monotonic and has a finite index.*
- *The $\approx^{L,K}$ equivalence is coarser than any K-monotonic L-preserving equivalence.*

Intuitively, one should think of K as the largest integer that needs to appear in the guards of an accepting automaton. The goal of the paper is to identify the notion of K-monotonicity and the equivalence $\approx^{L,K}$ that exhibit the above theorem. The characterization also leads to a canonical form for IRTA. In practice, the integer reset assumption allows for modeling multiple situations [17].

To the best of our knowledge, there is no learning algorithm that can compute an IRTA for systems that are known to satisfy the integer reset assumption. We fill this gap and show how Angluin's style learning [4] can be adapted to learn 1-IRDTA.

Related work. Getting a canonical form for timed languages has been studied in several works: [6] and [14] focus on a machine independent characterization for deterministic timed languages, whereas the works [10, 3, 21] extend the study of the canonical forms to an active learning algorithm. Languages accepted by event-recording automata are a class of languages where the value of the clocks is determined by the input word. This helps in coming up with a canonical form [10]. In [21], the author presents a Myhill-Nerode style characterization for deterministic timed languages by making use symbolic words rather than timed words directly. The author shows that the equivalence has a finite index iff the language is recognizable (under the notion of recognizability using right-morphisms proposed by Maler and Pnueli [14]). Further, Maler and Pnueli have given an algorithm to convert recognizable timed languages to deterministic timed automata, which resets a fresh clock in every transition and makes use of clock-copy updates $x := y$ in the transitions. It is known that automata with such updates can be translated to classical timed automata [16, 7].

Learning timed automata is a topic of active research. The foundations of timed automata learning were laid in the pioneering work of Grinchtein et al. [10] by providing a canonical form for event-recording automata (ERAs). These are automata having a clock for each letter in the alphabet, and a clock x_a records the time since the last occurrence of a. The canonical form essentially considers a separate state for each region. Since there are as many clocks as the number of letters, there are at least $|\Sigma|!$ number of regions. This makes the learning algorithm prohibitively expensive to implement. In contrast, as we will see, we are able to convert IRTAs into a subclass of single-clock IRTAs. If K is the maximum constant, there are only $2K + 2$ many regions. Later works on learning ERAs have considered identifying other forms of automata that merge the states of the canonical form [12, 11, 13]. Other models for learning timed systems consider one-clock timed automata [22, 2] and Mealy machines with timers [8, 19]. Approaches other than active learning for timed automata include passive learning of discrete timed automata [20] and learning timed automata using genetic programming [18].

We have considered a subclass of deterministic timed languages. Therefore, our class does fall under the purview of the [21, 14] work – however, the fundamental difference is that we continue to work with timed words and not symbolic timed words. This gives an alternate perspective and a direct and simpler 1-clock IRTA construction. The simplicity and directness also apply when it comes to learning 1-clock IRTA.

Outline of the paper. In Section 3, we define the class of one-clock languages with integer resets, and their acceptors thereof: the 1-clock Timed Automata (or 1-TA) with clock constraints given by region equivalence classes and transitions that always reset on integer clock values. Section 4 puts forward a notion of K-monotonicity and characterizates K-monotonic equivalences as a certain type of integer reset automata. Subsequently, Section 5 presents the Nerode-style equivalence, the Myhill-Nerode theorem and some examples applying the theorem. Finally, in Section 6, we give an algorithm to compute and learn the canonical form.

2 Background

Words and languages. An *alphabet* is a finite set of *letters* which we typically denote by Σ. An *untimed word* is a finite sequence $a_1 \cdots a_n$ of letters $a_i \in \Sigma$. We denote by Σ^* the set of untimed words over Σ. An *untimed language* is a subset of Σ^*. A *timestamp* is a finite sequence of non-negative real numbers. We denote the latter set by $\mathbb{R}_{\geq 0}$ and the set of all timestamps by $\mathbb{T}$. A *timed word* is a finite sequence $(t_1 \cdot a_1) \cdots (t_n \cdot a_n)$ where $a_1 \cdots a_n \in \Sigma^*$ and $t_1 \cdots t_n \in \mathbb{T}$. We denote the set of timed words by $\mathbb{T}\Sigma^*$. Given a timed word $u = (t_1 \cdot a_1) \ldots (t_n \cdot a_n) \in \mathbb{T}\Sigma^*$ we denote by $\sigma(u)$ the sum $t_1 + \cdots + t_n$ of its timestamps. A *timed language* is a set of timed words. As usual, we denote the empty (un)timed word by ϵ. The *residual language* of an (un)timed language L with regard to a (un)timed word u is defined as $u^{-1}L = \{w \mid uw \in L\}$. Therefore it is easy to see that $\epsilon^{-1}L = L$ for every (un)timed language L.

Timed automata [1] are recognizers of timed languages. Since we focus on subclasses of timed automata with a single clock, we do not present the definition of general timed automata. Instead, we give a modified presentation of one-clock timed automata that will be convenient for our work.

One-clock timed automata. A *One-clock Timed Automaton* (1-TA) over Σ is a tuple $\mathcal{A} = (Q, q_I, T, F)$ where Q is a finite set of states, $q_I \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $T \subseteq Q \times Q \times \Sigma \times \Phi \times \{0,1\}$ is a finite set of transitions where Φ is the set of clock constraints given by

$$\phi ::= x < m \quad | \quad m < x \quad | \quad x = m \quad | \quad \phi \wedge \phi \ , \text{ where } m \in \mathbb{N}.$$

For a clock constraint ϕ, we write $[\![\phi]\!]$ for the set of non-negative real values for x that satisfies the constraint. Notice that we have disallowed guards of the form $x \leq m$ which appear in standard timed automata literature, since its effect can be captured using two transitions, one with $x = m$ and another with $x < m$. A transition is a tuple (q, q', a, ϕ, r) where ϕ is a clock constraint called the *guard* of the transition and $r \in \{0,1\}$ denotes whether the single clock x is *reset* in the transition.

We say that a 1-TA with transitions T is *deterministic* whenever for every pair $\theta = (q, q', a, \phi, r)$ and $\theta_1 = (q_1, q_1', a_1, \phi_1, r_1)$ of transitions in T such that $\theta \neq \theta_1$ we have that either $q \neq q_1$, $a \neq a_1$ or $[\![\phi]\!] \cap [\![\phi_1]\!] = \emptyset$.

A *run* of $\mathcal{A}$ on a timed word $(t_1 \cdot a_1) \ldots (t_k \cdot a_k) \in \mathbb{T}\Sigma^*$ is a finite sequence

$$e = (q_0, \nu_0) \xrightarrow{t_1, \theta_1} (q_1, \nu_1) \xrightarrow{t_2, \theta_2} \cdots \xrightarrow{t_k, \theta_k} (q_k, \nu_k) \ ,$$

where $\{q_0, \ldots, q_k\} \subseteq Q$, $\{\nu_0, \ldots, \nu_k\} \subseteq \mathbb{R}_{\geq 0}$ and for each $i \in \{1, \ldots, k\}$ the following hold: $\theta_i \in T$ and θ_i is of the form $(q_{i-1}, q_i, a_i, \phi_i, r_i)$, $\nu_{i-1} + t_i \in [\![\phi_i]\!]$, and $\nu_i = r_i(\nu_{i-1} + t_i)$. Therefore if $r_i = 0$, we have $\nu_i = 0$ and if $r_i = 1$ we have $\nu_i = \nu_{i-1} + t_i$. A pair $(q, \nu) \in Q \times \mathbb{R}_{\geq 0}$ like the ones occurring in the run e is called a *configuration* of $\mathcal{A}$ and the configuration $(q_I, 0)$ is called *initial*. The run e is deemed *accepting* if $q_k \in F$.

For $w \in \mathbb{T}\Sigma^*$ we write $(q, \nu) \rightsquigarrow^w (q', \nu')$ if there is a run of $\mathcal{A}$ on w from (q, ν) to (q', ν'). Observe that if $\mathcal{A}$ is deterministic then for every timed word w there is at most one run on w starting from the initial configuration. We say that $\mathcal{A}$ is complete if every word admits a run. In the rest, we will always assume, without loss of generality, that our timed automata are complete. Finally, given a configuration (q, x) define $\mathcal{L}(q, x) = \{w \in \mathbb{T}\Sigma^* \mid (q, x) \rightsquigarrow^w (p, \nu), p \in F, \nu \in \mathbb{R}_{\geq 0}\}$, hence define $L(\mathcal{A})$ as $\mathcal{L}(q_I, 0)$.

Equivalence relation. A relation $\sim \subseteq S \times S$ on a set S is an *equivalence* if it is reflexive (i.e. $x \sim x$), transitive (i.e. $x \sim y \wedge y \sim z \implies x \sim z$) and symmetric (i.e. $x \sim y \implies y \sim x$). The equivalence class of $s \in S$ w.r.t. $\sim$ is the subset $[s]_\sim = \{s' \in S \mid s \sim s'\}$. A *representative* of the class $[s]_\sim$ is any element $s' \in [s]_\sim$. Given a subset D of S we define $[D]_\sim = \{[d]_\sim \mid d \in D\}$. We say that $\sim$ has *finite index* when $[S]_\sim$ is a finite set. An important notion in the analysis of timed automata is the *region equivalence* which we recall next for one-clock timed automata.

Region equivalence. Given a constant $K \in \mathbb{N}$ define the equivalence $\equiv^K \subseteq \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ by

$$x \equiv^K y \iff \big(\lfloor x \rfloor = \lfloor y \rfloor \wedge (\{x\} = 0 \Leftrightarrow \{y\} = 0)\big) \vee (x > K \wedge y > K) \ ,$$

where given $x \in \mathbb{R}_{\geq 0}$ we denote by $\lfloor x \rfloor$ its integral part and by $\{x\}$ its fractional part.

3 Languages with Integer Resets

We are interested in timed languages recognized by IRTAs. It is known that IRTAs can be converted to 1-clock deterministic IRTAs [15]. The key idea is that in any reachable valuation of an IRTA, all clocks have the same fractional value. Therefore, the integral values of all clocks can be encoded inside the control state, and the fractional values can be read from a single clock. In the sequel, we simply define IRTAs with a single clock, due to the equi-expressivity.

We define the class of one-clock integer-reset timed automata (1-IRTA) where transitions reset the clock provided its value is an integer. Formally, we say that a 1-TA $\mathcal{A} = (Q, q_I, T, F)$ is a 1-IRTA when for every resetting transition $(q, q', a, \phi, 0) \in T$ the clock constraint ϕ is of the form $x = m$, or, equivalently, $[\![\phi]\!] \in \mathbb{N}$. A deterministic 1-IRTA is called a 1-IRDTA.

▶ **Example 3.1.** The one-clock timed automata in Figures 1, 2 and 3 are all 1-IRTAs.

The definition of a run of a 1-IRTA on a timed word simply follows that of 1-TA. However, for 1-IRTA, we can identify positions in input timed words where resets can potentially happen. The next definition makes this idea precise.

▶ **Definition 3.2.** *Given $d = t_1 \cdots t_n \in \mathbb{T}$ and a $K \in \mathbb{N}$, we define the longest sequence of indices $s_d = \{0 = i_0 < i_1 < \cdots < i_p \leq n\}$ such that for every $j \in \{0, \ldots, p-1\}$ the value $\sum_{i=(i_j)+1}^{i_{(j+1)}} t_i$ is an integer between 0 and K. We refer to the set of positions of the sequence s_d as the* integral positions *of d. Note that s_d is never empty since it always contains 0. Next, define*

$$c^K(d) = \sum_{i=(i_p)+1}^{n} t_i \ .$$

The definitions of integral positions and the function c^K apply equally to timed words by taking the timestamp of the timed word.

Notice that the sequence s_d depends on the constant K (we do not explicitly add K to the notation for simplicity, as in our later usage, K will be clear from the context). Note also that when $i_p = n$ then $c^K(d) = 0$, otherwise $c^K(d)$ can be any real value except an integer value between 0 and K, i.e. $c^K(d) \in \mathbb{R}_{\geq 0} \setminus \{0, \ldots, K\}$.

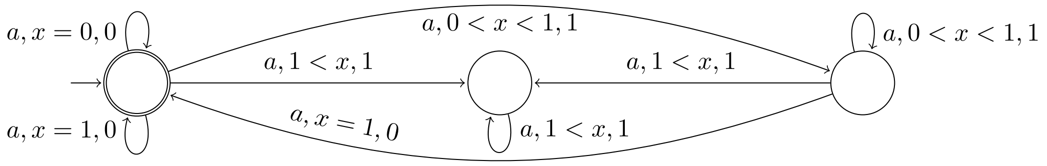


Figure 3 A strict 1-IRTA with alphabet $\Sigma = \{a\}$ accepting $M = \{u \in \mathbb{T}\Sigma^* \mid c^1(u) = 0\}$.

▶ **Example 3.3.** For $K = 1$ and $u = (0.2 \cdot a)(0.8 \cdot a)(0.2 \cdot a)$ we have $s_u = \{0 < 2\}$ and $c^1(u) = 0.2$. For $K = 1$ and $u' = (1.2 \cdot a)(0.8 \cdot a)(0.2 \cdot a)$ we have $s_{u'} = \{0\}$ and $c^1(u') = 2.2$. For $K = 2$, we have $s_{u'} = \{0 < 2\}$ and $c^2(u') = 0.2$.

Consider a run of a 1-IRTA on a word $u = (t_1 \cdot a_1) \cdots (t_n \cdot a_n) \in \mathbb{T}\Sigma^*$ and factor it according to $s_u = \{0 = i_0 < i_1 < \cdots < i_p \leq n\}$:

$$(q_{i_0}, \nu_{i_0}) \xrightarrow{t_{(i_0)+1},\theta_{(i_0)+1}\cdots t_{i_1},\theta_{i_1}} (q_{i_1}, \nu_{i_1}) \xrightarrow{t_{(i_1)+1},\theta_{(i_1)+1}\cdots t_{i_2},\theta_{i_2}} (q_{i_2}\nu_{i_2}) \to \cdots$$
$$\to (q_{i_p}, \nu_{i_p}) \xrightarrow{t_{(i_p)+1},\theta_{(i_p)+1}\cdots t_n,\theta_n} (q_n, \nu_n)$$

At each position i_j with $j \in \{0, \ldots, p\}$, $\nu_{i_j} \in \mathbb{N}$ and, moreover, $\nu_{i_j} = 0$ when $r_{i_j} = 0$.

In Definition 3.2 we identified the integral positions at which a 1-IRTA *could potentially* reset the clock. In the following, we recall a subclass of 1-IRTAs called strict 1-IRTAs [5] where every transition with an equality guard ϕ (ϕ is of the form $x = m$ or, equivalently, $[\![\phi]\!] \in \mathbb{N}$) *must* reset the clock. This feature, along with a special requirement on guards forces a reset on every position given by s_u for a word u.

Strict 1-IRTA

A 1-IRTA is said to be *strict* if there exists $K \in \mathbb{N}$ such that for each of its transitions (q, q', a, ϕ, r) the following holds:

1. the clock constraint of the guard ϕ is either $x = m$, $m < x \wedge x < m + 1$, or $K < x$,
2. the clock constraint of the guard ϕ is an equality iff $r = 0$.

▶ **Example 3.4.** The 1-IRTA in Figures 2 and 3 are strict 1-IRTAs whereas the one in Figure 1 is not strict since the transition $q_I \xrightarrow{a,x=1,1} q$ does not reset the clock. To make it strict while accepting the same language, we replace the transitions of guard $x = 1$ by resetting transitions of guard $x = 0$ and, split the transition $q_I \xrightarrow{a,x<1,1} q_I$ into $q_I \xrightarrow{a,0<x<1,1} q_I$ and $q_I \xrightarrow{a,x=0,0} q_I$.

A run of a strict 1-IRTA on a word u can be factored similarly as explained for a general 1-IRTA, however now, every r_{i_j} will be a reset transition: notice that we require each transition to be guarded using constraints of a special form, either $x = m$ or $m < x < m + 1$ or $K < x$; therefore, the transition reading (t_{i_1}, a_{i_1}) will necessarily have an equality guard $x = m$ forcing a reset, similarly at i_2 and so on. Therefore, the sequence s_u identifies the exact reset points in the word, no matter which strict 1-IRTA reads it. The quantity $c^K(u)$ gives the value of the clock on reading u by any strict 1-IRTA. This *input-determinism* is a fundamental property of strict 1-IRTAs that helps in the Myhill-Nerode characterization that we present in the later sections.

The question now is how expressive are strict 1-IRTAs. As shown by the proposition below, every language definable by a 1-IRTA is also definable by a strict 1-IRTA. Therefore, we could simply consider strict 1-IRTAs instead of 1-IRTAs. Even though a proof of this equi-expressivity theorem is known [5] we provide one in the full version [9].

▶ **Proposition 3.5** (see also Theorem 1 [5]). *A language accepted by a (deterministic)* 1*-IRTA is also accepted by a (deterministic) strict* 1*-IRTA with no greater constant in guards.*

4 From Equivalences to Automata and Back

We start the study of equivalences for languages accepted by integer reset automata. Proposition 3.5 says that every IRTA language can be recognized by a strict 1-IRDTAs. There are two advantages of strict 1-IRDTAs: there is a single clock; and the value of the clock on reading the word is simply determined by the word and not by the automaton that is reading it. This motivates us to restrict our attention to equivalences that make use of the quantity $c^K(u)$, and from which one can construct a strict 1-IRDTA with states as the equivalence classes. In order to be able to do so, we need a good notion of monotonicity (Challenge 3 of the Introduction).

Intuitively, the equivalence should satisfy two conditions whenever u is equivalent to v: (1) when u can elapse time and satisfy a guard, v should be able to elapse some time and satisfy the same guard, and (2) all one step extensions of u and v, say $u' = u(t \cdot a)$ and $v' = v(t' \cdot a)$ such that the clock values $c^K(u')$ and $c^K(v')$ satisfy the same set of guards w.r.t constant K, should be made equivalent. Each guard in a strict 1-IRDTA represents a K-region. All these remarks lead to the following definition of K-monotonicity.

▶ **Definition 4.1** (L-preserving, K-monotonic). *An equivalence* $\approx\ \subseteq \mathbb{T}\Sigma^* \times \mathbb{T}\Sigma^*$ *is* L-preserving *when* $u \approx v \implies (u \in L \iff v \in L)$*. Given a constant* $K \in \mathbb{N}$*,* $\approx\ \subseteq \mathbb{T}\Sigma^* \times \mathbb{T}\Sigma^*$ *is* K-monotonic *when* $u \approx v$ *implies:*

(a) $c^K(u) \equiv^K c^K(v)$*, and*

(b) $\forall a \in \Sigma$*,* $\forall t, t' \in \mathbb{R}_{\geq 0} : \ c^K(u) + t \equiv^K c^K(v) + t' \implies u(t \cdot a) \approx v(t' \cdot a)$*.*

From a K-monotonic equivalence, we can construct a strict 1-IRDTA whose states are the equivalence classes. Here is additional notation. For a number $t \in \mathbb{R}_{\geq 0}$, we define a clock constraint $\phi(\ [t]_{\equiv^K}\ )$ as:

$$\phi(\ [t]_{\equiv^K}\ ) = \begin{cases} x = t & \text{if } t \leq K \wedge t \in \mathbb{N} \ , \\ \lfloor t \rfloor < x \wedge x < \lfloor t \rfloor + 1 & \text{if } t \leq K \wedge t \notin \mathbb{N} \ , \\ K < x & \text{if } K < t \ . \end{cases}$$

▶ **Definition 4.2** (From equivalence $\approx$ to strict 1-IRDTA $\mathcal{A}_\approx$). *Let* $L \subseteq T\Sigma^*$*,* $K \geq 0$*, and* $\approx$ *a* K*-monotonic,* L*-preserving equivalence with finite index. The strict 1-IRDTA* $\mathcal{A}_\approx$ *has states* $\{[u]_\approx \mid u \in T\Sigma^*\}$*. The initial state is* $[\epsilon]_\approx$*. Final states are* $\{[u]_\approx \mid u \in L\}$*. Between two states* $[u]_\approx$ *and* $[v]_\approx$ *there is a transition* $([u]_\approx, [v]_\approx, a, g, s)$ *if there exists* $t \in \mathbb{R}_{\geq 0}$ *such that:*

- $u(t \cdot a) \approx v$*, and*
- $g = \phi(\ [c^K(u) + t]_{\equiv^K}\ )$*, and*
- $s = 0$ *if* $c^K(u) + t \in \{0, 1, \ldots, K\}$ *and* $s = 1$ *otherwise.*

We now explain why the above definition does not depend on the representative picked from an equivalence class. Suppose $([u]_\approx, [v]_\approx, a, g, s)$ is a transition. Let $t \in \mathbb{R}_{\geq 0}$ be a value which witnesses the transition, that is, it satisfies the conditions of the above definition.

Pick another word u' equivalent to u, that is, $u \approx u'$. By Definition 4.1 (a), we have $c^K(u) \equiv^K c^K(u')$. Therefore, there exists t' such that $c^K(u) + t \equiv^K c^K(u') + t'$. Moreover by (b), $u'(t' \cdot a) \approx u(t \cdot a)$, and hence $u'(t' \cdot a) \approx v$. Therefore, we observe that even if we had chosen u' instead of u, we get a witness t' for the same transition $([u]_\approx, [v]_\approx, a, g, s)$.

▶ **Lemma 4.3.** *Let $\approx$ be an L-preserving, K-monotonic equivalence with finite index. Then $\mathcal{L}(\mathcal{A}_\approx) = L$.*

Proof. By induction on the length of the timed words we show that for every $u \in \mathbb{T}\Sigma^*$, $([\epsilon]_\approx, 0) \rightsquigarrow^u ([u]_\approx, c^K(u))$. Let $u(t \cdot a) \in \mathbb{T}\Sigma^*$. Assume $([\epsilon]_\approx, 0) \rightsquigarrow^u ([u]_\approx, c^K(u))$. By definition of $\mathcal{A}_\approx$, there is a transition $([u]_\approx, [u(t \cdot a)]_\approx, a, g, s)$ such that $g = \phi([c^K(u) + t]_{\equiv^K})$ and, $s = 0$ if $c^K(u) + t \in \{0, 1, \ldots, K\}$ and $s = 1$ otherwise. Since $c^K(u(t \cdot a)) = (c^K(u) + t)s$ we deduce that $([\epsilon]_\approx, 0) \rightsquigarrow^u ([u]_\approx, c^K(u)) \rightsquigarrow^{(t \cdot a)} ([u(t \cdot a)]_\approx, c^K(u(t \cdot a)))$. Finally, since $\approx$ is L-preserving, a word is in L iff $\mathcal{A}_\approx$ accepts it. ◀

We now look at the reverse question of obtaining a monotonic equivalence from an automaton. Given a complete strict 1-IRDTA $\mathcal{B}$, we define an equivalence $\approx_\mathcal{B}$ as $u \approx_\mathcal{B} v$ if $\mathcal{B}$ reaches the same (control) state on reading u and v from its initial configuration. If K is the maximum constant appearing in $\mathcal{B}$, it is tempting to think that $\approx_\mathcal{B}$ is a K-monotonic equivalence. However $\approx_\mathcal{B}$ need not satisfy condition **(a)** of Definition 4.1. For instance, consider a strict 1-IRDTA $\mathcal{B}$ which has two self-looping transitions in its initial state: $q \xrightarrow{a, x=0, 0} q$ and $q \xrightarrow{a, 0<x<1, 1} q$. Observe that $(0 \cdot a) \approx_\mathcal{B} (0.5 \cdot a)$, but $c^1((0 \cdot a)) \neq c^1((0.5 \cdot a))$. Therefore, the state-based equivalence $\approx_\mathcal{B}$ needs to be further refined in order to satisfy monotonicity. This leads us to define an equivalence $\approx_\mathcal{B}^K$ as: $u \approx_\mathcal{B}^K v$ if $u \approx_\mathcal{B} v$ and $c^K(u) = c^K(v)$.

▶ **Lemma 4.4.** *Let $\mathcal{B}$ be a complete strict 1-IRDTA with maximum constant K. The equivalence $\approx_\mathcal{B}^K$ is $\mathcal{L}(\mathcal{B})$-preserving, K-monotonic and has finite index.*

Proof. The equivalence $\approx_\mathcal{B}^K$ is $\mathcal{L}(\mathcal{B})$-preserving because equivalent words reach the same state in $\mathcal{B}$, thus either both are accepted or both are rejected. It has finite index because the number of its equivalence classes is bounded by the number of states of $\mathcal{B}$ multiplied by the number of K regions. Condition **(a)** and **(b)** of Def. 4.1 respectively hold by definition of $\approx_\mathcal{B}^K$ and since $\mathcal{B}$ is deterministic. ◀

The goal of the section was to go from equivalences to automata and back. Lemma 4.3 talks about equivalence-to-automata. For the automata-to-equivalence, we needed to strengthen the state-based equivalence with the region equivalence. A close look at $\mathcal{A}_\approx$ of Definition 4.2 reveals whenever $u \approx v$, we also have $c^K(u) \equiv^K c^K(v)$. So, in the equivalence-to-automata, we get an automaton satisfying a stronger property. This motivates us to explicitly highlight a class of strict 1-IRDTAs where each state can be associated with a unique region. For this class, we will be able to go from equivalence-to-automata-and-back directly.

▶ **Definition 4.5** (K-acceptor)**.** *A K-acceptor $\mathcal{B}$ is a complete strict 1-IRDTA with maximum constant smaller than or equal to K such that for every $u, v \in T\Sigma^*$, $u \approx_\mathcal{B} v$ implies $c^K(u) \equiv^K c^K(v)$. Hence every state q of $\mathcal{B}$ can be associated to a unique K-region, denoted, region(q), i.e., whenever $(q_I, 0) \rightsquigarrow^w (q, c^K(w))$ then $c^K(w) \in$ region(q).*

Every strict 1-IRDTA $\mathcal{B}$ with maximum constant K can be converted into a K-acceptor by starting with the equivalence $\approx_\mathcal{B}^K$ and building $\mathcal{A}_{\approx_\mathcal{B}^K}$. Furthermore K-monotonic equivalences characterize K-acceptors (Lemmas 4.3 and 4.4). Therefore, for the rest of the document, we will restrict our focus to K-acceptors. As a next task, we look for the coarsest possible K-monotonic equivalence for a language. This will give a minimal K-acceptor for the language, which we deem to be the canonical integer reset timed automaton, with maximum constant K, for the language.

5 A Nerode-style Equivalence

In the previous section, we have established generic conditions required from an equivalence to construct a K-acceptor from it. In this section, we will present a concrete such equivalence: given a language L definable by a 1-IRTA with a maximum constant $K \in \mathbb{N}$ we define a *syntactic equivalence* $\approx^{L,K} \subseteq \mathbb{T}\Sigma^* \times \mathbb{T}\Sigma^*$. "Syntactic" means this equivalence is independent of a specific representation of L. We then show that $\approx^{L,K}$ is the coarsest L-preserving and K-monotonic equivalence.

The idea for defining $\approx^{L,K}$ is to identify two words u and u' whenever $c^K(u) \equiv^K c^K(u')$ and the residuals $u^{-1}L$ and $u'^{-1}L$ coincide modulo some rescaling w.r.t. $c^K(u)$ and $c^K(u')$. We start with examples to give some intuition behind the rescaling function.

Examples. Consider an automaton segment $q_0 \xrightarrow{a,0<x<1,1} q_1 \xrightarrow{b,x=1,0} q_2 \xrightarrow{c,0<x<1,1} q_3$, with q_3 being an accepting state. The words $u = (0.2 \cdot a)$ and $v = (0.6 \cdot a)$ both go to state q_1. Let us assume that all other transitions go to a sink state, and also that the maximum constant $K = 1$. The language L accepted is $\{(t_1 \cdot a)(t_2 \cdot b)(t_3 \cdot c)\}$ where $0 < t_1 < 1$, $t_1 + t_2 = 1$ and $0 < t_3 < 1$. Moreover, $u^{-1}L = \{(0.8 \cdot b)(t_3 \cdot c) \mid 0 < t_3 < 1\}$ and $v^{-1}L = \{(0.4 \cdot b)(t_3 \cdot c) \mid 0 < t_3 < 1\}$. Here we want to somehow "equate" the residual languages $u^{-1}L$ and $v^{-1}L$. The idea is to define a bijection between these two sets $u^{-1}L$ and $v^{-1}L$. In this case, the bijection maps $(0.8 \cdot b)(t_3 \cdot c)$ to $(0.4 \cdot b)(t_3 \cdot c)$ for every t_3. Observe that given u, v the bijection depends on the values $c^K(u)$ and $c^K(v)$, which in this example are 0.2 and 0.6 respectively.

Here is another example. Let u_1, v_1 be words with $c^K(u_1) = 0.2$ and $c^K(v_1) = 0.6$. Let $u_1^{-1}L = \{(t_1 \cdot a)(t_2 \cdot b)(t_3 \cdot c)(t_4 \cdot d) \mid c^K(u_1) + t_1 + t_2 + t_3 = 1\}$ and $v_1^{-1}L = \{(t_1' \cdot a)(t_2' \cdot b)(t_3' \cdot c)(t_4' \cdot d) \mid c^K(v_1) + t_1' + t_2' + t_3' = 1\}$. The bijection in this case is more complicated than the previous example. The idea is to first start with $c^K(u_1) = 0.2$, $c^K(v_1) = 0.6$ and consider a bijection f of the open unit interval $(0, 1)$ that maps the intervals $(0, 0.8]$ and $(0.8, 1)$ to $(0, 0.4]$ and $(0.4, 1)$ respectively. This bijection is essentially a rescaling of the intervals $(0, 1 - c^K(u_1)]$ and $(1 - c^K(u_1), 1)$ into the intervals $(0, 1 - c^K(v_1)])$ and $(1 - c^K(v_1), 1)$. We now pick the first letters in the residual languages $u_1^{-1}L$ and $v^{-1}L$ and create a mapping: $(t_1 \cdot a) \mapsto (f(t_1) \cdot a)$. Now we consider $c^K(u_1(t_1 \cdot a))$ and $c^K(v_1(f(t_1) \cdot a))$ in the place of $c^K(u_1), c^K(v_1)$, and continue the mapping process one letter at a time.

Rescaling function. We will now formalize this idea. We will start with bijections of the open unit interval.

Let $\lambda, \lambda' \in (0, 1)$ be arbitrary real values. Define a bijection $f_{\lambda \to \lambda'} : (0, 1) \to (0, 1)$ that scales the $(0, \lambda]$ interval to $(0, \lambda']$ and the $(\lambda, 1)$ interval to $(\lambda', 1)$:

$$f_{\lambda \to \lambda'}(t) = \begin{cases} \left(\dfrac{\lambda'}{\lambda}\right) t & \text{for } 0 < t \leq \lambda \\ \lambda' + \dfrac{(1 - \lambda')}{(1 - \lambda)}(t - \lambda) & \text{for } \lambda < t < 1 \end{cases}$$

Now, consider $x, x' \in \mathbb{R}_{\geq 0}$ such that $x \equiv^K x'$. We define a *length-preserving* bijection $\tau_{x \to x'} : \mathbb{T} \to \mathbb{T}$ inductively as follows: for the empty sequence ϵ, we define $\tau_{x\to x'}(\epsilon) = \epsilon$; for a timestamp $d \in \mathbb{T}$ and a $t \in \mathbb{R}_{\geq 0}$, $\tau_{x\to x'}(dt) = d't'$ where $d' = \tau_{x \to x'}(d)$ and t' is obtained as follows: let $y = c^K(xd)$ and $y' = c^K(x'd')$. If $y, y' \in \mathbb{N}$ or $y, y' > K$, then define $t' = t$. Else, define $\lfloor t' \rfloor = \lfloor t \rfloor$ and $\{t'\} = f_{(1-\{y\})\to(1-\{y'\})}(\{t\})$.

Here is an additional notation, before we describe some properties of the rescaling function. For $x_1, x_2, x_3 \in \mathbb{R}_{\geq 0}$ with $x_1 \equiv^K x_2 \equiv^K x_3$, we denote the composed function $(\tau_{x_2 \to x_3}) \circ (\tau_{x_1 \to x_2})$ as $\tau_{x_1\to x_2 \to x_3}$. So, $\tau_{x_1\to x_2\to x_3}(t) = \tau_{x_2\to x_3}(\tau_{x_1\to x_2}(t))$.

▶ **Lemma 5.1.** *The bijection $\tau_{x\to x'}$ satisfies the following properties:*

1. *for an arbitrary timestamp $t_1 t_2 \dots t_n \in \mathbb{T}$, if $\tau_{x\to x'}(t_1t_2\dots t_n) = t_1't_2'\dots t_n'$ then, we have $c^K(x + t_1 + \dots + t_{n-1}) + t_n \equiv^K c^K(x' + t_1' + \dots + t_{n-1}') + t_n'$,*
2. *$\tau_{x\to x'}^{-1}$ is identical to $\tau_{x'\to x}$,*
3. *$\tau_{x_1\to x_2\to x_3}$ is identical to $\tau_{x_1\to x_3}$.*

The rescaling function $\tau_{x\to x'}$ can be naturally extended to timed words: $\tau_{x\to x'}((t_1 \cdot a_1)(t_2 \cdot a_2)\dots(t_n \cdot a_n)) = (t_1' \cdot a_1)(t_2' \cdot a_2)\dots(t_n' \cdot a_n)$ where $t_1't_2'\dots t_n' = \tau_{x\to x'}(t_1t_2\dots t_n)$. The next observation follows from Property 1. of Lemma 5.1.

▶ **Lemma 5.2.** *Let $\mathcal{B}$ be a K-acceptor, and q a control state of $\mathcal{B}$. Let $x, x' \in \mathbb{R}_{\geq 0}$ such that $x \equiv^K x'$. Then, for every timed word w: $w \in \mathcal{L}(q, x)$ iff $\tau_{x\to x'}(w) \in \mathcal{L}(q, x')$.*

Syntactic equivalence. For timed words $u, v \in T\Sigma^*$ such that $c^K(u) \equiv^K c^K(v)$, we write $\tau_{u\to v}$ for the bijection $\tau_{c^K(u)\to c^K(v)}$. We now present the main equivalence.

▶ **Definition 5.3** (Equivalence $\approx^{L,K}$). *Let L be a timed language and K a natural number. We say $u \approx^{L,K} v$ if $c^K(u) \equiv^K c^K(v)$ and $\tau_{u\to v}(u^{-1}L) = v^{-1}L$.*

Note that the equivalence $\approx^{L,K}$ is L-preserving. Assume $u \approx^{L,K} v$. We have $u \in L \iff \epsilon \in u^{-1}L$ and $\epsilon \in u^{-1}L \iff \epsilon \in v^{-1}L$ since $\tau_{u\to v}(\epsilon) = \epsilon$ by definition. Thus, $u \in L \iff v \in L$.

▶ **Proposition 5.4.** *When $L \subseteq \mathbb{T}\Sigma^*$ is definable by a K-acceptor, then $\approx^{L,K}$ has finite index and is K-monotonic. Moreover, $\approx^{L,K}$ is the coarsest K-monotonic and L-preserving equivalence.*

Proof. We start by showing that $\approx^{L,K}$ is K-monotonic. Condition **(a)** of Definition 4.1 holds by definition. We move to **(b)**. Let $t_u, t_v \in \mathbb{R}_{\geq 0}$ s.t. $c^K(u) + t_u \equiv^K c^K(v) + t_v$. Let:

$$u_1 = u(t_u \cdot a) \quad v_1 = v(t_v \cdot a)$$

$$\text{To show:} \quad \tau_{u_1 \to v_1}(u_1^{-1}L) = v_1^{-1}L \tag{1}$$

Let $t_v' = \tau_{u\to v}(t_u)$ and $v_2 = v(t_v' \cdot a)$. We will prove 1 using these intermediate claims:

▷ Claim 5.5. $\tau_{u_1\to v_2}(u_1^{-1}L) = v_2^{-1}L$

▷ Claim 5.6. $\tau_{v_2\to v_1}(v_2^{-1}L) = v_1^{-1}L$

Hence: $\tau_{u_1\to v_2\to v_1}(u_1^{-1}L) = v_1^{-1}L$. By Lemma 5.1, we conclude $\tau_{u_1\to v_1}(u_1^{-1}L) = v_1^{-1}L$.

Proof of Claim 5.5. Let $w \in T\Sigma^*$. By definition of u_1, we have:

$$w \in u_1^{-1}L \quad \text{iff} \quad (t_u \cdot a)w \in u^{-1}L \tag{2}$$

Since $u \approx^{L,K} v$, we know that $\tau_{u \to v}(u^{-1}L) = v^{-1}L$. Therefore:

$$(t_u \cdot a)w \in u^{-1}L \quad \text{iff} \quad \tau_{u \to v}((t_u \cdot a)w) \in v^{-1}L \tag{3}$$

By the way we have constructed the rescaling function, we have

$$\tau_{u \to v}((t_u \cdot a)w) = (t'_v \cdot a)\tau_{u_1 \to v_2}(w) \tag{4}$$

Finally, by definition of v_2:

$$(t'_v \cdot a)\tau_{u_1 \to v_2}(w) \in v^{-1}L \quad \text{iff} \quad \tau_{u_1 \to v_2}(w) \in v_2^{-1}L \tag{5}$$

From (2), (3), (4) and (5), we conclude $w \in u_1^{-1}L$ iff $\tau_{u_1 \to v_2}(w) \in v_1^{-1}L$ for an arbitrary timed word w. This proves the claim. ◁

Proof of Claim 5.6. Let $\mathcal{B}$ be a K-acceptor recognizing L, and let q be the control state reached by $\mathcal{B}$ on reading word v. We first claim that:

$$c^K(v) + t_v \equiv^K c^K(v) + t'_v \tag{6}$$

This is because, by Lemma 5.1, we have $c^K(u) + t_u \equiv^K c^K(v) + t'_v$, and by assumption, we have $c^K(u) + t_u \equiv^K c^K(v) + t_v$. Since $\equiv^K$ is transitive, (6) follows.

The observation made in (6) implies on elapsing t_v or t'_v from v, the same outgoing transition is enabled, as $\mathcal{B}$ is deterministic. Therefore $v_1 = v(t_v \cdot a)$ and $v_2 = v(t'_v \cdot a)$ reach the same control state q'. Hence, (6) can be read as $c^K(v_1) \equiv^K c^K(v_2)$. By Lemma 5.2, for any timed word w, we have $w \in v_2^{-1}L$ iff $\tau_{v_2 \to v_1}(w) \in v_1^{-1}L$. The claim follows. ◁

We will now show that $\approx^{L,K}$ has finite index and is also the coarsest K-monotonic, L-preserving equivalence. Let $\mathcal{B}$ be a K-acceptor for L. Recall that $u \approx^{\mathcal{B}} v$ if $\mathcal{B}$ reaches the same control state on reading u and v. We will show:

$$u \approx^{\mathcal{B}} v \quad \text{implies} \quad u \approx^{L,K} v \tag{7}$$

This will immediately prove that $\approx^{L,K}$ has finite index. Secondly, for any K-monotonic, L-preserving equivalence $\approx$, we can build a K-acceptor $\mathcal{A}_{\approx}$ (Lemma 4.3) whose equivalence is identical to $\approx$. Thus, (7) also shows that $\approx^{L,K}$ is the coarsest such equivalence.

Proof of (7) is as follows. Let q be the control state reached by u and v in $\mathcal{B}$, and let $x = c^K(u), x' = c^K(v)$. Since $\mathcal{B}$ is a K-acceptor, we also have $x \equiv^K x'$. From Lemma 5.2, $\tau_{x \to x'}(\mathcal{L}(q, x)) = \mathcal{L}(q, x')$. Hence, we deduce: $\tau_{u \to v}(u^{-1}L) = v^{-1}L$. This proves $u \approx^{L,K} v$. ◀

The above proposition leads to the following Myhill-Nerode style characterization for timed languages recognized by IRTA.

▶ **Theorem 5.7.**

(a) *$L \subseteq \mathbb{T}\Sigma^*$ is a language definable by an IRTA if and only if there is a constant $K \in \mathbb{N}$ such that $\approx^{L,K}$ is K-monotonic and has finite index.*

(b) *$\approx^{L,K}$ is coarser than any K-monotonic and L-preserving equivalence.*

The conversion from a general IRTA to a strict 1-IRTA does not increase the constant. Similarly, a strict 1-IRTA with maximum constant K can be converted to a K-acceptor. Therefore, the overall conversion from an IRTA to an acceptor preserves the constant. From the second part of Theorem 5.7, we deduce that for a language L that is recognized by an

IRTA with constant K, the K-acceptor $\mathcal{A}_{\approx^{L,K}}$ built from the equivalence $\approx^{L,K}$ has the least number of states among all K-acceptors recognizing L. It is therefore a minimal automaton among all K-acceptors for L. We now present some examples that apply this Myhill-Nerode characterization.

▶ **Example 5.8.** Consider the language $L = \{(x \cdot a) \mid x \in \mathbb{N}\}$. The right-and side of Theorem 5.7 **(a)** does not hold. Indeed, there is no $K \in \mathbb{N}$ such that $\approx^{L,K}$ verifies the K-monotonicity **(b)**: For every $K \in \mathbb{N}$ we have $c^K(\epsilon) + K + 1 \equiv^K c^K(\epsilon) + K + 1.1$. Since $(K+1 \cdot a) \in L$ and $(K+1.1 \cdot a) \notin L$, by L-preservation, $(K+1 \cdot a) \not\approx^{L,K} (K+1.1 \cdot a)$. By Theorem 5.7, L is not 1-IRTA definable.

▶ **Example 5.9.** Consider the language $L = \{(x \cdot a)(1 \cdot b) \mid 0 < x < 1\}$. This language is accepted by a 1-TA that reads a on a guard $0 < x < 1$, resets the clock x and reads b at $x = 1$. This is clearly not an IRTA. We will once again see that K-monotonicity holds for no K. For $u = (0.2 \cdot a)$, $c^K(u) + 1 \equiv^K c^K(u) + 1.1$ holds for every constant $K \in \mathbb{N}$. Since $u(1 \cdot b) \in L$ and $u(1.1 \cdot b) \notin L$, we have $u(1 \cdot b) \not\approx^{L,K} u(1.1 \cdot b)$. Thus, there is no K such that $\approx^{L,K}$ verifies K-monotonicity **(b)**, hence L is not 1-IRTA definable by Theorem 5.7.

▶ **Example 5.10.** Consider the language $M = \{u \in \mathbb{T}\Sigma^* \mid c^1(u) = 0\}$ with alphabet $\Sigma = \{a\}$ which has the following residual languages. For $u \in \mathbb{T}\Sigma^*$,

$$u^{-1}M = M \quad \text{when } c^1(u) = 0, \quad u^{-1}M = \{v \in \mathbb{T}\Sigma^* \mid \sigma(uv) = 1\} \quad \text{when } 0 < c^1(u) < 1,$$
$$u^{-1}M = \emptyset \quad \text{when } 1 < c^1(u).$$

The equivalence classes for $\approx^{M,1}$ are the following:

$$[\epsilon]_{\approx^{M,1}} = M, \quad [(\tfrac{3}{2} \cdot a)]_{\approx^{M,1}} = \{u \in \mathbb{T}\Sigma^* \mid 1 < c^1(u)\}, \quad [(\tfrac{1}{2} \cdot a)]_{\approx^{M,1}} = \{u \in \mathbb{T}\Sigma^* \mid 0 < c^1(u) < 1\}.$$

The acceptor $\mathcal{A}_{\approx^{M,1}}$ is depicted in Figure 3 where $[\epsilon]_{\approx^{M,1}}$ is the initial (and final) state, $[(\frac{3}{2} \cdot a)]_{\approx^{M,1}}$ the sink state, and $[(\frac{1}{2} \cdot a)]_{\approx^{M,1}}$ the rightmost state.

▶ **Example 5.11.** Consider the language $L = \{(0 \cdot a)^n (0 \cdot b)^n \mid n \in \mathbb{N}\}$. There is no K such that $\approx^{L,K}$ has finite index, although $\approx^{L,0}$ is 0-monotonic: for $u \approx^{L,0} v$ with $0 < c^0(u) \equiv^0 c^0(v)$, no extension of u or v belongs to L and hence monotonicity **(b)** holds; pick u and v with $c^0(u) = c^0(v) = 0$, and let t, t' such that $c^0(u) + t \equiv^0 c^0(v) + t'$. If $t = 0$ then $t' = 0$ and monotonicity holds. Suppose $0 < t, t'$. Then $u(t \cdot \alpha) \notin L$ and $u(t' \cdot \alpha) \notin L$ for any letter α. Once again, **(b)** holds.

We now argue the infinite index, similar to the case of untimed languages. For distinct integers n and m we have $((0 \cdot a)^n)^{-1} L \neq ((0 \cdot a)^m)^{-1} L$. Since $\tau_{0 \to 0}$ is the identity we have $\tau_{0 \to 0}(((0 \cdot a)^n)^{-1} L) \neq ((0 \cdot a)^m)^{-1} L$. Thus, $(0 \cdot a)^n \not\approx^{L,K} (0 \cdot a)^m$.

▶ **Example 5.12.** Consider the language $L = \{u \in \mathbb{T}\Sigma^* \mid t_1 + \cdots + t_n = 1\}$ given in Figure 1. Given a timed word $u = (t_1 \cdot a_1) \ldots (t_n \cdot a_n) \in \mathbb{T}\Sigma^*$ we denote by $\sigma(u)$ the sum $t_1 + \cdots + t_n$ of its timestamps. Also we have that $\sigma(\epsilon) = 0$. The residual languages of L are the following. For $u \in \mathbb{T}\Sigma^*$,

$$u^{-1}L = \emptyset \quad \text{when } 1 < \sigma(u), \quad u^{-1}L = \{v \in \mathbb{T}\Sigma^* \mid \sigma(v) = 0\} \quad \text{when } \sigma(u) = 1,$$
$$u^{-1}L = L \quad \text{when } \sigma(u) = 0, \quad u^{-1}L = \{v \in \mathbb{T}\Sigma^* \mid \sigma(u) + \sigma(v) = 1\} \quad \text{when } 0 < \sigma(u) < 1.$$

The equivalence classes for $\approx^{L,1}$ are:

$$[(1 \cdot a)]_{\approx^{L,1}} = L, \qquad [(1 \cdot a)(\tfrac{1}{2} \cdot a)]_{\approx^{L,1}} = \{u \in \mathbb{T}\Sigma^* \mid 0 < c^1(u) < 1 \wedge 1 < \sigma(u)\},$$
$$[\epsilon]_{\approx^{L,1}} = \{u \in \mathbb{T}\Sigma^* \mid \sigma(u) = 0\}, \qquad [(\tfrac{1}{2} \cdot a)]_{\approx^{L,1}} = \{u \in \mathbb{T}\Sigma^* \mid 0 < c^1(u), \sigma(u) < 1\},$$
$$[(2 \cdot a)]_{\approx^{L,1}} = \{u \in \mathbb{T}\Sigma^* \mid 1 < c^1(u)\}, \qquad [(1 \cdot a)(1 \cdot a)]_{\approx^{L,1}} = \{u \in \mathbb{T}\Sigma^* \mid c^1(u) = 0 \wedge 1 < \sigma(u)\}.$$

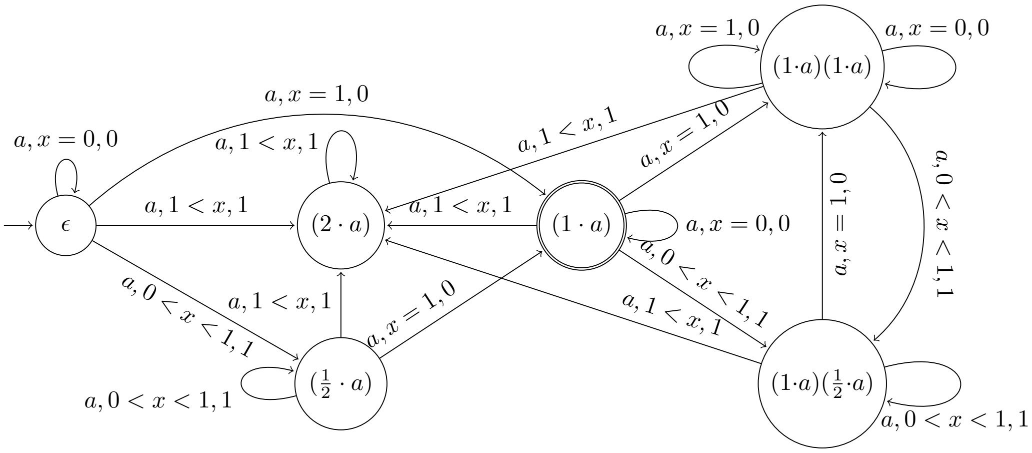


Figure 4 A strict 1-IRDTA $\mathcal{A}_{\approx^{L,1}}$ accepting $L = \{u \in \mathbb{T}\Sigma^* \mid \sigma(u) = 1\}$.

The acceptor $\mathcal{A}_{\approx^{L,1}}$ is depicted in Figure 4 and has six states, whereas the (strict) 1-IRDTA given in Example 3.4 for L only has two.

6 The Canonical Form

Sections 4 and 5 developed the idea of a canonical K-acceptor for a language recognized by an IRTA with constant K. In this section, we will further study this canonical form. We present a crucial property that will help effectively compute the canonical form, and also apply an Angluin-style L^* algorithm.

▶ **Definition 6.1** (Half-integral words). *We call a timed word $(t_1 \cdot a_1)(t_2 \cdot a_2) \dots (t_n \cdot a_n)$ to be a* half-integral *if for each $1 \leq i \leq n$, the fractional part $\{t_i\}$ is either 0 or $\frac{1}{2}$: in other words, each delay is either an integer or a rational with fractional value $\frac{1}{2}$. Also, the empty word ϵ is a half-integral word.*

▶ **Definition 6.2** (Small half-integral words). *Let $K \in \mathbb{N}$. A half-integral word $(t_1 \cdot a_1)(t_2 \cdot a_2) \dots (t_n \cdot a_n)$ is said to be* small *w.r.t K if $t_i < K + 1$ for all $1 \leq i \leq n$.*

For a finite alphabet Σ and $K \in \mathbb{N}$, let $\Sigma_K := \{0, \frac{1}{2}, 1, \dots, K - \frac{1}{2}, K, K + \frac{1}{2}\} \times \Sigma$. Every small integral word is therefore in Σ_K^*. The next lemma is a generalization of the following statement: for every timed word u, there is a small half-integral timed word w such that every K-acceptor reaches the same state on reading both u and w.

▶ **Lemma 6.3.** *Let u_0 be a half-integral word which is small w.r.t. K. For every timed word u, there is a half-integral word w such that $u_0 w$ is small w.r.t K and every K-acceptor reaches the same state on reading $u_0 u$ or $u_0 w$.*

This allows us to identify the canonical equivalence $\approx^{L,K}$ using small integral words.

▶ **Proposition 6.4.** *Let $\mathcal{B}$ be a K-acceptor for a language L. Then, the equivalence $\approx^{\mathcal{B}}$ coincides with $\approx^{L,K}$ iff for all half-integral words $u, v \in (\Sigma_K)^*$: $u \approx^{\mathcal{B}} v$ iff $u \approx^{L,K} v$.*

We make use of Proposition 6.4 to compute the canonical form.

6.1 Computing the Canonical Form

Given a K-acceptor $\mathcal{B}$, we can minimize it using an algorithm which is similar to the standard DFA minimization which proceeds by computing a sequence of equivalence relations on the states.

- Equivalence $\sim^0$: for a pair of states p, q of $\mathcal{B}$ define $p \sim^0 q$ if $\mathit{region}(p) = \mathit{region}(q)$ and either both are accepting states, or both are non-accepting states.
- Suppose we have computed the equivalence $\sim^i$ for some $i \in \mathbb{N}$. For a pair of states p, q, define $p \sim^{i+1} q$ if $p \sim^i q$ and for every letter $(t, a) \in \Sigma_K$, the outgoing transitions $(p, p', a, \phi([t]_{\equiv^K}), s)$ and $(q, q', a, \phi([t]_{\equiv^K}), s)$ in $\mathcal{B}$ satisfy $p' \sim^i q'$.
- Stop when $\sim^{i+1}$ equals $\sim^i$.

The next lemma is a simple consequence of the definition of $\sim^i$ and by an induction on the number of iterations i.

▶ **Lemma 6.5.** *Let p, q be such that $\mathit{region}(p) = \mathit{region}(q)$. Suppose $p \not\sim^i q$. Then there exists a word z of length at most i such that $\delta^*_{\mathcal{B}}(p, z)$ is accepting whereas $\delta^*_{\mathcal{B}}(q, z)$ is not.*

We define the quotient of $\mathcal{B}$ by $\sim^i$ as the K-acceptor whose states are the equivalence classes for $\sim^i$. There is a transition $([q]_{\sim^i}, [p]_{\sim^i}, a, \phi([t]_{\equiv^K}), s)$ if there is $q' \in [q]_{\sim^i}$ and $p' \in [p]_{\sim^i}$ such that $(q', p', a, \phi([t]_{\equiv^K}, s)$ is a transition of $\mathcal{B}$. The initial state is the class of the initial state of $\mathcal{B}$ and the final states are the classes that have a non empty intersection with the set of final states of $\mathcal{B}$. For $i \geq 1$ the quotient of $\mathcal{B}$ by $\sim^i$ is an acceptor for $L(\mathcal{B})$.

Suppose we reach a fixpoint at $m \in \mathbb{N}$. The quotient of $\mathcal{B}$ by $\sim^m$ gives the canonical automaton $\mathcal{A}_{\approx^{L,K}}$. Suppose the quotient does not induce the canonical equivalence. By Proposition 6.4 there are two words $u, v \in (\Sigma_K)^*$ such that u and v go to a different state, but $u \approx^{L,K} v$. Consider the iteration i when the two states were made non-equivalent. There is a small half-integral word z of length i which distinguishes u and v by Lemma 6.5 – a contradiction to $u \approx^{L,K} v$. Let us say $z \in u^{-1}L$, since u, v are half-integral, $\tau_{u \to v}$ is simply the identity function. Since $z \notin v^{-1}L$, we deduce that $\tau_{u \to v}(u^{-1}L) \neq v^{-1}L$, hence $u \not\approx^{L,K} v$.

6.2 Learning the Canonical Form

Our learning algorithm closely follows Angluin's L^* approach, so we assume familiarity with it and provide a brief example of its adaptation to the IRTA setting. Detailed definitions, proof of correctness, and a complete example are provided in the full version [9].

We assume that the Learner is aware of the maximum constant K for the unknown language L. The Learner's goal is to identify the equivalence classes of $\approx^{L,K}$ using small half-integral words, from Σ_K^*. Correspondingly, the rows and columns in an observation table are words in Σ_K^*. In the L^* algorithm, each row of the observation table corresponds to an identified state. Two identical rows correspond to the same state. In order to make a similar conclusion, we add a column to the observation table, that maintains $c^K(u)$ for every string u of a row. There is one detail: once $c^K(u)$ goes beyond K, we want to store it as a single entity $\top$. We define $c^K_\top(u)$ to be equal to $c^K(u)$ when this value is bellow K and equal to $\top$ otherwise.

▶ **Lemma 6.6.** *Let L be a timed language recognized by a K-acceptor, and let $u, v \in (\Sigma_K)^*$. Then $u \approx^{L,K} v$ iff $c^K_\top(u) = c^K_\top(v)$ and for all words $z \in (\Sigma_K)^*$, we have $uz \in L$ iff $vz \in L$.*

An observation table labels its rows and columns with words in $(\Sigma_K)^*$. The row words form a prefix-closed set, and the column words form a suffix-closed set, as in Angluin. Table 1 shows three observation tables. The lower part of these tables includes one-letter extensions of the row words, and their $c^K_\top$ values are shown in the extra column in red.

Table 1 A run of L^* for IRTA.

$\mathcal{T}_0$	ϵ	$c^1_\top(u)$
ϵ	0	0
$(0 \cdot a)$	0	0
$(\frac{1}{2} \cdot a)$	0	$\frac{1}{2}$
$(1 \cdot a)$	1	0
$(1 + \frac{1}{2} \cdot a)$	0	$\top$

$\mathcal{T}_1$	ϵ	$c^1_\top(u)$
ϵ	0	0
$(\frac{1}{2} \cdot a)$	0	$\frac{1}{2}$
$(1 \cdot a)$	1	0
$(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(0 \cdot a)$	0	0
$(\frac{1}{2} \cdot a)(0 \cdot a)$	0	$\frac{1}{2}$
$(\frac{1}{2} \cdot a)(\frac{1}{2} \cdot a)$	1	0
$(\frac{1}{2} \cdot a)(1 \cdot a)$	0	$\top$
$(\frac{1}{2} \cdot a)(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(1 \cdot a)(0 \cdot a)$	1	0
$(1 \cdot a)(\frac{1}{2} \cdot a)$	0	$\frac{1}{2}$
$(1 \cdot a)(1 \cdot a)$	0	0
$(1 \cdot a)(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(1 + \frac{1}{2} \cdot a)(\Sigma_K \cdot a)$	0	$\top$

$\mathcal{T}_2$	ϵ	$c^1_\top(u)$
ϵ	0	0
$(\frac{1}{2} \cdot a)$	0	$\frac{1}{2}$
$(1 \cdot a)$	1	0
$(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(1 \cdot a)(1 \cdot a)$	0	0
$(1 \cdot a)(1 \cdot a)(1 \cdot a)$	0	0
$(0 \cdot a)$	0	0
$(\frac{1}{2} \cdot a)(0 \cdot a)$	0	$\frac{1}{2}$
$(\frac{1}{2} \cdot a)(\frac{1}{2} \cdot a)$	1	0
$(\frac{1}{2} \cdot a)(1 \cdot a)$	0	$\top$
$(\frac{1}{2} \cdot a)(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(1 \cdot a)(0 \cdot a)$	1	0
$(1 \cdot a)(\frac{1}{2} \cdot a)$	0	$\frac{1}{2}$
$(1 \cdot a)(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(1 + \frac{1}{2} \cdot a)(\Sigma_K \cdot a)$	0	$\top$
$(1 \cdot a)(1 \cdot a)(0 \cdot a)$	0	0
$(1 \cdot a)(1 \cdot a)(\frac{1}{2} \cdot a)$	0	$\frac{1}{2}$
$(1 \cdot a)(1 \cdot a)(1 + \frac{1}{2} \cdot a)$	0	$\top$
$(1 \cdot a)(1 \cdot a)(1 \cdot a)(\Sigma_K \cdot a)$	0	..

Suppose the unknown IRTA language is $L = \{u \in \mathbb{T}\Sigma^* \mid \sigma(u) = 1\}$ for $\Sigma = \{a\}$ with a known constant $K = 1$. The learner starts with $\mathcal{T}_0$ containing only the ϵ row which is 0 since $\epsilon \notin L$, and the ϵ column. $\mathcal{T}_0$ is not closed as witnessed by $(1 \cdot a)$, whose row is 1, while the row of ϵ is 0. Additionally, $(\frac{1}{2} \cdot a)$ and $(1 + \frac{1}{2} \cdot a)$ also witness non-closure because their $c^K_\top$ values are non zero. This highlights the difference from the untimed case: row words are distinguished based on their clock values as well.

To obtain a closed table, the learner successively adds the words $(\frac{1}{2} \cdot a)$, $(1 \cdot a)$, $(1 + \frac{1}{2} \cdot a)$, forming $\mathcal{T}_1$. Every two words in $\mathcal{T}_1$ are distinguished either by their row or by their $c^K_\top$ value. Thus, $\mathcal{T}_1$ is consistent: if two words have identical rows and same $c^K_\top$ value then their extensions also satisfy this. Since $\mathcal{T}_1$ is closed and consistent the learner conjectures $\mathcal{A}_{\mathcal{T}_1}$ (see [9] for details), the K-acceptor induced by $\mathcal{T}_1$ (formal definition in [9]). The teacher provides a counterexample, assumed to be $(1 \cdot a)(1 \cdot a)(1 \cdot a)$, which is accepted by $\mathcal{A}_{\mathcal{T}_1}$ but not in L. The learner processes this counterexample and computes $\mathcal{T}_2$, which is closed but not consistent as shown by ϵ and $(1 \cdot a)(1 \cdot a)$ and their extensions by $(1 \cdot a)$. Hence, the learner adds a column for $(1 \cdot a)$ and computes $\mathcal{T}_3$. The process continues similarly. The rest of the run is detailed in the full version [9].

7 Conclusion

We have presented a Myhill-Nerode style characterization for timed languages accepted by timed automata with integer resets. There are three main technical ingredients: (1) the notion of K-monotonicity (Definition 4.1) that helps characterize equivalences on timed words with automata, that we call K-acceptors. This was possible since each word u determines the value $c^K(u)$ of the clock on reading u, in any K-acceptor; (2) the definition of the rescaling function (Section 5) that gives a Nerode-like equivalence, leading to a Myhill-Nerode theorem for IRTA languages, and the canonical equivalence $\approx^{L,K}$; (3) understanding canonical equivalence $\approx^{L,K}$ through half-integral words (Section 6), which are, in some sense, discretized words. This helps us to build and learn the canonical form. We believe these technical ingredients provide insights into understanding the languages recognized by IRTA. Typically, active learning algorithms begin by setting up a canonical form. We have laid the foundation for IRTAs. Future work lies in adapting these foundations for better learning algorithms.

References

1 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.

2 Jie An, Mingshuai Chen, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. Learning one-clock timed automata. In *TACAS'20: Proc. 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 12078 of *LNCS*, pages 444–462. Springer, 2020. doi:10.1007/978-3-030-45190-5_25.

3 Jie An, Bohua Zhan, Naijun Zhan, and Miaomiao Zhang. Learning Nondeterministic Real-Time Automata. *ACM Transactions on Embedded Computing Systems*, 20(5s):1–26, 2021. doi:10.1145/3477030.

4 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987. doi:10.1016/0890-5401(87)90052-6.

5 Devendra Bhave and Shibashis Guha. Adding Dense-Timed Stack to Integer Reset Timed Automata. In *RP'17: Proc. 11th International Conference on Reachability Problems*, volume 10506 of *LNCS*, pages 9–25. Springer, 2017. doi:10.1007/978-3-319-67089-8_2.

6 Mikołaj Bojańczyk and Sławomir Lasota. A Machine-Independent Characterization of Timed Languages. In *ICALP'12: Proc. of the 39th Int. Colloquium of Automata, Languages and Programming*, volume 7392, pages 92–103. Springer, 2012. doi:10.1007/978-3-642-31585-5_12.

7 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004. doi:10.1016/j.tcs.2004.04.003.

8 Véronique Bruyère, Bharat Garhewal, Guillermo A. Pérez, Gaëtan Staquet, and Frits W. Vaandrager. Active learning of mealy machines with timers. *CoRR*, abs/2403.02019, 2024. doi:10.48550/arxiv.2403.02019.

9 Kyveli Doveri, Pierre Ganty, and B. Srivathsan. A myhill-nerode style characterization for timed automata with integer resets. *CoRR*, abs/2410.02464, 2024. doi:10.48550/arxiv.2410.02464.

10 Olga Grinchtein, Bengt Jonsson, and Martin Leucker. Learning of event-recording automata. *Theoretical Computer Science*, 411(47):4029–4054, 2010. doi:10.1016/j.tcs.2010.07.008.

11 Olga Grinchtein, Bengt Jonsson, and Paul Pettersson. Inference of event-recording automata using timed decision trees. In *CONCUR'06: Proc. 17th International Conference on Concurrency Theory*, volume 4137 of *LNCS*, pages 435–449. Springer, 2006. doi:10.1007/11817949_29.

12 Shang-Wei Lin, Étienne André, Jin Song Dong, Jun Sun, and Yang Liu. An efficient algorithm for learning event-recording automata. In *ATVA'11: Proc. 9th International Symposium on Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 463–472. Springer, 2011. doi:10.1007/978-3-642-24372-1_35.

13 Anirban Majumdar, Sayan Mukherjee, and Jean-François Raskin. Greybox learning of languages recognizable by event-recording automata. In *ATVA'24: Proc. 22nd International Symposium on Automated Technology for Verification and Analysis*, LNCS. Springer, 2024.

14 Oded Maler and Amir Pnueli. On Recognizable Timed Languages. In *FoSSaCS'04: Proc. of the Int. Conf. on Foundations of Software Science and Computation Structures*, volume 2987 of *LNCS*, pages 348–362. Springer, 2004. doi:10.1007/978-3-540-24727-2_25.

15 Lakshmi Manasa and Krishna S. Integer Reset Timed Automata: Clock Reduction and Determinizability. *CoRR*, abs/1001.1215, 2010. doi:10.48550/arxiv.1001.1215.

16 Jan Springintveld and Frits W. Vaandrager. Minimizable timed automata. In *FTRTFT'96: Proc. of 4th Int. Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1135 of *LNCS*, pages 130–147. Springer, 1996. doi:10.1007/3-540-61648-9_38.

17 P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed Automata with Integer Resets: Language Inclusion and Expressiveness. In *FORMATS'08: Proc. of the Int. Conf. on Formal Modeling and Analysis of Timed Systems*, volume 5215, pages 78–92. Springer, 2008. doi:10.1007/978-3-540-85778-5_7.

18 Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In *FORMATS'19: Proc. 17th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 11750 of *LNCS*, pages 216–235. Springer, 2019. doi:10.1007/978-3-030-29662-9_13.

19 Frits W. Vaandrager, Masoud Ebrahimi, and Roderick Bloem. Learning mealy machines with one timer. *Inf. Comput.*, 295(Part B):105013, 2023. doi:10.1016/J.IC.2023.105013.

20 Sicco Verwer. *Efficient Identification of Timed Automata: Theory and practice*. PhD thesis, Delft University of Technology, Netherlands, 2010. URL: http://resolver.tudelft.nl/uuid:61d9f199-7b01-45be-a6ed-04498113a212.

21 Masaki Waga. Active Learning of Deterministic Timed Automata with Myhill-Nerode Style Characterization. In *CAV'23: Proc. of the 35th Int. Conf. on Computer Aided Verification*, volume 13964 of *LNCS*, pages 3–26. Springer, 2023. doi:10.1007/978-3-031-37706-8_1.

22 Runqing Xu, Jie An, and Bohua Zhan. Active learning of one-clock timed automata using constraint solving. In *ATVA'22: Proc. 20th International Symposium on Automated Technology for Verification and Analysis*, volume 13505 of *LNCS*, pages 249–265. Springer, 2022. doi:10.1007/978-3-031-19992-9_16.

Counterfactual Explanations for MITL Violations

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Felix Jahn
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Julian Siber
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

MITL is a temporal logic that facilitates the verification of real-time systems by expressing the critical timing constraints placed on these systems. MITL specifications can be checked against system models expressed as networks of timed automata. A violation of an MITL specification is then witnessed by a *timed trace* of the network, i.e., an execution consisting of both discrete actions and real-valued delays between these actions. Finding and fixing the root cause of such a violation requires significant manual effort since both discrete actions and real-time delays have to be considered. In this paper, we present an automatic explanation method that eases this process by computing the root causes for the violation of an MITL specification on the execution of a network of timed automata. This method is based on newly developed definitions of counterfactual causality tailored to networks of timed automata in the style of Halpern and Pearl's actual causality. We present and evaluate a prototype implementation that demonstrates the efficacy of our method on several benchmarks from the literature.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Modal and temporal logics

Keywords and phrases Timed automata, actual causality, metric interval temporal logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.22

Supplementary Material
Software: `https://github.com/reactive-systems/rt-causality.git` [44]
archived at `swh:1:dir:de6b34eb1137d85c4257b5adac4b15646bd8ea3e`

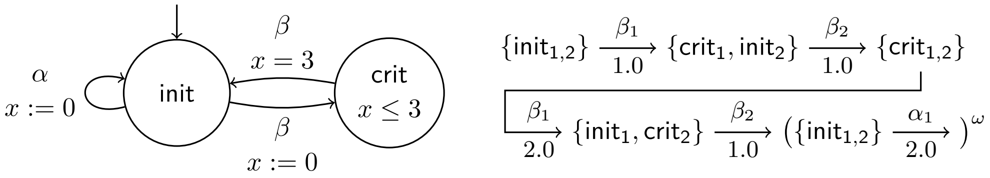
Funding This work was partially supported by the DFG in project 389792660 (Center for Perspicuous Systems, TRR 248) and by the ERC Grant HYPER (No. 101055412).

1 Introduction

Networks of timed automata are a popular formalism to model a wide range of real-time systems such as automotive controllers [27, 50] and communication protocols [23, 39]. These models can be automatically checked against specifications in *Metric Interval Temporal Logic* (MITL) [4], a real-time extension of linear-time temporal logic that allows to constrain temporal operators to non-singleton intervals over the real numbers. In case a network of timed automata does not satisfy an MITL specification, a model-checking procedure will return an execution of the network as a counterexample. Such an execution is defined by discrete actions of the automata in the network and by real-valued delays that describe the time that passes between the discrete actions. Hence, fixing an erroneous system requires insight into both actions and delays that caused the violation on the given counterexample.

In this paper, we present an approach that facilitates this insight through counterfactual explanations for the observed violation. Previous approaches for explaining real-time violations only consider safety properties [52] or only real-time delays without discrete actions [46], and hence cannot provide a comprehensive insight for violations of unconstrained MITL

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 22; pp. 22:1–22:25
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


α
x := 0
init
β
x = 3
crit
x ≤ 3
β
x := 0
{init1,2} —β1, 1.0→ {crit1, init2} —β2, 1.0→ {crit1,2}
—β1, 2.0→ {init1, crit2} —β2, 1.0→ ({init1,2} —α1, 2.0→)ω

(a) A component automaton of network $\mathcal{A}_1 \,||\, \mathcal{A}_2$. **(b)** An execution of the network $\mathcal{A}_1 \,||\, \mathcal{A}_2$.

Figure 1 The network and its execution discussed as an illustrative example in Subsection 1.1.

properties. Like related efforts for discrete systems [10, 20], we ground our explanation method in the theory of *actual causality* as formalized by Halpern and Pearl [34, 35, 36] and identify the actions and delays that are actual causes for the violation of the specification on the observed counterexample. This approach faces several new challenges when confronted with real-time models expressed as networks of timed automata, instead of the previously considered structural equation models [35], finite-state machines [20], and traces [10].

The first challenge pertains to the concept of *interventions*, which describe how the observed counterexample is modified when hypothetical counterfactual executions are considered during the analysis. While previous results usually consider models where the set of counterfactual scenarios is finite, modifying delays in executions of timed automata gives rise to infinitely many counterfactual scenarios. Our main insight to solve this problem is based on constructing networks of timed automata that model all such counterfactual executions, such that checking a causal hypothesis or even synthesizing a cause from scratch can be realized through model checking of these newly constructed automata. Actual causality in models with infinitely many variables, each potentially having an infinite domain, is only starting to be understood [37] and our results suggest that known techniques from timed automata verification are partially transferable to this general theory, e.g., for cause computation.

A second challenge we face in networks of timed automata pertains to the concept of *contingencies*. When two or more potential causes preempt each other, contingencies allow to isolate the true, non-preempted cause from the others. In structural equations models [33] and Coenen et al.'s definition for finite-state machines [20], this is realized by extending the system dynamics with resets that set variables back to the value they had in the actual, original scenario. Networks of timed automata have both local variables, i.e., component locations, and global variables such as clocks. We account for this by defining two automata constructions that allow such resets through contingencies on the local level by single components, as well as on the network level for global clock variables.

1.1 Illustrative Example

We discuss our approach for causal analysis with the example of a small network of timed automata $\mathcal{A}_1 \,||\, \mathcal{A}_2$ consisting of two identical component automata as depicted in Figure 1a, which we will also use as a running example throughout the paper. The two automata can each switch between the two locations init and crit, but whenever they enter the location crit with action β, they are required to stay there for exactly three time units. This is realized through an initial reset of a global clock variable ($x := 0$) with the first β action and a location invariant ($x \leq 3$) in location crit, as well as a clock guard ($x = 3$) on the second β action. We want to check the mutual exclusion property $\square_{[0,\infty)}(\neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2)$ expressed in MITL, which states that the two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are never both in location crit. It is easy to see that this property is violated, e.g., by the execution depicted in Figure 1b. This

Table 1 A contrastive overview of the four root causes on the execution in the illustrative example, inferred using but-for causality and actual causality.

Ref.	But-For Causes	Actual Causes	Intuitive Description
1	$\{(1.0, 1, \mathcal{A}_1)\}$	$\{(1.0, 1, \mathcal{A}_1)\}$	The first component did not wait.
2	$\{(2.0, 1, \mathcal{A}_2)\}$	$\{(2.0, 1, \mathcal{A}_2)\}$	The second component did not wait.
3	**a:** $\{(\beta, 1, \mathcal{A}_1), (\beta, 2, \mathcal{A}_1)\}$ **b:** $\{(\beta, 1, \mathcal{A}_1), (3.0, 2, \mathcal{A}_1)\}$	$\{(\beta, 1, \mathcal{A}_1)\}$	The first component entered crit.
4	$\{(\beta, 1, \mathcal{A}_2)\}$	$\{(\beta, 1, \mathcal{A}_2)\}$	The second component entered crit.

(simplified) execution is an infinite sequence of location labels constructed from delays and discrete actions, where the ω-part is repeated infinitely often. For abbreviation, we place the delay values and actions on the same arrow, which means that the action above the arrow is performed after delaying for as long as specified under the arrow. Both action and location labels refer to a component automaton performing the action and being in a location, respectively, through their index. The execution depicted in Figure 1b respects the dynamics of the automata, e.g., exactly three time units pass between entering and leaving crit. As we can see, Automaton 2 uses a β action less than three time units after Automaton 1, while the latter needs to stay in crit for exactly three time units.

We generate explanations through counterfactual reasoning: For instance, we can infer that one cause of the violation above is that the second component waits only two time units before entering crit by considering hypothetical executions with alternative delays at this particular point, all else being the same. This relaxed model allows an execution where the second component waits with entering the crit location until after the first component has already left theirs, such that no violation occurs. Hence, we can infer the *but-for* cause $\{(2.0, 1, \mathcal{A}_2)\}$ which says that the first delay of 2.0 time units by component $\mathcal{A}_2$ is a root cause for the violation. We measure delays locally on the component level and hence need to add all global delays between actions of the second component as defined in the execution above. Table 1 lists this cause (Cause 2) along with the other root causes inferred through *but-for* causal analysis in the second column. Cause 1 expresses that the delay of component $\mathcal{A}_1$ can similarly be set high enough that no violation occurs.

Cause 3 shows that the *but-for* counterfactual analysis is not always enough: With this naïve criterion we cannot infer that the first β action of $\mathcal{A}_1$ is a cause for the violation of the property on this execution, since changing it alone to, e.g., α, does not suffice to avoid the violation. The second β then steps in to produce the same effect, which means we are dealing with a *preemption* of potential causes. In the but-for causal analysis, we consequently have to additionally intervene on the preempted causes to obtain executions to avoid the effect. In this case, we can either additionally change the second β (Cause 3a), or the second delay (Cause 3b – this way we can set the entering of crit to after component $\mathcal{A}_2$ has already left). These larger causes are not desirable, because they do not only point to the root of the issue. As a solution in such cases of preemption, Halpern and Pearl [35] suggest *contingencies*, and Coenen et al. [20] have recently lifted this to finite-state machines with infinite executions. Inspired by these efforts, we propose a contingency mechanism for networks of timed automata that similarly allows us to infer the first β as the true cause in the given scenario (Cause 3). This mechanism extends the network with contingency edges that, e.g., allow the second β to move to the same location as in the original execution, i.e., to init. This then produces a witnessing counterfactual run that avoids the effect.

1.2 Outline and Contributions

After recalling preliminaries in Section 2, we develop our definitions of counterfactual causality in networks of timed automata (Section 3). We follow Halpern's approach [34] in first defining a notion of *minimal but-for* causality. Counterfactual reasoning is realized through an automaton construction that allows to search for a witnessing intervention in the infinite set of counterfactual runs through model checking. Inspired by Coenen et al. [20, 21], we extend but-for causality through a construction of contingency automata, which model contingencies on the local level of components as well as on the network level for global variables such as clocks (Subsection 3.3), yielding a main building block for our definition of *actual* causality. In Section 4, we present algorithms for computing and checking but-for and actual causes. These algorithms exploit a property of both notions of causality that we term cause *monotonicity*, which allows us to reduce the potential causes we need to consider during computation. We have implemented a prototype of this algorithm and report on its experimental evaluation in Section 5. We show that causes can be computed in reasonable time and help in narrowing down the behavior responsible for an MITL violation. To summarize, we make the following contributions:

- We define and study the notions of but-for causality and actual causality in networks of timed automata, for effects described by arbitrary MITL properties;
- We propose an algorithm for computing these causes and study its theoretical complexity;
- We report the results of a prototype implementation of this algorithm for automated explanations of counterexamples in real-time model checking.

2 Preliminaries

We recall background on actual causality, timed automata as models of real-time systems, and MITL as a temporal logic for specifying real-time properties.

2.1 Actual Causality

We recall Halpern's modified version [33] of actual causality [35], which uses *structural equation models* to define the causal dependencies of a system. Formally, a *causal model* is a tuple $M = (\mathcal{S}, \mathcal{F})$ that consists of a *signature* $S = (\mathcal{U}, \mathcal{V}, \mathcal{R})$ and *structural equations* $\mathcal{F} = \{F_X \mid X \in \mathcal{V}\}$. The sets $\mathcal{U}$ and $\mathcal{V}$ define *exogenous variables* and *endogenous variables*, respectively. The *range* $\mathcal{R}(Y)$ specifies the possible values of each variable $Y \in \mathcal{Y} = \mathcal{U} \cup \mathcal{V}$. A structural equation $F_X \in \mathcal{F}$ defines the value of an endogenous variable $X \in \mathcal{V}$ as a function $F_X : (\times_{Y \in \mathcal{Y} \setminus \{X\}} \mathcal{R}(Y)) \to \mathcal{R}(X)$ of the values of all other variables in $\mathcal{U} \cup \mathcal{V}$, without creating cyclic dependencies in $\mathcal{F}$. Therefore, the structural equations have a unique solution for a given *context* $\vec{u} \in (\times_{U \in \mathcal{U}} \mathcal{R}(U))$, i.e., a valuation for the variables in $\mathcal{U}$. Actual causality then defines whether a value assignment $\vec{X} = \vec{x}$ causes φ, a conjunction of *primitive events* $Y = y$ for $Y \in \mathcal{V}$, in a given context.

▶ **Definition 1** (Halpern's Version of Actual Causality [33]). *$\vec{X} = \vec{x}$ is an actual cause of φ in $(M, \vec{u})$, if the following three conditions hold:*

AC1. *$(M, \vec{u}) \models \vec{X} = \vec{x}$ and $(M, \vec{u}) \models \varphi$.*

AC2. *There is a* contingency *$\vec{W} \subseteq \mathcal{V}$ with $(M, \vec{u}) \models \vec{W} = \vec{w}$ and a setting $\vec{x}'$ for the variables in $\vec{X}$ s.t. $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}]\neg\varphi$.*

AC3. *$\vec{X}$ is minimal, i.e., no strict subset of $\vec{X}$ satisfies* ***AC1*** *and* ***AC2****.*

AC1 simply states that both the cause and the effect have to be satisfied in the given context $\vec{u}$ and causal model $\mathcal{M}$. AC2 appeals to an *intervention* $\vec{X} \leftarrow \vec{x}'$ that overrides the structural equations for all $\vec{X}_i \in \vec{X}$ such that $F_{\vec{X}_i} = \vec{x}'_i$. While the witness $\vec{x}'$ can be chosen arbitrarily, the valuation $\vec{w}$ for the *contingency* variables $\vec{W}$ has to be the same as in the original context. The contingency is applied after the intervention, and in this way allows to reset certain variables to their original values, with the aim to infer more precise causes in certain scenarios. Hence, AC2 requires that some intervention together with a contingency avoids the effect, i.e., the resulting solution to the modified structural equations falsifies at least one primitive event in φ. AC3 ensures that $\vec{X} = \vec{x}$ is a concise description of causal behavior by enforcing minimality. In particular, this ensures that for no variable the valuation in $\vec{x}'$ (AC2) coincides with its original valuation in $\vec{x}$.

▶ **Example 2.** We recall a classic example of Suzy and Billy throwing rocks at a bottle [35]. We have the endogenous variables BT, ST for Billy and Suzy throwing their rock, respectively. BH, SH signify that they hit, and BB encodes that the bottle breaks from a hit. BT and ST directly depend on some nondeterministic exogenous variables, while the other structural equations are $BH = BT \wedge \neg ST$, $SH = ST$ and $BB = BH \vee SH$, i.e., Suzy's throw is always faster than Billy's. Hence, in the context where both throw their rock, we have $BT = ST = SH = BB = 1$ and $BH = 0$. The *intervention* $ST = 0$ does not suffice to avoid the effect, because the structural equations still evaluate to $BB = 1$ due to Billy's throw. We say Billy's throw was *preempted*. We can pick the contingency $BH = 0$ from the original evaluation as a *contingency*. This means we set both $ST = 0$ and $BH = 0$, the latter of which "blocks" the influence of Billy's throw, and obtain an evaluation where the effect disappears, i.e., with $BB = 0$. Finally, only the event $ST = 1$ is in the cause.

2.2 Networks of Timed Automata

We use networks of timed automata [1] to model real-time systems. We fix a finite set AP of *atomic propositions* and a finite set of *actions* Act. Given a set of real-valued *clocks* X, a *clock constraint* is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ with $x, y \in X$, $\sim \in \{<, \leq, =, \geq, >\}$, and $n \in \mathbb{N}$. The set of clock constraints over a clock set X is denoted $\mathsf{C}(X)$. Then, a *timed automaton* is a tuple $\mathcal{A} = (Q, q_0, X, E, I, L)$, where Q is a finite set of *locations*, $q_0 \in Q$ is the *initial location*, X is a finite set of *clocks*, $E \subseteq (Q \times \mathsf{C}(X) \times Act \times \mathsf{U}(X) \times Q)$ is the *edge relation*, $I : Q \to \mathsf{C}(X)$ is an *invariant assignment*, and $L : Q \to 2^{AP}$ is a *labeling function*. We consider a version of *updatable* timed automata that can reset clocks to constants [16]. Hence, the set of *clock updates* $\mathsf{U}(X)$ is the set of partial functions mapping clocks to natural numbers: $\mathsf{U}(X) = \{U : X \rightharpoonup \mathbb{N}\}$. A *clock assignment* for a set of clocks X is a function $u : X \to \mathbb{R}_{\geq 0}$. u_0 denotes the assignment where all clocks are mapped to zero. We write $u \models g$ if u satisfies a clock constraint $g \in \mathsf{C}(C)$, $u + \delta$ for the clock assignment that results from u after $\delta \in \mathbb{R}_{\geq 0}$ time units have passed, i.e., $(u + \delta)(x) = u(x) + \delta$, and $u \leftarrow U$ for the assignment that updates u in accordance with U, i.e., $(u \leftarrow U)(x) = U(x)$ if $x \in dom(U)$ else $(u \leftarrow U)(x) = u(x)$.

▶ **Definition 3** (Semantics of Timed Automata). *The semantics of a timed automaton* $\mathcal{A} = (Q, q_0, X, E, I, L)$ *is defined by a transition system* $(Q \times \mathbb{R}_{\geq 0}^{|X|}, (q_0, u_0), \to)$*, where* $\to$ *contains:*

delays: $(q, u) \xrightarrow{\delta} (q, u + \delta)$ *iff* $\delta \in \mathbb{R}_{\geq 0}$ *and* $(u + \delta') \models I(q)$ *for all* $0 \leq \delta' \leq \delta$*, and*

actions: $(q, u) \xrightarrow{\alpha} (q', u \leftarrow U)$ *iff* $(q, g, \alpha, U, q') \in E$*,* $u \models g$*, and* $(u \leftarrow U) \models I(q')$*.*

A *run* $\rho = (q_0, u_0) \xrightarrow{\delta_1}\xrightarrow{\alpha_1} (q_1, u_1) \xrightarrow{\delta_2}\xrightarrow{\alpha_2} \ldots$ of $\mathcal{A}$ is a sequence of alternating delay and action transitions. The set $\Pi(\mathcal{A})$ is the set of all runs of $\mathcal{A}$. The *trace* $\pi(\rho) = \langle \delta_1^\rho, \alpha_1^\rho \rangle \langle \delta_2^\rho, \alpha_2^\rho \rangle \ldots$ of a run ρ is the sequence of delay and action transitions. We sometimes denote the elements of some run ρ or trace π at index i with q_i^ρ, δ_i^ρ etc. We define the accumulated delay as $\delta(i,j) = \sum_{k=i,\ldots,j} \delta_k$ and $\delta_0 = 0$. The *signal* σ^ρ of the run ρ maps time points to location labels: $\sigma^\rho(t) = \{a \mid \exists i.\ a \in L(q_i) \wedge \delta(0,i) \leq t < \delta(0, i+1)\}$. The language $\mathcal{L}(\mathcal{A})$ is the set of all signals with a corresponding run of $\mathcal{A}$. We use this *left-closed right-open* interpretation of signals due to Maler et al. [51] because of its simplicity. It is straightforward to extend our counterfactual analysis technique to other semantics, e.g., continuous time and point wise [4], or even to other logics with linear-time semantics, as long as their model checking problem is decidable. Note that we make use of an intersection operation $\cap$ for timed automata which intersects the *actions*, i.e., the edge label of the automata. You may assume that the operation unifies the labels of the locations, but we apply it such that only one operand automaton has location labels. This means that the result of $\mathcal{A}_1 \cap \mathcal{A}_2$ is not (singal-based) language intersection in the classical sense, i.e., we do not have $\mathcal{L}(\mathcal{A}_1 \cap \mathcal{A}_2) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$.

▶ **Definition 4** (Network of Timed Automata). *A network of timed automata $\mathcal{A}_1 \mid\mid \ldots \mid\mid \mathcal{A}_n$ is constructed through parallel composition. Let $\mathcal{A}_i = (Q^i, l_0^i, X, E^i, I^i, L^i)$ for all $1 \leq i \leq n$ with a common set of global clocks X. The network $\mathcal{A}_1 \mid\mid \ldots \mid\mid \mathcal{A}_n$ is defined by the automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$, where the locations are the Cartesian product $Q = Q^1 \times \ldots \times Q^n$, with the initial state $q_0^n = (q_0^1, \ldots, q_0^n)$, the invariants are combined as $I(\vec{q}) = \bigwedge_{1 \leq i \leq n} I^i(q_i)$, and the labels are unified as $L(q) = \bigcup_{1 \leq i \leq n} L^i(q_i)$. The edge relation E contains two types:*

internal: $\big(\vec{q}, g, \langle \mathcal{A}_i, \mathcal{A}_i, \alpha \rangle, U, \vec{q}[q_i'/q_i]\big)$ *iff* $(q_i, g, \alpha, U, q_i') \in E^i$*, and*

synchronized: $\big(\vec{q}, g_i \wedge g_j, \langle \mathcal{A}_i, \mathcal{A}_j, \alpha \rangle, U_i \cup U_j, \vec{q}[q_i'/q_i, q_j'/q_j]\big)$ *iff* $i \neq j$*,* $(q_i, g_i, \alpha, U, q_i') \in E^i$*, and* $(q_j, g_j, \bar{\alpha}, U, q_j') \in E^j$*.*

Hence, we do explicitly identify the component automata participating in action transitions by constructing tuples containing actions and automata handles. This is for technical convenience in later constructions, and we define a predicate to check whether an automaton participates in an action transition as *participates*$(\mathcal{A}_i, \langle \mathcal{A}_j, \mathcal{A}_k, \alpha \rangle) := (i = j) \vee (i = k)$, as well as a partial function for accessing the original action as *action*$(\mathcal{A}_i, \langle \mathcal{A}_j, \mathcal{A}_k, \alpha \rangle) = \alpha$ iff $i = j$ and *action*$(\mathcal{A}_i, \langle \mathcal{A}_j, \mathcal{A}_k, \alpha \rangle) = \bar{\alpha}$ iff $i = k$.

2.3 Metric Interval Temporal Logic

We use *Metric Interval Temporal Logic* (MITL) [4] for defining real-time properties such as system specifications and effects. The syntax of MITL formulas over a set of atomic propositions AP is defined by $\phi := p \mid \neg\phi \mid \phi \wedge \phi \mid \phi\, \mathcal{U}_I\, \phi$, where $p \in AP$ and I is a non-singleton interval of the form $[a,b]$, $(a,b]$, $[a,b)$, (a,b), (a,∞), or $[a,\infty)$ with $a, b \in \mathbb{N}$ and $a < b$. We also consider the usual derived Boolean and temporal operators ($\Diamond_I\ \phi := \top\ \mathcal{U}_I\ \phi$, $\Box_I\ \phi := \neg \Diamond_I\ \neg\phi$, $\phi\, \mathcal{U}\, \psi := \phi\, \mathcal{U}_{[0,\infty)}\, \psi$, $\Diamond\ \phi := \Diamond_{[0,\infty)}\ \phi$, and $\Box\ \phi := \Box_{[0,\infty)}\ \phi$).
The semantics of MITL is defined inductively with respect to a signal $\sigma : \mathbb{R}_{\geq 0} \to 2^{AP}$ and a timepoint $t \in \mathbb{R}_{\geq 0}$.

$$\begin{array}{lll}
\sigma, t \models p & \text{iff} & p \in \sigma(t) \\
\sigma, t \models \neg\phi & \text{iff} & \sigma, t \not\models \phi \\
\sigma, t \models \phi \wedge \psi & \text{iff} & \sigma, t \models \phi \text{ and } \sigma, t \models \psi \\
\sigma, t \models \phi\, \mathcal{U}_I\ \psi & \text{iff} & \exists t' > t.\ t' - t \in I,\ \sigma, t' \models \psi \text{ and } \forall t'' \in (t, t').\, \sigma, t'' \models \phi
\end{array}$$

A run ρ *satisfies* an MITL formula ϕ, iff $\sigma(\rho), 0 \models \phi$. A timed automaton $\mathcal{A}$ *satisfies* ϕ, iff all of its runs satisfy ϕ. We write $\rho \models \phi$ and $\mathcal{A} \models \phi$, respectively.

3 Counterfactual Causality in Real-Time Systems

In this section, we develop two notions of counterfactual causality in real-time systems. We first define our language for describing causes and how they define interventions on timed traces (Subsection 3.1). We then start with a simple notion of *but-for* causality in networks of timed automata based on interventions without contingencies (Subsection 3.2). Afterward, we outline how to model contingencies in a timed automaton (Subsection 3.3) and use them to define *actual* causes, in the sense of Halpern and Pearl (cf. Subsection 2.1). Note that the proofs of all nontrivial statements of this section are in Appendix A.

3.1 Interventions on Timed Traces

We describe actual causes as finite sets of *events*. Events have two distinct types such that they either refer to an action or a delay transition in a given run.

▶ **Definition 5** (Event). *A delay event is a tuple* $(\delta, i) \in \mathbb{R}_{\geq 0} \times \mathbb{N}$ *and an action event is a tuple* $(\alpha, i) \in Act \times \mathbb{N}$*. The sets of all delay and action events are denoted as* $\mathcal{DE}$ *and* $\mathcal{AE}$*, respectively. The set of all events is* $\mathcal{E} = \mathcal{DE} \,\dot{\cup}\, \mathcal{AE}$*. For a trace* π*, the set of events on* π *is defined as* $\mathcal{E}_\pi = \{(\alpha_i^\pi, i) \mid i \in \mathbb{N}_{>0}\} \cup \{(\delta_i^\pi, i) \mid i \in \mathbb{N}_{>0}\}$*.*

When we describe a cause as a set of events, we are mainly interested in the counterfactual runs obtained by modifying the events contained in the cause. In the style of Halpern and Pearl, we call such modifications *interventions*. If the actual run is given in a finite, lasso-shaped form and the cause is a finite set of events, these interventions can be described by a timed automaton that follows the dynamics of the actual trace, except for events that appear in the cause. For these events, the behavior is relaxed to allow arbitrary alternative actions or delays. We call a run ρ *lasso-shaped* if it can be composed of a (possibly empty) prefix and an infinitely occurring loop, i.e., if it is of the form

$$\rho = (q_0, u_0) \dots \big((q_n, u_n) \dots (q_{p-1}, u_{p-1}) \xrightarrow{\delta_p} \xrightarrow{\alpha_p} \big)^\omega \ ,$$

where the ω-part is repeated infinitely often. Note that strictly speaking, a lasso-shaped trace as defined here does not exists for all models that violate an MITL property, because clock valuations are not guaranteed to stabilize in some infinitely-repeating loop $u_n \dots u_{p-1}$. We use the valuations to reset clock values in our contingency construction that will be introduced in Subsection 3.3. This construction may be generalized by considering clock *regions* or *zones* [14] instead of the valuations. This requires defining the resets in the contingency automaton accordingly.

In this paper, we simplify by assuming the existence of a lasso-shaped run as defined above. In general, we can further assume the clocks to be assigned to values in $\mathbb{Q}$, as timed automata do not distinguish between the real and rational numbers [3]. For a lasso-shaped run ρ as described above, we define a function to access the successor index of an action as $dst_\rho : \{1, \dots, p\} \mapsto \{0, \dots, p-1\}$ with $dst_\rho(k) = k$ if $k \neq p$ and $dst_\rho(p) = n$ else. We define the length of the run ρ as $|\rho| = p$. The functions dst_π and $|\pi|$ are defined analogously for the trace of a lasso-shaped run. We are now ready to define the automaton modelling traces with interventions.

$$\frac{(\delta_i^\pi, i) \notin \mathcal{C} \qquad (\alpha_i^\pi, i) \notin \mathcal{C}}{\big(i-1, d = \delta_i^\pi, \alpha_i^\pi, d := 0, dst_\pi(i)\big) \in E} \qquad \frac{(\delta_i^\pi, i) \in \mathcal{C} \qquad (\alpha_i^\pi, i) \notin \mathcal{C}}{\big(i-1, \top, \alpha_i^\pi, d := 0, dst_\pi(i)\big) \in E}$$

$$\frac{(\delta_i^\pi, i) \notin \mathcal{C} \qquad (\alpha_i^\pi, i) \in \mathcal{C} \qquad \beta \in Act}{\big(i-1, d = \delta_i^\pi, \beta, d := 0, dst_\pi(i)\big) \in E} \qquad \frac{(\delta_i^\pi, i) \in \mathcal{C} \qquad (\alpha_i^\pi, i) \in \mathcal{C} \qquad \beta \in Act}{\big(i-1, \top, \beta, d := 0, dst_\pi(i)\big) \in E}$$

Figure 2 Rules defining the edge relation E of the counterfactual trace automaton $\mathcal{A}_\pi^\mathcal{C}$.

▶ **Definition 6** (Counterfactual Trace Automaton). *Let π be a lasso-shaped trace over the set of actions Act and let $\mathcal{C} \subseteq \mathcal{E}_\pi$ be a finite set of events. The counterfactual trace automaton of trace π for the set of events $\mathcal{C}$ is defined as $\mathcal{A}_\pi^\mathcal{C} := (Q, q_0, X, E, I, L)$ with $Q := \{0, \dots, |\pi|-1\}$, $q_0 := 0$, $X := \{d\}$. The transition relation E is defined by the following rules depicted in Figure 2, we have $L(q) = \emptyset$ for all $q \in Q$, and*

$$I(q) := \begin{cases} d \le \delta_{q+1}^\pi, & \textit{if } (\delta_{q+1}^\pi, q+1) \notin \mathcal{C} \\ \top, & \textit{otherwise.} \end{cases}$$

The main idea of the counterfactual automaton $\mathcal{A}_\pi^\mathcal{C}$ is to follow the actions and delays of the original run for all events that are not in the event set $\mathcal{C}$, and allow arbitrary action and delays for events in $\mathcal{C}$. Hence, $\mathcal{A}_\pi^\mathcal{C}$ modifies the *trace* of ρ, i.e., the sequence of action and delay events. Subsequently, we will combine a local $\mathcal{A}_\pi^\mathcal{C}$ with the dynamics of the original components to obtain full counterfactual runs of a network of timed automata. The interventions on actions and delays are captured by the rules that define the transition relation and are listed in Figure 2, which treat the different combination of events that may or may not be in the cause at a specific index i. Crucially, the automaton $\mathcal{A}_\pi^\mathcal{C}$ then captures not just a single concrete intervention on the events in $\mathcal{C}$ with respect to the run ρ, such as a modified trace with a specific alternative delay deviating from the actual trace, but *all* (possibly infinitely many) interventions on the events, i.e., it contains all traces with possibly varying actions and delays at specific indices.

▶ **Example 7.** For the trace $\pi = \langle 1.0, \beta\rangle\langle 3.0, \beta\rangle(\langle 2.0, \alpha\rangle)^\omega$ and the set of events $\mathcal{C} = \{(\beta, 1), (3.0, 2)\}$, we depict the counterfactual trace automaton $\mathcal{A}_\pi^\mathcal{C}$ in Figure 3. For the first action and the second delay, arbitrary interventions are allowed, all other action and delay events are enforced to be as in π.

3.2 But-For Causality in Networks of Timed Automata

We now use the construction from the previous section to define counterfactual causes for MITL-expressible effects on runs of networks of timed automata. In practice, an effect ϕ may be the violation of a specification ψ, such that the effect corresponds to the negation of the specification: $\phi \equiv \neg\psi$. The main idea of our definition is to isolate the local traces of the

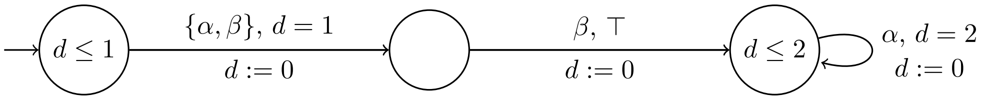


Figure 3 Counterfactual trace automaton $\mathcal{A}_\pi^\mathcal{C}$.

component automata, and then construct a counterfactual trace automaton (cf. Definition 6) for each component, where the former intervenes on the events in a given cause that refer to the specific component. Afterward, each counterfactual trace automaton is intersected with its corresponding component automaton, and the network of all these intersections describes the counterfactual runs after intervention. To apply interventions locally, we start by defining the local projections of a run in a network of timed automata.

▶ **Definition 8** (Local Projection). *For a network $\mathcal{A}^n = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ and one of its runs ρ, we denote $\{j_1, \ldots, j_l\} := \{\, j \in \mathbb{N} \mid participates(\mathcal{A}_i, \alpha^\rho_j)\}$ as the the event points of some component automaton $\mathcal{A}_i$, whereby we let $j_1 < \ldots < j_l$. Then the local projection $\rho(\mathcal{A}_i)$ of the component automaton $\mathcal{A}_i$ is defined as the trace $\rho(\mathcal{A}_i) := \langle \delta_1, \alpha_1 \rangle \langle \delta_2, \alpha_2 \rangle \ldots$, in which*

- *$\alpha^{\rho(\mathcal{A}_i)}_k := action(\mathcal{A}_i, \alpha^\rho_{j_k})$ for all $k = 1, 2, \ldots$, i.e., the identity of the actions is preserved;*
- *$\delta^{\rho(\mathcal{A}_i)}_k = \Sigma_{x=j_{k-1}+1,\ldots,j_k} \delta^\rho_x$ for all $k = 1, 2, \ldots$ and with $j_0 := 1$, i.e., the delays in the local projection are the cumulative delays between two actions of the automaton in the global run of the network.*

Furthermore, we denote with $locations(\rho, \mathcal{A}_i) := (q^\rho_0)_i, (q^\rho_{j_1})_i, (q^\rho_{j_2})_i, \ldots$ the sequence of local locations, i.e., the projection to the i-th component of the network location. We define the localization function as $localize(\rho, \mathcal{A}^n) := (\rho(\mathcal{A}_1), \ldots, \rho(\mathcal{A}_n))$.

Note that the local projection as defined here differs fundamentally from local runs as defined for the local time semantics of timed automata [12], as the clocks still advance globally at the same speed. However, by conducting counterfactual interventions on the delays in a local projection of a run, we are able to change the order of transitions, which is not possible by interacting with delays in the global run of the network. It should also be noted that even if the global run ρ is infinite, the local projections may still turn out to be finite because the transitions occurring infinitely often may stem from a subset of the automata.

▶ **Example 9.** For the run ρ from Subsection 1.1, the first local projection $\rho(\mathcal{A}_1)$ is exactly the trace π considered in Example 7 and $locations(\rho, \mathcal{A}_1) = \mathsf{init}, \mathsf{crit}, (\mathsf{init})^\omega$ as the sequence of local locations. The second local projection $\rho(\mathcal{A}_2)$ is the finite trace $\rho(\mathcal{A}_2) = \langle 2.0, \beta \rangle \langle 3.0, \beta \rangle$.

It is worth pointing out that every global run induces well-defined local projections, however, the tuple $localize(\rho, \mathcal{A})$ of local traces may have multiple associated global runs. This stems from nondeterminism in the order of actions happening at the same timepoint. In essence, we treat the scheduler's decisions in such a situation as nondeterministic, and allow different resolution of this nondeterminism in counterfactual runs of the network.

▶ **Proposition 10.** *The localization function is not injective: There exists a network $\mathcal{A}$ and two runs $\rho \neq \rho'$ such that $localize(\rho, \mathcal{A}) = localize(\rho', \mathcal{A})$.*

Since we want to apply the construction of the counterfactual trace automaton locally to every component, we lift the definition of events from traces to (network) runs, such that the events of a network run are the union of events on local projections of the run.

▶ **Definition 11** (Events of a Network Run). *Given a run ρ of a network $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, we define the set of associated events as*

$$\mathcal{E}_\rho := \{\, (e, i, \mathcal{A}_k) \mid (e, i) \in \mathcal{E}_{\rho(\mathcal{A}_k)} \text{ for some component } 1 \leq k \leq n \,\} \ .$$

We lift the set of all events to a network $\mathcal{A}^n$ and define it as $\mathcal{E}(\mathcal{A}^n) = \{(e, i, \mathcal{A}_k) \mid (e, i) \in \mathcal{E} \wedge 1 \leq k \leq n\}$ and say a run ρ satisfies a set of events $\mathcal{C} \subseteq \mathcal{E}(\mathcal{A}^n)$, denoted $\rho \models \mathcal{C}$, if $\mathcal{C}$ is a subset of the events on ρ, i.e., if $\mathcal{C} \subseteq \mathcal{E}_\rho$. We further define an operator to filter for events of a specific component k: $\mathcal{C}|_k := \{(e, i) \mid (e, i, \mathcal{A}_k) \in \mathcal{C}\}$.

Note that $\mathcal{E}_{\rho(\mathcal{A}_k)}$ contains both action events as well as *locally projected* delay events. Hence, when we speak about the events on a network run we talk about the actions of the respective component automata (identified through the third position in the event tuple), as well as about the time between these actions of a component automaton (i.e., the cumulative delays between two actions of a component). These events are the atomic building blocks of our counterfactual expalantions. With this at hand we can define our first notion of counterfactual causality based on allowing arbitrary alternatives for all the events appearing in a hypothetical cause. The corresponding notion for structural equation models was termed *but-for* causality by Halpern [34], so we adopt the same name here. Crucially, in our setting with networks of timed automata, the alternative delays and events are realized with respect to the local projections of the network run, such that an alternative delay can change the order of actions emerging in different component automata.

▶ **Definition 12** (But-For Causality in Real-Time Systems). *Let $\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$ be a network of timed automata and ρ a run of the network. A set of events $\mathcal{C} \subseteq \mathcal{E}(\mathcal{A}^n)$ is a but-for cause for ϕ in ρ of $\mathcal{A}$, if the following three conditions hold:*

SAT *$\rho \models \mathcal{C}$ and $\rho \models \phi$, i.e., cause and effect are satisfied by the actual run.*

CF$_{\mathsf{BF}}$ *There is an intervention on the events in $\mathcal{C}$ s.t. the resulting run avoids the effect ϕ, i.e., we have*

$$(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi \ .$$

MIN *$\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies* **SAT** *and* **CF$_{\mathsf{BF}}$***.*

▶ **Example 13.** Consider again the system and run from Subsection 1.1, and the cause $\mathcal{C} = \{(\beta, 1, \mathcal{A}_1), (\beta, 2, \mathcal{A}_1)\}$, i.e., the two β-actions of the first component (Cause 3a). **SAT** is satisfied, since the effect $\neg\Box_{[0,\infty)}(\neg\mathsf{crit}_1 \vee \neg\mathsf{crit}_2)$, i.e., the negation of the MITL specification is satisfied and the local projection of $\mathcal{A}_1$ is $\rho(\mathcal{A}_1) = \langle 1.0, \beta\rangle\langle 3.0, \beta\rangle(\langle 2.0, \alpha\rangle)^\omega$. For **CF$_{\mathsf{BF}}$**, consider that the network run emerging from setting $\mathcal{A}_1$'s local projection to $\langle 1.0, \alpha\rangle\langle 3.0, \alpha\rangle(\langle 2.0, \alpha\rangle)^\omega$ does not violate the specification since the first component never enters crit_1. To see that $\mathcal{C}$ also satisfies **MIN** consider its two singleton subsets. Setting the alternative α for either of the actions alone does not suffice to avoid the effect due to the temporal ordering of the β-actions, e.g., when intervening only on the first β, then the second β enters crit_1 while the second component is also in its critical section, hence the effect is still present. Similarly, we can show $\{2.0, 1, \mathcal{A}_2\}$, i.e., the first delay of the second component (Cause 2), as well as all the other but-for causes from Table 1 to be but-for causes for ϕ in ρ.

Besides this intuitive example, we can prove several sanity properties about but-for causality. These properties concern the existence and identity of causes in certain distinctive cases. First up, we show that the existence of a but-for cause is guaranteed as long as a system run avoiding the effect exists.

▶ **Proposition 14.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n = \mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$, then for every run ρ of the network in which ϕ appears, there is a but-for cause for ϕ in ρ of $\mathcal{A}^n$, if and only if there exists a run ρ' of the network with $\rho' \not\models \phi$.*

Next, we consider the case where there is nondeterminism on the actual run, i.e., when there is another run with the same trace, that does no satisfy the effect. In this case, our definition returns the empty set as a unique actual cause.

▶ **Proposition 15.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n$, $\emptyset$ is the (unique) but-for cause for an effect ϕ on a run ρ of $\mathcal{A}^n$, if and only if there exists a run η of $\mathcal{A}^n$ with localize$(\rho, \mathcal{A}^n) =$ localize$(\eta, \mathcal{A}^n)$ and $\eta \not\models \phi$, i.e., a run with the same local traces as the actual run, that does, however, not satisfy the effect.*

From a philosophical point of view, the empty set is a desirable verdict: It conveys that the smallest change necessary to avoid the effect does not consist of any changes of delay or action events, instead simply an alternative resolution of the underlying nondeterminism of this trace suffices to obtain a witnessing counterfactual run. Also from a practical perspective, it is helpful to know that the empty set gets returned only in this distinguishable scenario.

3.3 Contingencies in Networks of Timed Automata

Actual causality employs a *contingency* mechanism to isolate the true cause in the case of preemption. The key idea of contingencies to overcome this preemption is to reset certain propositions in counterfactual executions to their value as it is in the actual world. Coenen et al. [20] have outlined how to model contingencies for lasso-shaped traces of a Moore machine. We now describe a construction that applies this idea to networks of timed automata. The central idea is that the state resets resulting from applying a contingency now do not only reset the discrete machine state, but the clock assignment and the location of the timed automaton, i.e., the full underlying state. However, a central issue in networks of timed automata is that clocks are global variables shared by all component automata of the network, while the location is a local attribute of single components. We respect this dichotomy by allowing location contingencies only by actions of the corresponding component automaton and clock contingencies by any action in the global network. This is realized by two automata constructions, i.e., a local one applied to all component automata (for resetting locations) and a global one applied to the full network (for resetting clocks). In both cases, we model the behavior as an updatable timed automaton, as we outline in the following.

▶ **Definition 16** (Location Contingency Automaton). *Let ρ be a lasso-shaped run of a network and the timed automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$ a component of this network. The location contingency automaton of $\mathcal{A}$ and ρ is defined as $\mathcal{A}^{\mathsf{loc}(\rho)} := (Q', q'_0, X, E', I', L')$ with $Q' := Q \times \{0, \ldots, |locations(\rho, \mathcal{A})| - 1\}$, $q'_0 := \langle q_0, 0\rangle$, $I'(\langle q, i\rangle) := I(q)$, $L'(\langle q, i\rangle) := L(q)$, and E' is defined as follows, where $\pi = localize(\rho, \mathcal{A})$.*

$$\frac{(q, g, \alpha, U, q') \in E \qquad i = 1, \ldots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, U, \langle q', dst_\pi(i)\rangle) \in E'} \qquad\qquad \frac{(q, g, \alpha, U, q') \in E \qquad i = 1, \ldots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, U, \langle q^\pi_j, dst_\pi(i)\rangle) \in E'}$$

The location contingency automaton $\mathcal{A}^{\mathsf{loc}}_\rho$ hence consists of copies of the original system, one for each position in the lasso-shaped local projection π. With an action transition, it moves from one copy into the next, either following the edge (q, g, α, U, q') of the original system (left rule in Definition 16) or moving to the same location q^π_j as present in π at the respective position $dst_\pi(i)$ by applying a contingency (right rule in Definition 16). Note that after the end of the loop in the lasso-shaped projection, the transitions are redirected to the copy corresponding to the initial position of the loop by the definition of the function dst_π. The same principle can now also be applied to global variables. In our setting, this only concerns clocks, but the following definition of the clock contingency automaton can be generalized to all global variables such as integers, if these are included in the system model.

▶ **Definition 17** (Clock Contingency Automaton). *Let ρ be a lasso-shaped run of a timed automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$. The clock contingency automaton of $\mathcal{A}$ and ρ is defined as $\mathcal{A}^{\mathsf{clk}(\rho)} := (Q', q'_0, X, E', I', L')$ with $Q' := Q \times \{0, \ldots, |\rho| - 1\}$, $q'_0 := \langle q_0, 0\rangle$, $I'(\langle q, i\rangle) := I(q)$, $L'(\langle q, i\rangle) := L(q)$, and E' is defined as follows.*

$$\frac{(q, g, \alpha, U, q') \in E \qquad i = 1, \ldots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, U, \langle q', dst_\rho(i)\rangle) \in E'} \qquad\qquad \frac{(q, g, \alpha, U, q') \in E \qquad i = 1, \ldots, |\rho|}{(\langle q, i-1\rangle, g, \alpha, u^\rho_j, \langle q', dst_\rho(i)\rangle) \in E'}$$

Note that strictly speaking we have defined clock updates to values in $\mathbb{Q}$, instead of $\mathbb{N}$ as considered in classic decidability results. It is, however, straightforward to scale these values to the natural numbers [3]. Clearly, the signals modeled by the contingency automata subsume the ones by the original automaton, because it is possible to simply never invoke a contingency and, hence, always follow the dynamics of the original system.

▶ **Proposition 18.** *For all timed automata $\mathcal{A}$ and runs ρ of $\mathcal{A}$, we have that the languages of the contingency automata subsume the language of the original automaton: $\mathcal{L}(\mathcal{A}_\rho^{\mathsf{loc}}) \supseteq \mathcal{L}(\mathcal{A})$ and $\mathcal{L}(\mathcal{A}_\rho^{\mathsf{clk}}) \supseteq \mathcal{L}(\mathcal{A})$.*

▶ **Definition 19** (Actual Causality in Real-Time Systems)**.** *Let $\mathcal{A}_1 \mid\mid \ldots \mid\mid \mathcal{A}_n$ be a network of timed automata and ρ a run of the network. A set of events $\mathcal{C} \subseteq \mathcal{E}(\mathcal{A}^n)$ is an actual cause for ϕ in ρ of $\mathcal{A}$, if the following three conditions hold:*

SAT *$\rho \models \mathcal{C}$ and $\rho \models \phi$, i.e., cause and effect are satisfied by the actual run.*

CF$_{\mathsf{Act}}$ *There is an intervention on the events in $\mathcal{C}$ and a location and clock contingency (denoted by $\mathsf{loc}(\rho)$ and $\mathsf{clk}(\rho)$ resp.) s.t. the resulting run avoids the effect ϕ, i.e., we have*

$$\left((\mathcal{A}_1^{\mathsf{loc}(\rho)} \cap \mathcal{A}_{\rho(\mathcal{A}_1)}^{\mathcal{C}|_1}) \mid\mid \ldots \mid\mid (\mathcal{A}_n^{\mathsf{loc}(\rho)} \cap \mathcal{A}_{\rho(\mathcal{A}_n)}^{\mathcal{C}|_n})\right)^{\mathsf{clk}(\rho)} \not\models \phi \ .$$

MIN *$\mathcal{C}$ is minimal, i.e., no strict subset of $\mathcal{C}$ satisfies* **SAT** *and* ***CF***$_{\mathsf{Act}}$*.*

▶ **Example 20.** Consider the but-for cause $\mathcal{C} = \{(\beta, 1, \mathcal{A}_1), (\beta, 2, \mathcal{A}_1)\}$ from Example 13. This $\mathcal{C}$ is not an *actual* cause because it does not satisfy the **MIN** condition of Definition 19: The subset $\mathcal{C}' = \{(\beta, 1, \mathcal{A}_1)\}$ satisfies **SAT** and **CF**$_{\mathsf{Act}}$. For **CF**$_{\mathsf{Act}}$ we can use contingencies to neutralize the effect of the second β in the local projection $\rho(\mathcal{A}_1) = \langle 1.0, \beta\rangle\langle 3.0, \beta\rangle(\langle 2.0, \alpha\rangle)^\omega$. Since this action moves to init in the original run (cf. Subsection 1.1), it can also move to this location in the contingency automaton $\mathcal{A}_1^{\mathsf{loc}(\rho)}$. Hence we find an intervention (setting $\langle 1.0, \boldsymbol{\beta}\rangle$ to $\langle 1.0, \boldsymbol{\alpha}\rangle$) and a contingency (setting the target location of $\langle 3.0, \beta\rangle$ to init) that avoid the effect together. A more detailed construction of the contingency automaton is given in Appendix C. In fact, $\mathcal{C}' = \{(\beta, 1, \mathcal{A}_1)\}$ is an actual cause (Cause 3) since additionally to **SAT** and **CF**$_{\mathsf{Act}}$ it also satisfies **MIN** – the empty set does not satisfy **CF**$_{\mathsf{Act}}$. Again, also all the other actual causes from Table 1 can be shown to fulfill our definition.

▶ Remark 21. Note that as a consequence of Proposition 18, the statements regarding the existence and identity of causes as proven in Propositions 14 and 15 can be lifted to actual causality, but require replacing the original networks in the equivalence statements by the contingency automata construction used in **CF**$_{\mathsf{Act}}$ (cf. Definition 19).

4 Computing Counterfactual Causes

In this section, we develop algorithms for computing but-for and actual causes for any MITL effect. Proofs and further details related to this section can be found in Appendix B. We only explicitly present the algorithm for but-for causes; for actual causes the central model checking query needs to be substituted (cf. Definition 19, Lines 4 and 10 in Algorithm 1).

In principle, the algorithms are based on enumerating all candidate causes. However, we can speed up this process significantly by utilizing what we term the monotonicity of causes.

▶ **Lemma 22** (Cause Monotonicity)**.** *For every network of timed automaton $\mathcal{A}$, run ρ, and effect ϕ, we have that*

1. *if a set of events $\mathcal{C}$ fulfills* **SAT** *also every subset $\mathcal{C}' \subseteq \mathcal{C}$ fulfills* **SAT***.*
2. *if a set of events $\mathcal{C}$ fulfills* **CF**$_{\mathsf{BF}}$ *(fulfills* **CF**$_{\mathsf{Act}}$*) also every superset $\mathcal{C}' \supseteq \mathcal{C}$ fulfills* **CF**$_{\mathsf{BF}}$ *(fulfills* **CF**$_{\mathsf{Act}}$*).*

Algorithm 1 Compute But-For Causes.

```
Input: network $\mathcal{A} = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, run $\rho$ of $\mathcal{A}$ satisfying effect $\phi$, i.e. $\rho \models \phi$
Output: set of all but-for causes for $\phi$ in $\rho$ of $\mathcal{A}$
$Res_s := \{\}, Res_l := \{\}, Power := \mathcal{P}(\mathcal{E}_\rho)$
for $i = 0, 1, 2, \ldots, \frac{|\mathcal{E}_\rho|}{2}$ do
    for $\mathcal{C} \in Power$ with $|\mathcal{C}| = i$: do
        if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$ then      // cause found?
            $Res_s := Res_s \cup \mathcal{C}$
            $Power := \{\mathcal{C}' \in Power \,|\, \mathcal{C} \not\subseteq \mathcal{C}'\}$;      // remove all supersets
        end
    end
    for $\mathcal{C} \in Power$ with $|\mathcal{C}| = \frac{|\mathcal{E}_\rho|}{2} - i$: do
        if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\, \ldots \,||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$ then      // cause found?
            $Res_l := Res_l \cup \mathcal{C}$
        else
            $Power := \{\mathcal{C}' \in Power \,|\, \mathcal{C}' \not\subseteq \mathcal{C}\}$;      // remove all subsets
        end
    end
end
return $Res_s \cup \{\mathcal{C} \in Res_l \,|\, \neg\exists \mathcal{C}' \subsetneq \mathcal{C}.\, \mathcal{C}' \in Res_s \cup Res_l\}$;   // filter $Res_l$ for MIN
```

The second monotonicity property enables efficient checking of the **MIN** condition, as it suffices to check only the subsets with one element less instead of checking all subsets of a potential cause. For the computation of causes on a given run ρ, a naive approach could now simply enumerate all elements of $\mathcal{P}(\mathcal{E}_\rho)$, that is, all subsets of the all events $\mathcal{E}_\rho$ on ρ, and check whether they form a cause. By further exploiting monotonicity properties, we can find a more efficient enumeration significantly accelerating the computation of causes. The key idea is to enumerate through the powerset $\mathcal{P}(\mathcal{E}_\rho)$ simultaneously from below (starting with the empty cause and then causes of increasing size) and above (starting with the full cause and then causes of decreasing size). This then allows to exclude certain parts of the powerset from the computation in two ways: First, when finding a set of events $\mathcal{C}$ fulfilling $\mathbf{CF}_{\mathsf{BF}}$, we can exclude all of its supersets as we know that they cannot satisfy **MIN**. Second, when finding a set of events $\mathcal{C}$ not fulfilling $\mathbf{CF}_{\mathsf{BF}}$, we can exclude all of its subsets as the monotonicity of $\mathbf{CF}_{\mathsf{BF}}$ implies that $\mathcal{C}' \subseteq \mathcal{C}$ will neither fulfill $\mathbf{CF}_{\mathsf{BF}}$. This idea of the simultaneous enumeration of $\mathcal{P}(\mathcal{E}_\rho)$ is implemented in Algorithm 1.

▶ **Theorem 23.** *Algorithm 1 is sound and complete, i.e., it terminates with*

$$\textsf{Compute But-For Causes}(\mathcal{A}, \rho, \phi) = \{\, \mathcal{C} \,|\, \mathcal{C} \text{ is a but-for cause for } \phi \text{ in } \rho \text{ of } \mathcal{A} \,\} \;,$$

for all networks $\mathcal{A} = \mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$ and runs ρ of $\mathcal{A}$ satisfying an effect ϕ.

While it is clear that our algorithm requires to solve several model checking problems for the effect ϕ, we can show that we cannot do better: Model checking some formula ϕ can be encoded as a cause checking problem. Hence, asymptotically, cause checking and computation scale similar to MITL model checking for the formula φ.

▶ **Theorem 24.** *Checking and computing causes for an effect ϕ on the run ρ in a network of timed automata $\mathcal{A}$ is* EXPSPACE(ϕ)*-complete.*

Table 2 Experimental results. $\boldsymbol{n}$: number of automata in the network; $|\boldsymbol{\rho}|$: run length; $|\boldsymbol{\mathcal{E}_\rho}|$: number of events on the run; $\boldsymbol{\#\mathcal{C}}$: number of but-for/actual causes; $|\boldsymbol{\mathcal{C}}|$: *average* but-for/actual cause size; $\boldsymbol{t}$: runtime for computing but-for/actual causes.

Instance	n	$\lvert\rho\rvert$	$\lvert\mathcal{E}_\rho\rvert$	$\#\mathcal{C}_{BF}$	$\#\mathcal{C}_{Act}$	$\lvert\mathcal{C}_{BF}\rvert$	$\lvert\mathcal{C}_{Act}\rvert$	t_{BF}	t_{Act}
Run. Ex.	2	5	11	6	5	1.83	1.2	5.67s	5.42s
		6	16	10	7	3.2	2	88.2s	128.8s
Run. Ex.	3	5	11	6	5	1.83	1.2	5.70s	5.53s
		7	16	6	6	1.5	1.17	55.0s	78.6s
Run. Ex.	4	9	19	8	7	1.625	1.14	279.3s	331.8s
Fischer	2	4	12	2	2	1	1	2.37s	9.73s
		7	20	5	5	1.2	1.2	273.1s	1499s
Fischer	3	5	14	2	2	1	1	3.40s	16.9s
		7	20	5	5	1.2	1.2	283.6s	1516s
Fischer	4	6	16	2	2	1	1	4.58s	28.8s
		7	20	5	5	1.2	1.2	295.3s	1535s

Note that this discussion on the complexity with respect to the size of the effect abstracts away from the, e.g., the length of the counterexample, which contributes polynomially to cause checking and exponentially to cause computation since we need to check all subsets of events. In practice, we have observed that the bidirectional enumeration of the powerset realized in Algorithm 1 significantly speeds up the compuation of causes.

5 Experimental Evaluation

We have implemented a prototype in Python.[1] For model checking networks of timed automata, we use Uppaal [13] and the library PyUppaal [56]. Our tool can check and compute causes for effects in the fragment of MITL that is supported by the Uppaal verification suite. We conducted experiments on the running example of this paper, as well as on Fischer's protocol, a popular benchmark for real-time model checking. The experiments were run on a MacBook Pro with an Apple M3 Max and 64GB of memory. The results can be found in Table 2. For the running example, the tool found exactly the causes depicted in Table 1; for Fischer's protocol, we report details in Appendix D. The computed causes narrow down the responsible behavior on a given execution, with the average size of the causes between 1 and 3.2 on execution with a large number of events ($|\mathcal{E}_\rho|$). Using contingencies does result in smaller causes (cf. $\boldsymbol{Avg.}\ |\boldsymbol{\mathcal{C}_{BF}}|$ vs. $|\boldsymbol{\mathcal{C}_{Act}}|$) on the running example. This is not the case for Fischer's protocol, where but-for and actual causes are identical. These findings suggest some directions for optimization, since computing but-for causes is more efficient than computing actual causes. Since the latter are always subsets of the former, it may be sensible to first compute but-for causes and then refine them by taking into account contingencies. Further, the times in Table 2 refer to the time to compute *all* causes. Hence, the performance in practical applications may be improved by iteratively presenting the user with (but-for or actual) causes that have already been found during the execution.

[1] Our prototype and benchmarks are available on GitHub [43].

6 Related Work

Providing explanatory insight into *why* a system does not satisfy a specification has been of growing interest in the verification community: Baier et al. [6] provide a recent and detailed survey. Most works focus on discrete systems and perform error localization in executions [9, 31, 59, 45] or by identifying responsible components [58, 29, 28, 60, 32, 5]. There are also several works on program slicing for analyzing dependencies between different parts of a program [61, 41, 38]. The concepts of vacuity and coverage can be used to gain causal insight also in the case of a successful verification [11, 8, 42, 18]. There are several recent works that take a state-based view of responsibility allocation in transition systems [7, 53], but they do not consider infinite state systems where such an approach is not directly applicable. There are several works [54, 21, 26] that use a notion of distance defined by similarity relations in the counterfactual tradition of Lewis [49]. These are more closely related to our work since the minimality criterion in our definitions of but-for and actual causality can be interpreted as a similarity relation [26]. Like this paper, a range of works has been inspired by Halpern and Pearl's actual causality for generating explanations [10, 22, 32, 20, 48, 57, 15]. Our contingency automata constructions are particularly inspired by Coenen et al. [20, 21]. There is a growing interest in counterfactual causality in models with infinitely many variables or infinite domains [26, 37]. In the latter work, Halpern and Peters provide an axiomatic account for counterfactual causes in such (structural equation) models, where variables are further allowed to have infinite ranges. Our results suggest that fragments of structural equation models related to networks of timed automata as studied here may be particularly amenable to cause computation. A correspondence between these modeling formalisms has already been pointed out by the same authors [55], albeit to the even more expressive hybrid automata [2] that subsume timed automata. For real-time systems, Dierks et al. develop an automated abstraction refinement technique [19] for timed automata based on considering causal relationships [24]. Wang et al. introduce a framework for the causal analysis of component-based real-time systems [60]. Kölbl et al. follow a similar direction and propose a repairing technique of timed systems focusing on static clock bounds [47]. In a further contribution, they consider the delay values of timed systems to compute causal delay values and ranges in traces violating reachability properties [46]. Mari et al. propose an explanation technique for the violation of safety properties in real-time systems [52], their approach is based on their corresponding work on explaining discrete systems [30]. Hence, in the domain of real-time systems ours is the first technique to consider arbitrary MITL properties, i.e., safety *and* liveness, as effects, together with both actions *and* real-time delays as causes.

7 Conclusion

Based on the seminal works of Halpern and Pearl, we have proposed notions of but-for and actual causality for networks of timed automata, which define counterfactual explanations for violations of MITL specifications. Our definitions rely on the idea of counterfactual automata that represent infinitely many possible counterfactual executions. We then leveraged results on real-time model checking for algorithms that check and compute but-for and counterfactual causes, demonstrating with a prototype that our explanations significantly narrow down the root causes in counterexamples of MITL properties. Interesting directions of future work are to study symbolic causes [48, 21, 25] in real-time system, i.e., to consider real-time properties specified in MITL or an event-based logic as causes [48, 17], and to develop tools for visualizing [40] counterfactual explanations in networks of timed automata.

References

1 Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron A. Peled, editors, *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 8–22. Springer, 1999. doi:10.1007/3-540-48683-6_3.

2 Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992. doi:10.1007/3-540-57318-6_30.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

4 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, January 1996. doi:10.1145/227595.227602.

5 Uwe Aßmann, Christel Baier, Clemens Dubslaff, Dominik Grzelak, Simon Hanisch, Ardhi Putra Pratama Hartono, Stefan Köpsell, Tianfang Lin, and Thorsten Strufe. *Tactile computing: Essential building blocks for the Tactile Internet*, pages 293–317. Academic Press, 2021. 46.23.01; LK 01. doi:10.1016/B978-0-12-821343-8.00025-3.

6 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.

7 Christel Baier, Roxane van den Bossche, Sascha Klüppelholz, Johannes Lehmann, and Jakob Piribauer. Backward responsibility in transition systems using general power indices. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 20320–20327. AAAI Press, 2024. doi:10.1609/AAAI.V38I18.30013.

8 Thomas Ball and Orna Kupferman. Vacuity in testing. In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs*, pages 4–17, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-79124-9_2.

9 Thomas Ball, Mayur Naik, and Sriram K. Rajamani. From symptom to cause: Localizing errors in counterexample traces. In *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '03, pages 97–105, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/604131.604140.

10 Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard Trefler. Explaining counterexamples using causality. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, pages 94–108, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-02658-4_11.

11 Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient detection of vacuity in actl formulas. In Orna Grumberg, editor, *Computer Aided Verification*, pages 279–290, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. doi:10.1007/3-540-63166-6_28.

12 Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR '98: Concurrency Theory, 9th International Conference, Nice, France, September 8-11, 1998, Proceedings*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998. doi:10.1007/BFB0055643.

13 Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL – A tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995. doi:10.1007/BFB0020949.

14 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003. doi:10.1007/978-3-540-27755-2_3.

15 Raven Beutner, Bernd Finkbeiner, Hadar Frenkel, and Julian Siber. Checking and sketching causes on temporal sequences. 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 314–327. Springer, 2023. doi:10.1007/978-3-031-45332-8_18.

16 Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theor. Comput. Sci.*, 321(2-3):291–345, 2004. doi:10.1016/J.TCS.2004.04.003.

17 Georgiana Caltais, Sophie Linnea Guetlein, and Stefan Leue. Causality for general LTL-definable properties. *Electronic Proceedings in Theoretical Computer Science*, 286:1–15, January 2019. doi:10.4204/eptcs.286.1.

18 Hana Chockler, Joseph Y. Halpern, and Orna Kupferman. What causes a system to satisfy a specification? *ACM Trans. Comput. Logic*, 9(3), June 2008. doi:10.1145/1352582.1352588.

19 Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and Aravinda Prasad Sistla, editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/10722167_15.

20 Norine Coenen, Raimund Dachselt, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Tom Horak, Niklas Metzger, and Julian Siber. Explaining hyperproperty violations. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 407–429. Springer, 2022. doi:10.1007/978-3-031-13185-1_20.

21 Norine Coenen, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Niklas Metzger, and Julian Siber. Temporal causality in reactive systems. In Ahmed Bouajjani, Lukás Holík, and Zhilin Wu, editors, *Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings*, volume 13505 of *Lecture Notes in Computer Science*, pages 208–224. Springer, 2022. doi:10.1007/978-3-031-19992-9_13.

22 Anupam Datta, Deepak Garg, Dilsun Kaynar, Divya Sharma, and Arunesh Sinha. Program actions as actual causes: A building block for accountability. In *2015 IEEE 28th Computer Security Foundations Symposium*, pages 261–275, 2015. doi:10.1109/CSF.2015.25.

23 Alexandre David and Wang Yi. Modelling and analysis of a commercial field bus protocol. In *12th Euromicro Conference on Real-Time Systems (ECRTS 2000), 19-21 June 2000, Stockholm, Sweden, Proceedings*, pages 165–172. IEEE Computer Society, 2000. doi:10.1109/EMRTS.2000.854004.

24 Henning Dierks, Sebastian Kupferschmid, and Kim Guldstrand Larsen. Automatic abstraction refinement for timed automata. In Jean-François Raskin and P. S. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, volume 4763 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2007. doi:10.1007/978-3-540-75454-1_10.

25 Bernd Finkbeiner, Hadar Frenkel, Niklas Metzger, and Julian Siber. Synthesis of temporal causality. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part III*, volume 14683 of *Lecture Notes in Computer Science*, pages 87–111. Springer, 2024. doi:10.1007/978-3-031-65633-0_5.

26 Bernd Finkbeiner and Julian Siber. Counterfactuals modulo temporal logics. In Ruzica Piskac and Andrei Voronkov, editors, *LPAR 2023: 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, June 4-9, 2023*, volume 94 of *EPiC Series in Computing*, pages 181–204. EasyChair, 2023. doi:10.29007/qtw7.

27 Michael Gerke, Rüdiger Ehlers, Bernd Finkbeiner, and Hans-Jörg Peter. Model checking the flexray physical layer protocol. In Stefan Kowalewski and Marco Roveri, editors, *Formal Methods for Industrial Critical Systems - 15th International Workshop, FMICS 2010, Antwerp, Belgium, September 20-21, 2010. Proceedings*, volume 6371 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2010. doi:10.1007/978-3-642-15898-8_9.

28 Gregor Gössler and Daniel Le Métayer. A General Trace-Based Framework of Logical Causality. Research Report RR-8378, INRIA, October 2013. URL: https://inria.hal.science/hal-00873665.

29 Gregor Gössler, Daniel Le Métayer, and Jean-Baptiste Raclet. Causality analysis in contract violation. In Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Roşu, Oleg Sokolsky, and Nikolai Tillmann, editors, *Runtime Verification*, pages 270–284, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

30 Gregor Gössler, Thomas Mari, Yannick Pencolé, and Louise Travé-Massuyès. Towards Causal Explanations of Property Violations in Discrete Event Systems. In *DX'19 - 30th International Workshop on Principles of Diagnosis*, pages 1–8, Klagenfurt, Austria, November 2019. URL: https://inria.hal.science/hal-02369014.

31 Alex Groce. Error explanation with distance metrics. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 108–122, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-24730-2_8.

32 Gregor Gössler and Jean-Bernard Stefani. Causality analysis and fault ascription in component-based systems. *Theoretical Computer Science*, 837:158–180, 2020. doi:10.1016/j.tcs.2020.06.010.

33 Joseph Y. Halpern. A modification of the halpern-pearl definition of causality. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3022–3033. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/427.

34 Joseph Y. Halpern. *Actual Causality*. MIT Press, 2016.

35 Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part i: Causes. *The British Journal for the Philosophy of Science*, 2005.

36 Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part ii: Explanations. *The British Journal for the Philosophy of Science*, 2005.

37 Joseph Y. Halpern and Spencer Peters. Reasoning about causal models with infinitely many variables. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5668–5675, June 2022. doi:10.1609/aaai.v36i5.20508.

38 Mark Harman and Robert M. Hierons. An overview of program slicing. *Softw. Focus*, 2(3):85–92, 2001. doi:10.1002/SWF.41.

39 K. Havelund, A. Skou, K.G. Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal. In *Proceedings Real-Time Systems Symposium*, pages 2–13, 1997. doi:10.1109/REAL.1997.641264.

40 Tom Horak, Norine Coenen, Niklas Metzger, Christopher Hahn, Tamara Flemisch, Julián Méndez, Dennis Dimov, Bernd Finkbeiner, and Raimund Dachselt. Visual analysis of hyperproperties for understanding model checking results. *IEEE Trans. Vis. Comput. Graph.*, 28(1):357–367, 2022. doi:10.1109/TVCG.2021.3114866.

41 Susan B. Horwitz, Thomas Reps, and Dave Binkley. Interprocedural slicing using dependence graphs. *SIGPLAN Not.*, 23(7):35–46, June 1988. doi:10.1145/960116.53994.

42 Yatin Vasant Hoskote, Timothy Kam, Pei-Hsin Ho, and Xudong Zhao. Coverage estimation for symbolic model checking. In Mary Jane Irwin, editor, *Proceedings of the 36th Conference on Design Automation, New Orleans, LA, USA, June 21-25, 1999*, pages 300–305. ACM Press, 1999. doi:10.1145/309847.309936.

43 Felix Jahn. Prototype tool of our approach. URL: https://github.com/FelixJahnFJ/Real-Time-Causality-Tool.git.

44 Felix Jahn. Real Time Causality Analysis Tool. Software, swhId: swh:1:dir:de6b34eb1137d85c4257b5adac4b15646bd8ea3e (visited on 2024-10-11). URL: https://github.com/reactive-systems/rt-causality.git.

45 Manu Jose and Rupak Majumdar. Cause clue clauses: error localization using maximum satisfiability. In Mary W. Hall and David A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 437–446. ACM, 2011. doi:10.1145/1993498.1993550.

46 Martin Kölbl, Stefan Leue, and Robert Schmid. Dynamic causes for the violation of timed reachability properties. In Nathalie Bertrand and Nils Jansen, editors, *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings*, volume 12288 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2020. doi:10.1007/978-3-030-57628-8_8.

47 Martin Kölbl, Stefan Leue, and Thomas Wies. Clock bound repair for timed systems. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2019. doi:10.1007/978-3-030-25540-4_5.

48 Florian Leitner-Fischer and Stefan Leue. Causality checking for complex system models. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 248–267. Springer, 2013. doi:10.1007/978-3-642-35873-9_16.

49 David K. Lewis. *Counterfactuals.* Cambridge, MA, USA: Blackwell, 1973.

50 Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. In Bernhard Steffen, editor, *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, volume 1384 of *Lecture Notes in Computer Science*, pages 281–297. Springer, 1998. doi:10.1007/BFB0054178.

51 Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In Eugene Asarin and Patricia Bouyer, editors, *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings*, volume 4202 of *Lecture Notes in Computer Science*, pages 274–289. Springer, 2006. doi:10.1007/11867340_20.

52 Thomas Marix, Thao Dang, and Gregor Gössler. Explaining safety violations in real-time systems. In Catalin Dima and Mahsa Shirmohammadi, editors, *Formal Modeling and Analysis of Timed Systems - 19th International Conference, FORMATS 2021, Paris, France, August 24-26, 2021, Proceedings*, volume 12860 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2021. doi:10.1007/978-3-030-85037-1_7.

53 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 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470597.

54 Julie Parreaux, Jakob Piribauer, and Christel Baier. Counterfactual causality for reachability and safety based on distance functions. In Antonis Achilleos and Dario Della Monica, editors, *Proceedings of the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2023, Udine, Italy, 18-20th September 2023*, volume 390 of *EPTCS*, pages 132–149, 2023. `doi:10.4204/EPTCS.390.9`.

55 Spencer Peters and Joseph Y. Halpern. Causal modeling with infinitely many variables. *CoRR*, abs/2112.09171, 2021. `arXiv:2112.09171`.

56 Pyuppaal library webpage. URL: `https://pypi.org/project/pyuppaal/1.0.0/`.

57 Arshia Rafieioskouei and Borzoo Bonakdarpour. Efficient discovery of actual causality using abstraction-refinement. *CoRR*, abs/2407.16629, 2024. `doi:10.48550/arXiv.2407.16629`.

58 Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987. `doi:10.1016/0004-3702(87)90062-2`.

59 Chao Wang, Zijiang Yang, Franjo Ivančić, and Aarti Gupta. Whodunit? causal analysis for counterexamples. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis*, pages 82–95, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

60 Shaohui Wang, Anaheed Ayoub, BaekGyu Kim, Gregor Gössler, Oleg Sokolsky, and Insup Lee. A causality analysis framework for component-based real-time systems. In Axel Legay and Saddek Bensalem, editors, *Runtime Verification*, pages 285–303, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. `doi:10.1007/978-3-642-40787-1_17`.

61 Mark Weiser. Program slicing. *IEEE Transactions on Software Engineering*, SE-10(4):352–357, 1984. `doi:10.1109/TSE.1984.5010248`.

A Proofs of Section 3

▶ **Proposition 14.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n = \mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$, then for every run ρ of the network in which ϕ appears, there is a but-for cause for ϕ in ρ of $\mathcal{A}^n$, if and only if there exists a run ρ' of the network with $\rho' \not\models \phi$.*

Proof. Let ρ be a run of such a network with $\rho \models \phi$. We show both direction of the equivalence separately.

"$\Rightarrow$": Assume there is a but-for cause $\mathcal{C}$ for ϕ in ρ of $\mathcal{A}$. From $\mathbf{CF}_{\mathsf{BF}}$, we know that there exists a run $\rho' \in \Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)})\big)$ such that $\rho' \not\models \phi$. Since the components of the network are built from (trace) intersections, is easy to see that $\Pi(\mathcal{A}_i \cap \mathcal{A}^{\mathcal{C}}_{\rho(\mathcal{A}_i)}) \subseteq \Pi(\mathcal{A}_i)$ for all components $1 \leq i \leq n$. From the semantics of the network based on parallel composition, it follows that $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)})\big) \subseteq \Pi(\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n)$, from which this direction of the claim immediately follows.

"$\Leftarrow$": Let ρ' be a run of the network $\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$ with $\rho' \not\models \phi$. We show that the set of events $\mathcal{E}_\rho$, i.e., the set of *all* events appearing on the path ρ, fulfills the **SAT** and the $\mathbf{CF}_{\mathsf{BF}}$ condition: From our initial assumption, it follows that $\rho \models \phi$ and from the definition of $\mathcal{E}_\rho$ we have $\rho \models \mathcal{E}_\rho$, hence the **SAT** condition is fulfilled. From the definition of the counterfactual trace automaton, it follows that the language $\mathcal{A}^{\mathcal{E}_\rho|_i}_{\rho(\mathcal{A}_i)}$ of every component i describes all possible traces, i.e., arbitrary orderings of actions, with arbitrary delays, over the alphabet of actions Act. From this we can deduce that the runs of the network under arbitrary interventions are in fact the runs of the original network, i.e., we have $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{E}_\rho|_1}_{\rho(\mathcal{A}_1)}) \,||\ldots||\, (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{E}_\rho|_n}_{\rho(\mathcal{A}_n)})\big) = \Pi(\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n)$. Since by our initial assumption there exists a $\rho' \not\models \phi$ in $\mathcal{A}_1 \,||\ldots||\, \mathcal{A}_n$, we can deduce that $\mathbf{CF}_{\mathsf{BF}}$ is fulfilled. Finally, since $\mathcal{E}_\rho$ is finite, it either has a minimal subset that satisfies the two criteria and hence witnesses this direction of our claim, or $\mathcal{E}_\rho$ itself is the desired witness. ◀

▶ **Proposition 15.** *Given an effect ϕ and a network of timed automata $\mathcal{A}^n$, $\emptyset$ is the (unique) but-for cause for an effect ϕ on a run ρ of $\mathcal{A}^n$, if and only if there exists a run η of $\mathcal{A}^n$ with $\mathit{localize}(\rho, \mathcal{A}^n) = \mathit{localize}(\eta, \mathcal{A}^n)$ and $\eta \not\models \phi$, i.e., a run with the same local traces as the actual run, that does, however, not satisfy the effect.*

Proof. Let a network $\mathcal{A}^n = \mathcal{A}_1 \mid\mid \ldots \mid\mid \mathcal{A}_n$ be given. First up, it is easy to see that whenever $\emptyset$ is a but-for cause, it is unique: No other set $\mathcal{C} \neq \emptyset$ can satisfy **MIN**, since $\emptyset \subset \mathcal{C}$ and $\emptyset$ satisfies **SAT** and $\mathbf{CF}_{\mathsf{BF}}$ by assumption. We proceed with proving the equivalence:

"⇒": Assume that $\emptyset$ is a but-for cause on some run ρ, then from $\mathbf{CF}_{\mathsf{BF}}$ it follows that there exists a run $\rho' \in \Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ such that $\rho' \not\models \phi$. From the definition of the counterfactual trace automaton $\mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)}$ it follows that for all components i and for all $\rho_i \in \Pi(\mathcal{A}_i \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)})$ we have that $\pi^{\rho_i} = \rho(\mathcal{A}_i)$. From the definition of the localization function it then follows that for all $\zeta \in \Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ we have that $\mathit{localize}(\rho, \mathcal{A}) = \mathit{localize}(\zeta, \mathcal{A})$, so in particular for ρ', which shows this direction of the claim.

"⇐": Assume there is such an η with $\mathit{localize}(\rho, \mathcal{A}) = \mathit{localize}(\eta, \mathcal{A})$ and $\eta \not\models \phi$. It is easy to see that $\emptyset$ trivially satisfies **SAT** and **MIN**. Hence, we only need to show that $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ includes η (and indeed all runs with the same local traces as ρ). This follows from the fact that $\Pi(\mathcal{A}_i \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)})$ includes all runs η_i that have the same trace as the local projection of ρ with respect to this component, i.e., all $\eta_i = \rho(\mathcal{A}_i)$, due to the definition of $\mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_i)}$ and of trace intersection. By the definition of parallel composition, we can conclude that $\Pi\big((\mathcal{A}_1 \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\emptyset}_{\rho(\mathcal{A}_n)})\big)$ includes all ρ' with $\mathit{localize}(\rho, \mathcal{A}) = \mathit{localize}(\rho', \mathcal{A})$, hence it also includes η, which can then serve as a witness for $\emptyset$ satisfying $\mathbf{CF}_{\mathsf{BF}}$, which closes this direction of the equivalence. ◀

B Algorithms and Proofs of Section 4

In this section, we give the algorithm for checking causality and detailed proofs of the statements from Section 4. We start by proving the monotonicity properties.

▶ **Lemma 22** (Cause Monotonicity). *For every network of timed automaton $\mathcal{A}_1 \mid\mid \ldots \mid\mid TA_n$, run ρ, and effect ϕ, we have that*

1. *if a set of events $\mathcal{C}$ fulfills* **SAT** *also every subset $\mathcal{C}' \subseteq \mathcal{C}$ fulfills* **SAT**.
2. *if a set of events $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$ (fulfills $\mathbf{CF}_{\mathsf{Act}}$) also every superset $\mathcal{C}' \supseteq \mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$ (fulfills $\mathbf{CF}_{\mathsf{Act}}$).*

Proof. We show the two statements separately:

1. Follows by the transitivity of set inclusions: If $\mathcal{C}$ fulfills **SAT**, we have that $\rho \models \mathcal{C}$ and $\rho \models E$. Hence, $\mathcal{C}' \subseteq \mathcal{C} \subseteq \mathcal{E}_\rho$ and therefore $\rho \models \mathcal{C}'$ such that also $\mathcal{C}'$ fulfills **SAT**.
2. Let $\mathcal{C}$ fulfill $\mathbf{CF}_{\mathsf{BF}}$, that is, there is a counterfactual run ρ' of $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)})$ with $\rho \not\models \phi$. Now notice that for $\mathcal{C}' \supseteq \mathcal{C}$, also the transition relation of each counterfactual trace automaton of $\mathcal{C}'$ is a superset of the one of $\mathcal{C}$ such that we also have $\Pi(\mathcal{A}^{\mathcal{C}'|_i}_{\rho(\mathcal{A}_i)}) \supseteq \Pi(\mathcal{A}^{\mathcal{C}|_i}_{\rho(\mathcal{A}_i)})$. Therefore, ρ' is also a run of $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}'|_1}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}'|_n}_{\rho(\mathcal{A}_n)})$ such that $\mathcal{C}'$ fulfills $\mathbf{CF}_{\mathsf{BF}}$. The proof for $\mathbf{CF}_{\mathsf{Act}}$ works analogously for the run in intersection of the contingency and counterfactual trace automata. ◀

Algorithm 2 decides whether a given set of events forms a but-for cause. It is a straightforward implementation of Definition 12 of but-for causality under the use of monotonicity for accelerating the verification of the **MIN** condition. Hence, we do not give a detailed proof of correctness for Algorithm 2 and continue directly with cause computation.

Algorithm 2 Checking But-For Cause.

```
Input: network $\mathcal{A} = \mathcal{A}_1 \mid\mid \ldots \mid\mid \mathcal{A}_n$, run $\rho$, effect $\phi$, set of events $\mathcal{C}$
Output: “Is $\mathcal{C}$ a but-for cause for $\phi$ in $\rho$ of $\mathcal{A}$?”
if $\rho \not\models \phi$ or $\mathcal{C} \not\subseteq \mathcal{E}_\rho$ then                    // checking SAT
    return false
end
if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \models \phi$ then   // checking CF_BF
    return false
end
for event $e \in \mathcal{C}$ do                                    // checking MIN
    $C' := \mathcal{C} \setminus \{e\}$
    if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}'|_1}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}'|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$ then
        return false
    end
end
return true
```

▶ **Theorem 23.** *Algorithm 1 is sound and complete, i.e., it terminates with*

$$\textit{Compute But-For Causes}(\mathcal{A}, \rho, \phi) = \{\mathcal{C} \,|\, \mathcal{C} \text{ is a but-for cause for } \phi \text{ in } \rho \text{ of } \mathcal{A}\},$$

for all networks $\mathcal{A} = \mathcal{A}_1 \mid\mid \ldots \mid\mid \mathcal{A}_n$ and runs ρ of $\mathcal{A}$ satisfying an effect ϕ.

Proof. We argue for soundness ($\subseteq$) and completeness ($\supseteq$) separately:

“$\supseteq$”: Let $\mathcal{C}$ be a but-for cause for ϕ in ρ of $\mathcal{A}$, i.e. fulfilling **SAT**, $\mathbf{CF}_{\mathsf{BF}}$, **MIN**. We first notice that the algorithm does then not remove $\mathcal{C}$ from *Power* (until it may be added to Res_s): $\mathcal{C}$ is not removed by Line 6 since the minimality of $\mathcal{C}$ implies that it has no subset fulfilling $\mathbf{CF}_{\mathsf{BF}}$; and $\mathcal{C}$ is not removed by Line 13 since the monotonicity of $\mathbf{CF}_{\mathsf{BF}}$ implies that it has no superset not fulfilling $\mathbf{CF}_{\mathsf{BF}}$. Now since $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$, if $|\mathcal{C}| \leq \frac{\mathcal{E}_\rho}{2}$, $\mathcal{C}$ is added to Res_s in Line 5, if $|\mathcal{C}| > \frac{\mathcal{E}_\rho}{2}$ it is added to Res_l in Line 11 and is, in addition, not removed in the last line as $\mathcal{C}$ fulfills the **MIN** condition. Therefore, $\mathcal{C}$ is returned by Compute But-For Causes$(\mathcal{A}, \rho, \phi)$.

“$\subseteq$”: Let $\mathcal{C} \in$ Compute But-For Causes$(\mathcal{A}, \rho, \phi)$. As for all set of events considered by the algorithm, we have $\mathcal{C} \in \mathcal{P}(\mathcal{E}_\rho)$ and, hence, $\mathcal{C} \subseteq \mathcal{E}_\rho$ such that $\mathcal{C}$ fulfills **SAT**. By definition of the algorithm, $\mathcal{C}$ is only returned as a result when it was added to Res_s or Res_l. This, in turn, is only the case, if $(\mathcal{A}_1 \cap \mathcal{A}^{\mathcal{C}|_1}_{\rho(\mathcal{A}_1)}) \mid\mid \ldots \mid\mid (\mathcal{A}_n \cap \mathcal{A}^{\mathcal{C}|_n}_{\rho(\mathcal{A}_n)}) \not\models \phi$. Therefore, $\mathcal{C}$ fulfills $\mathbf{CF}_{\mathsf{BF}}$. Lastly to establish the **MIN** condition, we have to show that there are no proper subsets of $\mathcal{C}$ that fulfill **SAT** and $\mathbf{CF}_{\mathsf{BF}}$. Towards a contradiction, lets assume there are such subsets and let $\mathcal{C}' \subsetneq \mathcal{C}$ be the minimal one. Then, $\mathcal{C}'$ is but-for cause and by the first inclusion $\mathcal{C}' \in$ Compute But-For Causes$(\mathcal{A}, \rho, \phi)$. Now, if $\mathcal{C}$ was returned by the algorithm since $\mathcal{C} \in Res_s$, then $|\mathcal{C}'| < |\mathcal{C}|$ implies that the algorithm has considered $\mathcal{C}'$ earlier. From this point, however, $\mathcal{C} \notin Power$, a contradiction. If $\mathcal{C}$ was returned since $\mathcal{C} \in Res_l$, the filtering in Line 17 results in a contradiction. Therefore, $\mathcal{C}$ is a but-for cause for ϕ in ρ of $\mathcal{A}$. ◀

▶ **Theorem 24.** *Checking and computing causes for an effect ϕ on the run ρ in a network of timed automata $\mathcal{A}$ is* EXPSPACE(ϕ)*-complete.*

Proof. Analyzing the computational compexity of Algorithms 1 and 2 shows the two problems of cause checking and computations to be solvalbe in $\mathsf{EXPSPACE}(\phi)$. For showing $\mathsf{EXPSPACE}(\phi)$-hardness, we present a reduction from the model checking problem, that is EXPSPACE-complete [4]. We construct for a timed automaton $\mathcal{A} = (Q, q_0, X, E, I, L)$ an extended reduction automaton $\mathcal{A}_{\mathsf{red}} := (Q \,\dot{\cup}\, \{s_{\mathsf{new}}, q_{\mathsf{new}}\}\,,\, s_{\mathsf{new}}\,,\, X \,\dot{\cup}\, \{x_{\mathsf{new}}\}\,,\, E'\,,\, I'\,,\, L')$ over an extended set of actions $Act \,\dot{\cup}\, \{\alpha_{\mathsf{new}}, \beta_{\mathsf{new}}\}$ and labels $AP \,\dot{\cup}\, \{p_{\mathsf{new}}\}$ whereby s_{new} and q_{new} are fresh locations, α_{new} and β_{new} are fresh actions, x_{new} is a fresh clock, p_{new} is a fresh atomic proposition, and we have

$$E' \;:=\; E \cup \{(s_{\mathsf{new}}, \top, \alpha_{\mathsf{new}}, \{x_{\mathsf{new}} := 1\}, q_{\mathsf{new}})\,,\, (q_{\mathsf{new}}, \top, \alpha_{\mathsf{new}}, \emptyset, q_{\mathsf{new}})\,,\, (s_{\mathsf{new}}, \top, \beta_{\mathsf{new}}, \epsilon, q_0)\},$$

$$I'(q) := \begin{cases} I(q), & q \in Q, \\ x_{\mathsf{new}} \leq 0, & q = s_{\mathsf{new}}, \\ x_{\mathsf{new}} \leq 1, & q = q_{\mathsf{new}}, \end{cases} \qquad \text{and} \qquad L'(q) := \begin{cases} L(q), & q \in Q, \\ \{\,\}, & q = s_{\mathsf{new}}, \\ \{p_{\mathsf{new}}\}, & q = q_{\mathsf{new}}. \end{cases}$$

That is, $\mathcal{A}_{\mathsf{red}}$ is an extension of $\mathcal{A}$ that has a new initial location s_{new} from which a direct transition (delay of 0) to either a second new location q_{new} or to the initial state of the original automaton $\mathcal{A}$ is enforced. This new automaton has a new run, namely $\rho_{\mathsf{red}} := (s_{\mathsf{new}}, u_0) \xrightarrow{0.0} \xrightarrow{\alpha_{\mathsf{new}}} (q_{\mathsf{new}}, u_0) \xrightarrow{1.0} \xrightarrow{\alpha_{\mathsf{new}}})^{\omega}$ fulfilling the effect $\phi_{\mathsf{red}} := \phi \vee \Diamond\, p_{\mathsf{new}}$.

Instances $(\mathcal{A}, \phi)$ of the model checking problem are now mapped to instances of the cause checking problem via the reduction $r : (\mathcal{A}, \phi) \mapsto (\mathcal{A}_{\mathsf{red}},\ \rho_{\mathsf{red}},\ \phi_{\mathsf{red}},\ \mathcal{C}_{\mathsf{red}})$ with $\mathcal{C}_{\mathsf{red}} := \{((\alpha_{\mathsf{new}}, 1, \rho(\mathcal{A}_{\mathsf{red}})\}$. Now, we have that $\mathcal{A} \not\models \phi$ iff $\mathcal{C}_{\mathsf{red}}$ is a cause for ϕ_{red} in ρ_{red} of $\mathcal{A}_{\mathsf{red}}$. ◀

C Contingency Automaton

In this section, we illustrate the contingency automaton construction from Example 20. For automaton $\mathcal{A}_1$, run ρ from Subsection 1.1 and its sequence of local locations $\mathsf{loc}(\rho, \mathcal{A}_1) = \mathsf{init}, \mathsf{crit}, \mathsf{init}$, the location contingency automaton $\mathcal{A}_1^{\mathsf{loc}(\rho)}$ is depicted in Figure 4. Following Definiton 16, the contingency automaton is constructed in the following way:

- we copy the automaton $|\rho(\mathcal{A}_1)|$ times, to encode the current step in the states (second component of the tuple);
- we redirect the transitions from the original automata (black transitions) to their target location in the next copy;
- in each step, we add contingency transitions (red transitions), allowing the location to be reset to what it had been in the corresponding step of the original run ρ.

We can now find a counterfactual run in $\mathcal{A}_1^{\mathsf{loc}(\rho)}$ avoiding the critical section by taking the contingency from $(\mathsf{init}, 1) \xrightarrow{\beta} (\mathsf{init}, 2)$. That is, the location after the second transition is reset to what it had been in the original run, namely to location init. The construction of the clock contingency automaton works in a similar way: We allow additional transitions to reset the clocks as they had been in the original run at the respective positions.

D Experimental Setup and Results for Fischer's Protocol

We report on the details in the experimental evaluation for Fischer's protocol and the causes identified by the tool. Fischer's protocol is a popular real-time mutual exclusion protocol, we depict one component $\mathcal{A}_i$ in Figure 5. We then test the effect $\phi := \neg\Box_{[0,\infty)} \neg\mathsf{crit}_1$ on the network $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$, i.e. that the first component reaches its critical section.

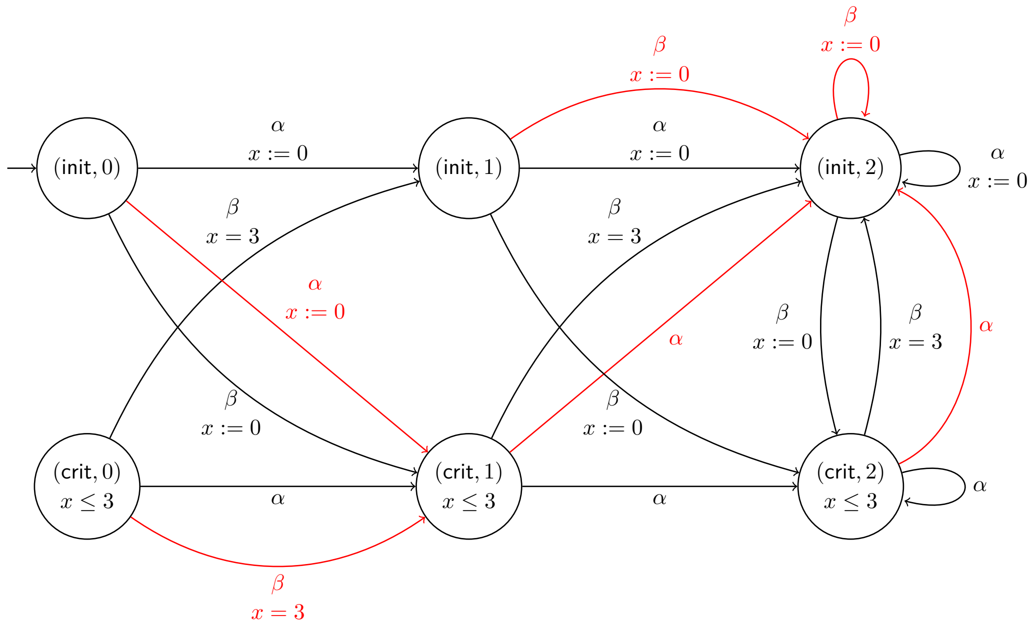


Figure 4 The contingency automaton $\mathcal{A}_1^{\mathsf{loc}(\rho)}$ from Example 20.

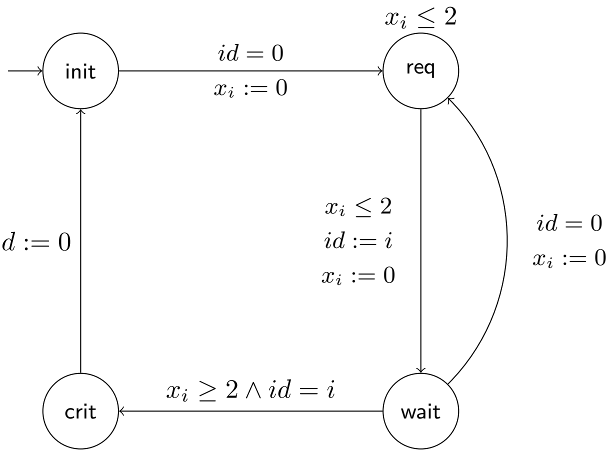


Figure 5 A single component automaton $\mathcal{A}_i$ of Fischer's protocol network $\mathcal{A}_1 \,||\, \ldots \,||\, \mathcal{A}_n$.

We state the tested runs and results exemplary for $n = 2$:

$$\rho_1 := \Big(\{\mathsf{init}_{1,2}\} \xrightarrow[1.0]{\tau_1} \{\mathsf{req}_1, \mathsf{init}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{req}_1, \mathsf{init}_2\} \xrightarrow[4.0]{\tau_1} \{\mathsf{wait}_1, \mathsf{init}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{crit}_1, \mathsf{init}_2\} \xrightarrow[2.0]{\tau_1} \Big)^{\omega}$$

$$\rho_2 := \{\mathsf{init}_{1,2}\} \xrightarrow[1.0]{\tau_2} \Big(\{\mathsf{init}_1, \mathsf{req}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{req}_1, \mathsf{req}_2\} \xrightarrow[1.0]{\tau_2} \{\mathsf{req}_1, \mathsf{wait}_2\}$$

$$\xrightarrow[1.0]{\tau_1} \{\mathsf{wait}_1, \mathsf{wait}_2\} \xrightarrow[3.0]{\tau_1} \{\mathsf{crit}_1, \mathsf{wait}_2\} \xrightarrow[1.0]{\tau_1} \{\mathsf{init}_1, \mathsf{wait}_2\} \xrightarrow[1.0]{\tau_2} \Big)^{\omega}$$

The detected causes for those two runs are reported in Table 3. As in Fischer's protocol only internal actions are used, the detected root causes only consist out of delay actions. Further, since we do not encounter cases of preemption, but-for and actual causes agree on this example. For $n = 3, 4$ we tested runs with the same lasso-part.

Table 3 Overview of the root causes found in the experiments for Fischer's protocol.

Ref.	ρ_1**: BF Causes**	ρ_1 **: Actual Causes**	ρ_2**: BF Causes**	ρ_2 **: Actual Causes**
1	$\{(1.0, 1, \mathcal{A}_1)\}$	$\{(1.0, 1, \mathcal{A}_1)\}$	$\{(1.0, 1, \mathcal{A}_2)\}$	$\{(1.0, 1, \mathcal{A}_2)\}$
2	$\{(4.0, 3, \mathcal{A}_1)\}$	$\{(4.0, 3, \mathcal{A}_1)\}$	$\{(2.0, 1, \mathcal{A}_1)\}$	$\{(2.0, 1, \mathcal{A}_1)\}$
3			$\{(2.0, 2, \mathcal{A}_2)\}$	$\{(2.0, 2, \mathcal{A}_2)\}$
4			$\{(2.0, 2, \mathcal{A}_1)\}$	$\{(2.0, 2, \mathcal{A}_1)\}$
5			$\{(3.0, 3, \mathcal{A}_1),$ $(6.0, 3, \mathcal{A}_2)\}$	$\{(3.0, 3, \mathcal{A}_1),$ $(6.0, 3, \mathcal{A}_2)\}$

Oblivious Complexity Classes Revisited: Lower Bounds and Hierarchies

Karthik Gajulapalli
Georgetown University, Washington, DC, USA

Zeyong Li
National University of Singapore, Singapore

Ilya Volkovich
Boston College, MA, USA

Abstract

In this work we study *oblivious* complexity classes. These classes capture the power of interactive proofs where the prover(s) are only given the input size rather than the actual input. In particular, we study the connections between the symmetric polynomial time – $\mathsf{S_2P}$ and its oblivious counterpart – $\mathsf{O_2P}$. Among our results:

- For each $k \in \mathbb{N}$, we construct an explicit language $L_k \in \mathsf{O_2P}$ that cannot be computed by circuits of size n^k.
- We prove a hierarchy theorem for $\mathsf{O_2TIME}$. In particular, for any time constructible function $t : \mathbb{N} \to \mathbb{N}$ and any $\varepsilon > 0$ we show that: $\mathsf{O_2TIME}[t(n)] \subsetneq \mathsf{O_2TIME}[t(n)^{1+\varepsilon}]$.
- We prove new structural results connecting $\mathsf{O_2P}$ and $\mathsf{S_2P}$.
- We make partial progress towards the resolution of an open question posed by Goldreich and Meir (TOCT 2015) that relates the complexity of NP to its oblivious counterpart - ONP.
- We identify a natural class of problems in $\mathsf{O_2P}$ from computational Ramsey theory, that are not expected to be in P or even BPP.

To the best of our knowledge, these results constitute the first explicit fixed-polynomial lower bound and hierarchy theorem for $\mathsf{O_2P}$. The smallest uniform complexity class for which such lower bounds were previously known was $\mathsf{S_2P}$ due to Cai (JCSS 2007). In addition, this is the first uniform hierarchy theorem for a semantic class. All previous results required some non-uniformity. In order to obtain some of the results in the paper, we introduce the notion of *uniformly-sparse extensions* which could be of independent interest.

Our techniques build upon the de-randomization framework of the powerful Range Avoidance problem that has yielded many new interesting explicit circuit lower bounds.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness; Theory of computation → Circuit complexity; Theory of computation → Interactive proof systems; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases fixed circuit lower bounds, semantic time hierarchy, oblivious complexity, range avoidance

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.23

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2024/049/`

Acknowledgements The authors would like to thank Alexander Golovnev, William Gasarch and Edward Hirsch for many helpful discussions and feedback on an earlier draft of this manuscript. The authors would also like to thank the anonymous referees for useful comments.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 23; pp. 23:1–23:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Proving circuit lower bounds has been one of the holy grails of theory of computation with strong connections to many fundamental questions in complexity theory. For instance, proving that there exists a function in E[1] that requires exponential-size circuits would entail a strong derandomization: $\mathsf{BPP} = \mathsf{P}$ and $\mathsf{MA} = \mathsf{NP}$ [46, 32]. And yet, while by counting arguments (i.e. [52]) the vast majority of Boolean functions/languages do require exponential-size circuits, the best "explicit" lower bounds are still linear! (in fact the best known lower bound for any language in E^{NP} is just linear [39]). Indeed, although it is widely *believed* that NP requires super-polynomial-size circuits (i.e. $\mathsf{NP} \not\subseteq \mathsf{P/poly}$) establishing the statement even for NEXP (i.e. $\mathsf{NEXP} \not\subseteq \mathsf{P/poly}$), the exponential version of NP, has remained elusive for many years. The best known explicit lower bound is due to a seminal work of Williams [54], where it was shown that NEXP requires super-polynomial-size circuits in a "very" restricted model ($\mathsf{NEXP} \not\subseteq \mathsf{ACC}^0$).

In the high-end regime, Kannan [33] has shown that the exponential hierarchy requires exponential-size circuits, via diagonalization[2]. More precisely, it was shown that the class $\Sigma_3\mathsf{E} \cap \Pi_3\mathsf{E}$ contains a language that cannot be computed by a circuit family of size $2^n/n$. This result was later improved to $\Delta_3\mathsf{E} = \mathsf{E}^{\Sigma_2\mathsf{P}}$ by Miltersen, Vinodchandran and Watanabe [45]. Moreover, it was shown that $\Delta_3\mathsf{E}$ actually requires circuits of "maximum possible" size. Subsequently, the status of the problem remained stagnant for more than two decades until very recently, Chen, Hirahara and Ren [13] and a follow-up work by Li [41] improved the result to $\mathsf{S}_2\mathsf{E}$ [3]. In particular, this result was obtained via solving the Range Avoidance (Avoid) problem with "single-valued, symmetric polynomial-time" algorithm. Indeed, the focus of our work is on "oblivious" symmetric polynomial time and related complexity classes.

1.1 Background

1.1.1 Symmetric Time

Symmetric polynomial time, denoted by $\mathsf{S}_2\mathsf{P}$, was introduced independently by Canetti [9], and Russell and Sundaram [49]. Intuitively speaking, this class captures the interaction between an efficient (polynomial-time) verifier V and two all-powerful provers: the "YES"-prover Y and the "NO"-prover Z, exhibiting the following behaviour:

- If x is a yes-instance, then the "YES"-prover Y can send an irrefutable proof/certificate y to V that will make V *accept*, **regardless** of the communication from Z.
- Likewise, if x is a no-instance, then the "NO"-prover can send an irrefutable proof/certificate proof z to V that will make V *reject*, **regardless** of the communication from Y.

We stress that in both cases the irrefutable certificates can depend on x itself. One can also define $\mathsf{S}_2\mathsf{E}$ - the exponential version of $\mathsf{S}_2\mathsf{P}$, by allowing the verifier to run in linear-exponential time. For a formal definition see Definition 8. A seminal result of [7] provides the best known upper bound $\mathsf{S}_2\mathsf{P} \subseteq \mathsf{ZPP}^{\mathsf{NP}}$. At the same time, $\mathsf{S}_2\mathsf{P}$ appears to be a very powerful class as it contains MA and $\Delta_2\mathsf{P} = \mathsf{P}^{\mathsf{NP}}$.

[1] Deterministic time $2^{O(n)}$.

[2] In fact, this argument could be viewed as solving an instance of the *Range Avoidance* problem. See below.

[3] Symmetric exponential time. Indeed, $\mathsf{S}_2\mathsf{E} \subseteq \Sigma_2\mathsf{E} \cap \Pi_2\mathsf{E} \subseteq \Delta_3\mathsf{E}$. For a formal definition see Definition 8.

1.1.2 Oblivious Complexity Classes

The study of oblivious complexity classes was initiated in [10] and has subsequently received more attention [2, 22, 11, 27]. Roughly speaking, let Λ be a complexity class such that in addition to the input x, the machines $M(x, w)$ of Λ also take a witness w (and possibly other inputs). Examples of such classes include: $\mathsf{NP}, \mathsf{MA}, \mathsf{S_2P}$, etc. The corresponding *oblivious* version of Λ is obtained by stipulating that the for every $n \in \mathbb{N}$ there exists a "common" witness w_n for **all** the "respective" inputs of length n. For instance, a language L belongs to ONP – the oblivious version of NP, if there exists a polynomial-time machine $M(x, w)$ such that:

1. $\forall n \in \mathbb{N}$ there exists w_n such that $\forall x \in \{0,1\}^n : x \in L \implies M(x, w_n) = 1$.
2. $x \notin L \implies \forall w : M(x, w) = 0$.

Thus, in a similar manner, one can define the class $\mathsf{O_2P}$ – the oblivious version of $\mathsf{S_2P}$, that is referred to as "oblivious symmetric polynomial time" in the literature. $\mathsf{O_2P}$ has the additional requirement that for every $n \in \mathbb{N}$ there exist an irrefutable yes-certificate y^* and an irrefutable no-certificate z^* for all the yes-instances and the no-instances of length n, respectively. For a formal definition, see Definition 10.

It is immediate from the definitions that $\mathsf{ONP} \subseteq \mathsf{NP}$, $\mathsf{O_2P} \subseteq \mathsf{S_2P}$ and $\mathsf{ONP} \subseteq \mathsf{O_2P}$. On the other hand, by hard-coding the witnesses/certificates we get that $\mathsf{RP} \subseteq \mathsf{ONP} \subseteq \mathsf{O_2P} \subseteq \mathsf{P/poly}$. In addition, it was also observed in [10] that $\mathsf{O_2P}$ is *self-low*. That is $\mathsf{O_2P}^{\mathsf{O_2P}} = \mathsf{O_2P}$. While the oblivious classes seem to be more restricted than their non-oblivious counterparts, proving any non-trivial upper bounds could still be challenging. In terms of lower bounds, the best known containment of a non-oblivious class is $\mathsf{BPP} \subseteq \mathsf{O_2P}$ [4]. For more details and discussion see [10, 27]. Nonetheless, to the best of our knowledge, no "natural" problem for $\mathsf{O_2P}$ (or even ONP), known to lie outside of BPP, has been identified in the literature.

1.1.3 Sparsity

A language L is *sparse*, if for every input length $n \in \mathbb{N}$ the number of yes-instances of size n is at most $\text{poly}(n)$. We will denote the class of all sparse languages by SPARSE. Sparse languages have seen many applications in complexity theory. Perhaps, the most fundamental one is known as "Mahaney's theorem" [43] that implies that a sparse language cannot be NP-hard, unless $\mathsf{P} = \mathsf{NP}$. In [22] and [27], sparse languages were also studied in the context of oblivious complexly classes. In particular, they observed that $\mathsf{NP} \cap \mathsf{SPARSE} \subseteq \mathsf{ONP}$. That is, every sparse NP language is also in ONP. The same argument also implies that $\mathsf{NE} = \mathsf{ONE}$, that is, **equality** between the exponential versions of NP and ONP, respectively. Given the former claim we observe that the *Grid Coloring* problem, defined in [3], constitutes a natural ONP (and hence $\mathsf{O_2P}$) problem. For a formal statement, see Observation 6.

Subsequently, Goldreich and Meir [27] posed an open question whether a similar relation holds true for coNP and coONP. That is, whether every sparse coNP language is also in coONP[5]. Motivated by this question, we observe that essentially the same issues arise when one attempts to show that every sparse $\mathsf{S_2P}$ language is also in $\mathsf{O_2P}$. While we do not accomplish this task, we make a partial progress by introducing *uniformly-sparse extensions*. The intuition behind this definition is to have a uniform "cover" of the segments of the yes-instances for **all** input lengths. For a formal definition see Definition 25. This is our main

[4] One way to see this is by observing that $\mathsf{BPP} \subseteq \mathsf{RP}^{\mathsf{ONP}}$ and then using the self-lowness of $\mathsf{O_2P}$.

[5] The original (equivalent) formulation of the question in [27] was w.r.t NP and co-sparse languages.

conceptual contribution. As a corollary, we obtain that $\mathsf{S_2E} = \mathsf{O_2E}$. Although this might not be a new result, to the best of our knowledge, this result has not appeared in the literature previously.

1.1.4 Range Avoidance

The study of the Range Avoidance problem (Avoid) was initiated in [35]. The problem itself takes an input-expanding Boolean Circuit $\mathsf{C} : \{0,1\}^n \to \{0,1\}^{n+1}$ as input and asks to find an element y, outside the range of C. Since its introduction, there has been a steady line of exciting work studying the complexity and applications of Avoid [37, 28, 48, 15, 24, 31, 17, 13, 41, 16, 38].

Informally, Avoid algorithmically captures the probabilistic method where the existence of an object with some property follows from a union bound. In particular, Korten [37] showed that solving Avoid (deterministically) would result in finding optimal explicit constructions of many important combinatorial objects, including but not limited to Ramsey graphs [47], rigid matrices [28, 24], pseudorandom generators [14], two-source extractors [12, 40], linear codes [28], strings with maximum time-bounded Kolmogorov complexity (K^{poly}-random strings) [48] and truth tables of high circuit complexity [37].

The connection between Avoid and hard truth table makes it relevant to the study of circuit lower bounds. It has been observed and pointed out in many prior works (see, e.g. [13]) that proving explicit circuit lower bounds is effectively finding single-valued[6] constructions of hard truth tables. Indeed, this is the framework adopted for proving circuit lower bounds in [15, 13, 41]: Designing a single-valued algorithm for solving Avoid.

1.1.5 Time Hierarchy Theorem

Time Hierarchy theorems are among the most fundamental results in computational complexity theory, which (loosely speaking) assert that computation with more time is strictly more powerful. Time hierarchy theorems are known for deterministic computation (DTIME) [29, 30] and non-deterministic computation (NTIME) [18, 51, 55] which are syntactic classes. The situation for semantic classes such as BPTIME is much more elusive as it is unclear how to enumerate and simulate all BPTIME machines while ensuring that the simulating machine itself remains a proper BPTIME machine. In fact, even verifying that a machine is a BPTIME machine is itself an undecidable problem. For BPTIME, a time hierarchy theorem is only known for its promise version, or when given one bit of advice [5, 20, 21]. This was further generalized in [44], where they show most semantic classes (e.g. MA, $\mathsf{S_2TIME}$) admit a time hierarchy theorem with one bit of advice.

Along a different line of research, it was shown in [42, 19] that coming up with a pseudodeterministic algorithm (single-valued randomized algorithms) for estimating the acceptance probability of a circuit would imply a uniform hierarchy theorem for BPTIME.

1.2 Previous Results

A parallel line of work focused on the "low-end" regime by proving the so-called "fixed-polynomial" circuit lower bounds. That is, the goal is to show that for every $k \in \mathbb{N}$ there is a language L_k (that may depend on k) which cannot be computed by circuits of size

[6] Roughly speaking, a single-valued algorithm on successful executions should output a fixed (canonical) solution given the same input.

n^k. The first result in this sequel – fixed-polynomial lower bounds for the polynomial hierarchy, was obtained by Kannan [33] via diagonalization. In particular, it was shown that for every $k \in \mathbb{N}$ there exists a language $L_k \in \Sigma_4\mathsf{P}$ that cannot be computed by circuits of size n^k. This result was then improved to $\Sigma_2\mathsf{P}$. The key idea behind this and, in fact, the vast majority of subsequent improvements is a "win-win" argument that relies on the *Karp-Lipton* collapse theorem [34]: if NP has polynomial-size circuits (i.e $\mathsf{NP} \subseteq \mathsf{P/poly}$) then the (whole) polynomial hierarchy collapses to $\Sigma_2\mathsf{P}$. More specifically, the argument proceeds by a two-pronged approach:

- Suppose $\mathsf{NP} \not\subseteq \mathsf{P/poly}$. Then the claim follows as $\mathsf{NP} \subseteq \Sigma_2\mathsf{P}$.
- On the other hand, suppose $\mathsf{NP} \subseteq \mathsf{P/poly}$. Then by Karp-Lipton: $\Sigma_4\mathsf{P} = \Sigma_2\mathsf{P}$ and in particular for all $k \in \mathbb{N} : L_k \in \Sigma_2\mathsf{P}$.

Indeed, by deepening the collapse, the result was further improved to $\mathsf{ZPP}^{\mathsf{NP}}$ [36, 6], $\mathsf{P}^{\mathsf{prMA}}$ [11] and $\mathsf{S_2P}$ [7]. By using different versions of the Karp-Lipton theorem, the result has also been extended to PP [53, 1] and $\mathsf{MA}/1$ [50].

Yet, despite the success of the "win-win" argument, the obtained lower bounds are often non-explicit due to the non-constructiveness nature of the argument. Different results [8, 50] were required to exhibit explicit "hard" languages in $\Sigma_2\mathsf{P}, \mathsf{PP}$ and $\mathsf{MA}/1$. Nonetheless, the last word about $\mathsf{S_2P}$ is yet to be said. For instance, we know that there is a language in $\mathsf{S_2P}$ that requires circuits of size, say, n^2 from such arguments. However, prior to the results of [13, 41], one could not prove any super-linear lower bound for **any** particular language in $\mathsf{S_2P}$. Another limitation of the "win-win" argument stems from the fact that it only applies to complexity classes which (provably) contain NP. In particular, in [10] it was actually shown that if $\mathsf{NP} \subseteq \mathsf{P/poly}$ then the polynomial hierarchy collapses all the way to $\mathsf{O_2P}$! Unfortunately, this result does not immediately imply fixed-polynomial lower bounds for $\mathsf{O_2P}$[7] as it is unknown and, in fact, *unlikely* that $\mathsf{O_2P}$ contains NP. Furthermore, such a containment will be "self-defeating". Recall that $\mathsf{O_2P} \subseteq \mathsf{P/poly}$. Hence, if $\mathsf{NP} \subseteq \mathsf{O_2P}$ then $\mathsf{NP} \subseteq \mathsf{P/poly}$ which in and of itself already implies the collapse of the whole polynomial hierarchy!

Finally, it is important to mention a result of [22] that for any $k \in \mathbb{N}$, NP has circuits of size n^k iff $\mathsf{ONP}/1$ does. In that sense ONP already nearly captures the hardness of showing fixed-polynomial lower bounds for NP.

1.3 Our Results

In our first result we extend the lower bounds for $\mathsf{S_2P}$ and $\mathsf{S_2E}$, to their weaker oblivious counterparts $\mathsf{O_2P}$ and $\mathsf{O_2E}$, respectively. This result follows the recent line of research that obtains circuit lower bounds by means of *deterministically* solving (i.e. derandomizing) instances of the Range Avoidance problem [15, 13, 41].

▶ **Theorem 1.** *For all $k \in \mathbb{N}$, $\mathsf{O_2P} \not\subseteq \mathsf{SIZE}[n^k]$. Moreover, for each k there exists an explicit language $L_k \in \mathsf{O_2P}$ such that $L_k \notin \mathsf{SIZE}[n^k]$.*

In fact we prove a stronger parameterized version of this statement (see Theorem 29, Corollary 36, and Corollary 30). We now highlight three main reasons why such a result is fascinating:

[7] Indeed, the authors in [10] could only obtain fixed-polynomial lower bounds for $\mathsf{NP}^{\mathsf{O_2P}}$ which was later subsumed by the results of [50].

1. Our lower bound does not follow the framework of “win-win” style Karp-Lipton collapses. As was mentioned above, since already $\mathsf{O_2P} \subseteq \mathsf{P/poly}$ the pre-requisite for proving the bound via the “win-win” argument will be self-defeating.
2. Our proof is constructive and for every $k \in \mathbb{N}$ we define an explicit language $L_k \in \mathsf{O_2P}$ for which $L_k \not\subseteq \mathsf{SIZE}[n^k]$.
3. $\mathsf{O_2P}$ becomes the smallest uniform complexity class known for which fixed-polynomial lower bounds are known. Moreover, after more than 15 years, this class coincides again with the deepest known collapse result of the Karp-Lipton Theorem[8].

Our second result gives a hierarchy theorem for $\mathsf{O_2TIME}$.

▶ **Theorem 2.** *For any time constructible function $t : \mathbb{N} \to \mathbb{N}$ such that $t(n) \geq n$ and any $\varepsilon > 0$ it holds that:* $\mathsf{O_2TIME}[t(n)] \subsetneq \mathsf{O_2TIME}[t(n)^{1+\varepsilon}]$.

We remark, that to the best of our knowledge, this is the first known hierarchy theorem for a uniform semantic class (that contains BPTIME). At the same time, we observe that the proof of the non-deterministic time hierarchy theorem (NTIME) (see e.g. [55]) actually extends to the *oblivious* non-deterministic time (ONTIME) since the hard language constructed in their proof is unary and hence is contained in ONTIME. On the other hand, that same language also diagonalizes against **all** NTIME machines which is a superset of all ONTIME machines.

In our time hierarchy theorem for $\mathsf{O_2TIME}$, which goes through a reduction to Avoid, one can view Avoid as a tool for diagonalization against all circuits of fixed size, which in turn contains all $\mathsf{O_2TIME}$ machines with bounded time complexity. This (together with the time hierarchy theorem for ONTIME) might suggest an approach for proving time hierarchy theorem for semantic classes in general: diagonalize against a syntactic class that encompasses the semantic class in consideration.

Finally, we introduce the notion of *uniformly-sparse extensions* (for a formal definition, see Definition 25) to get structural complexity results relating $\mathsf{O_2TIME}$ and $\mathsf{S_2TIME}$. This relation provides an alternate method of proving Theorem 1.

▶ **Theorem 3.** *Let $L \in \mathsf{S_2P}$. If L has a* uniformly-sparse extension *then $L \in \mathsf{O_2P}$.*

While not much was known between the classes $\mathsf{O_2TIME}$ and $\mathsf{S_2TIME}$, except that $\mathsf{O_2TIME} \subseteq \mathsf{S_2TIME}$, we show new connections between the two classes. In fact, we prove a stronger parameterized version of Theorem 3 that yields as a corollary a proof of the equivalence $\mathsf{S_2E} = \mathsf{O_2E}$ (see Corollary 37). Going back to the original motivation, by repeating the same argument, we make a partial progress towards the resolution of the open question posed by Goldreich and Meir in [27]. See Lemma 35 for more details.

▶ **Theorem 4.** *Let $L \in \mathsf{coNP}$. If L has a* uniformly-sparse extension *then $L \in \mathsf{coONP}$.*

Finally, we observe that computational Ramsey theory provides some very natural problems in ONP (and hence $\mathsf{O_2P}$). As an example, we re-introduce the grid coloring problem below. While Claim 2.5 in [27] suggests a generic way to generate problems via padding arguments[9], these problems are, however, not very intuitive.

[8] Indeed, in the universe of [7] and [10] prior to our work, the smallest class has been $\mathsf{S_2P}$, while the deepest known collapse was to $\mathsf{O_2P}$.

[9] The approach is to pick a language L in $\mathsf{S_2E}$ that is (conjectured) not in BPE. Then the padded version of L will be in $\mathsf{O_2P} \setminus \mathsf{BPP}$.

▶ **Definition 5** (Grid Coloring [3]).
$\mathsf{GC} = \{(1^n01^m01^c) \mid \textit{the } n \times m \textit{ grid can be } c\textit{-colored and not have any monochromatic squares.}\}$

Note that Grid Coloring is an example of one of such problems that are in $\mathsf{NP} \cap \mathsf{SPARSE} \subseteq \mathsf{ONP}$, and hence unlikely to be in BPP. Other problems that come from computational Ramsey theory like the *Gallai-Witt* theorem, and the *Van der Waerden's* theorem have a very similar flavor.

▶ **Observation 6.** $\mathsf{GC} \in \mathsf{ONP} \subseteq \mathsf{O_2P}$.

Below we make a few remarks. For a further discussion see [25].

- $\mathsf{GC} \in \mathsf{NP}$ since the coloring itself is a witness.
- GC is not known to be in P or even BPP.
- $\mathsf{GC} \in \mathsf{SPARSE}$. In fact, GC has a uniformly-sparse extension.
- Therefore, by the results of [22, 27], $\mathsf{GC} \in \mathsf{ONP}$.
- On the other hand, by Mahaney's theorem GC is *unlikely* to be NP-complete.

1.4 Proof Overview

Our work builds on the recent line of work on Range Avoidance. [37] provides a reduction of generating hard truth tables from Avoid, and [13, 41] give a single-valued $\mathsf{S_2P}$ time algorithm for Avoid.

Avoid Framework for Circuit Lower bounds

Let $\mathsf{TT_{n,s}} : \{0,1\}^{s \log s} \to \{0,1\}^{2^n}$ be the truth table generator circuit (see Definition 19), i.e. $\mathsf{TT_{n,s}}$ take as input an encoding of a n-input s-size circuit and outputs the truth table of the circuit. By construction, $\mathsf{TT_{n,s}}$ maps all circuits of size s (encoded using $s \log s$ bits) to their corresponding truth tables. Then, $\mathsf{Avoid}(\mathsf{TT_{n,s}})$ will output a truth-table not in the range of $\mathsf{TT_{n,s}}$ and hence not decided by any s-sized circuit (a circuit lower bound!!). For correctness we only need to ensure that $s \log s < 2^n$, so the $\mathsf{TT_{n,s}}$ is input-expanding, and hence a valid instance of Avoid.

While the above construction gives us a way of getting explicit exponential lower bounds against even $\mathsf{SIZE}[2^n/n]$, the input to Avoid is also exponential in input length n. As a result, the lower bounds we get are for the exponential class $\mathsf{S_2E}$ and not $\mathsf{S_2P}$. Note that one can scale down this lower bound in a black-box manner to get fixed-polynomial lower bounds for $\mathsf{S_2P}$, but will lose explicitness in the process.

To fix this we modify the above reduction to take as input the prefix truth table generator circuit, $\mathsf{PTT_{n,s}} : \{0,1\}^{s \log s} \to \{0,1\}^{s \log s + 1}$, where instead of evaluating the input circuit on the whole truth table, $\mathsf{PTT_{n,s}}$ evaluates on the lexicographically first $(s \log s + 1)$ inputs (see Definition 20). Let $f_{n,s} = \mathsf{Avoid}(\mathsf{PTT_{n,s}})$, and define the truth table of the hard language to be $L := f_{n,s} || 0^{2^n - s \log s - 1}$. By construction, L cannot be decided by any n-input s-size circuit. Moreover, when s is polynomial, the size of $\mathsf{PTT_{n,s}}$ is also polynomial[10] (Lemma 21). Hence the single-valued[11] algorithm computing $f_{n,s}$ is in $\mathsf{S_2P}$ and the explicit fixed-polynomial bounds follow.

[10] In literature the complexity of computing $\mathsf{PTT_{n,s}}$ (Circuit-Eval) is often left as poly, however for our application of getting explicit lower bounds it is crucial to get its fine-grained complexity (see Lemma 16 and Lemma 21).

[11] For the language to be well defined it is essential for the output of our algorithm to be single-valued.

To see that the language $L \in \mathsf{O_2P}$, observe that the $\mathsf{S_2P}$ time algorithm is oblivious to x, since for any x of length n, $f_{n,s}$ is the same. One important observation here is that, for the purpose of obtaining circuit lower bound, it suffices to solve Range Avoidance on *one* specific family of circuits (the truth table generating circuit that maps another circuit to its truth table). Hence, while it is unclear whether Range Avoidance can be solved in $\mathsf{FO_2P}$, we could still obtain circuit lower bound for $\mathsf{O_2P}$.

Hierarchy Theorems for $\mathsf{O_2TIME}$

To get a hierarchy theorem for $\mathsf{O_2TIME}$, we first get an upper bound on $\mathsf{O_2TIME}$ computation via a standard Cook-Levin argument that converts the $\mathsf{O_2TIME}$ verifier into a circuit (SAT-formula) for which we can hard code the "YES" and "NO" irrefutable certificates at every input length (Lemma 31). A lower bound follows via the Avoid framework discussed above (Theorem 29). We can now lift the hierarchy theorem on circuit size (Theorem 14) to get a hierarchy on $\mathsf{O_2TIME}$ (see Theorem 32).

Sparsity and Lower Bounds

We begin by introducing the notion of *uniformly-sparse extensions.* Roughly speaking a sparse language L has a *uniformly-sparse extension* if there is a language $L' \in \mathsf{P}$, such that $L \subseteq L'$ and L' is also sparse (for formal definitions see Section 2.4).

We show that if a language $L \in \mathsf{S_2P}$ has a *uniformly-sparse extension*, then $L \in \mathsf{O_2P}$. Let L' be the *uniformly-sparse extension* of a language $L \in \mathsf{S_2P}$ and let $X = \{x \in L'\}$. Since $L' \in \mathsf{P}$, we first apply the polynomial time algorithm for L' which let us filter out most inputs, i.e. $x \notin L'$, and hence $x \notin L$. We are now left with deciding membership in L over the set X, where $|X| \leq \text{poly}$.

Let V^* be the polynomial time $\mathsf{S_2}$-verifier for L, then for every $x \in X$ there exists either an irrefutable YES certificate (y_x) s.t. $V^*(x, y_x, \cdot) = 1$, or an irrefutable NO certificate (z_x) s.t. $V^*(x, \cdot, z_x) = 0$. Let Y be the set of all such y_x's and Z be the set of all such z_x's. Now for any $x \in X$, it suffices to find the correct y_x from Y (or z_x from Z) and apply $V^*(x, y_x, z_x)$ to decide x.

In Lemma 35 we prove a more efficient parameterized version of this argument. In addition, we are able to apply this in the exponential regime to show the equivalence $\mathsf{O_2E} = \mathsf{S_2E}$ (see Corollary 37).

2 Preliminaries

Let $L \subseteq \{0,1\}^*$ be a language. For $n \geq 1$ we define the *n-th slice of L*, $L|_n := L \cap \{0,1\}^n$ as all the strings in L of length n. The characteristic string of $L|_n$, denoted by $\mathcal{X}_{L|_n}$, is the binary string of length 2^n which represents the truth table defined by $L|_n$.

2.1 Complexity Classes

We assume familiarity with complexity theory and notion of non-uniform circuit families (see for e.g. [4, 26]).

▶ **Definition 7** (Deterministic Time). *Let $t : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{TIME}[t(n)]$, if there exists a deterministic time multi-tape Turing machine that decides L, in at most $O(t(n))$ steps.*

▶ **Definition 8** (Symmetric Time). *Let $t : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{S_2TIME}[t(n)]$, if there exists a $O(t(n))$-time predicate $P(x, y, z)$ that takes $x \in \{0,1\}^n$ and $y, z \in \{0,1\}^{t(n)}$ as input, satisfying that:*

1. *If $x \in L$, then there exists a y such that for all z, $P(x, y, z) = 1$.*
2. *If $x \notin L$, then there exists a z such that for all y, $P(x, y, z) = 0$.*

Moreover, we say $L \in \mathsf{S_2P}$, if $L \in \mathsf{S_2TIME}[p(n)]$ for some polynomial $p(n)$, and $L \in \mathsf{S_2E}$, if $L \in \mathsf{S_2TIME}[t(n)]$ for $t(n) \le 2^{O(n)}$.

▶ **Definition 9** (Single-valued $\mathsf{FS_2P}$ algorithm). *A single-valued $\mathsf{FS_2P}$ algorithm A is specified by a polynomial $\ell(\cdot)$ together with a polynomial-time algorithm $V_A(x, \pi_1, \pi_2)$. On an input $x \in \{0,1\}^*$, we say that A outputs $y_x \in \{0,1\}^*$, if the following hold:*

1. *There exists a $\pi_1 \in \{0,1\}^{\ell(|x|)}$ such that for every $\pi_2 \in \{0,1\}^{\ell(|x|)}$, $V_A(x, \pi_1, \pi_2)$ outputs y_x.*
2. *For every $\pi_1 \in \{0,1\}^{\ell(|x|)}$ there exists a $\pi_2 \in \{0,1\}^{\ell(|x|)}$, such that $V_A(x, \pi_1, \pi_2)$ outputs either y_x or $\perp$.*

And we say that A solves a search problem Π if on any input x it outputs a string y_x and $y_x \in \Pi_x$, where a search problem Π maps every input $x \in \{0,1\}^*$ into a solution set $\Pi_x \subseteq \{0,1\}^*$.

We now formally define $\mathsf{O_2TIME}$ - the oblivious version of the class $\mathsf{S_2TIME}$. The key difference is that unlike $\mathsf{S_2TIME}$, where each irrefutable yes/no certificate can depend on the input x itself, in $\mathsf{O_2TIME}$ the yes/no certificates can **only** depend on $|x|$, the length of x. In other words, for every input length n, there exist a common YES-certificate $\mathbf{y}^*$ and a common NO-certificate $\mathbf{z}^*$ for checking membership of $x \in L|_n$.

▶ **Definition 10** (Oblivious Symmetric Time). *Let $t : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{O_2TIME}[t(n)]$, if there exists a $O(t(n))$-time predicate $P(x, y, z)$ such that for every $n \in \mathbb{N}$ there exist $\mathbf{y}^*$ and $\mathbf{z}^*$ of length $O(t(n))$ satisfying the following for every input $x \in \{0,1\}^n$:*

1. *If $x \in L$, then for all z, $P(x, \mathbf{y}^*, z) = 1$.*
2. *If $x \notin L$, then for all y, $P(x, y, \mathbf{z}^*) = 0$.*

Moreover, we say $L \in \mathsf{O_2P}$, if $L \in \mathsf{O_2TIME}[p(n)]$ for some polynomial $p(n)$, and $L \in \mathsf{O_2E}$, if $L \in \mathsf{O_2TIME}[t(n)]$ for $t(n) \le 2^{O(n)}$.

2.2 Nonuniformity

We recall certain circuit properties:

▶ **Definition 11.** *A boolean circuit C with n inputs and size s, is a Directed Acyclic Graph (DAG) with $(s + n)$ nodes. There are n source nodes corresponding to the inputs labelled $1, \ldots, n$ and one sink node labelled $(n + s)$ corresponding to the output. Each node, labelled $(n + i)$, for $1 \le i \le s$ has an in-degree of 2 and corresponds to a gate computing a binary operation over its two incoming edges.*

▶ **Definition 12.** *Let $s : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in \mathsf{SIZE}[s(n)]$ if L can be computed by circuit families of size $O(s(n))$ for all sufficiently large input size n.*

▶ **Definition 13.** *Let $s : \mathbb{N} \to \mathbb{N}$. We say that a language $L \in$ i.o.-$\mathsf{SIZE}[s(n)]$ if L can be computed by circuit families of size $O(s(n))$ for infinitely many input size n.*

By definition, we have $\mathsf{SIZE}[s(n)] \subseteq i.o.\text{-}\mathsf{SIZE}[s(n)]$. Hence, circuit lower bounds against $i.o.\text{-}\mathsf{SIZE}[s(n)]$ are stronger and sometimes denoted as *almost-everywhere* circuit lower bound in the literature.

We now state the hierarchy theorem for circuit size. The standard proof of this result is existential and goes through a counting argument (see e.g. [4]). However, we comment that using the framework of Avoid, we can now actually get a constructive size hierarchy theorem, albeit with worse parameters.

▶ **Theorem 14** (Circuit Size Hierarchy Theorem[4]). *For all functions $s : \mathbb{N} \to \mathbb{N}$ with $n \le s(n) < o(2^n/n)$:*

$$\mathsf{SIZE}[s(n)] \subsetneq \mathsf{SIZE}[10s(n)] .$$

For our applications, it will be essential to have a tight encoding scheme for circuits. In fact, we will also need the fine-grained complexity of the Turing machine computing Circuit-Eval (i.e. given as input a description of a circuit C and a point x, computes $C(x)$).

▶ **Lemma 15.** *For $n, s \in \mathbb{N}$, and $s \ge n \ge 12$, any n-input, s-size circuit C, there exists an encoding scheme $E_{n,s}$ which encodes C using $5s \log s$ bits.*

Proof. Let C be an n-input, s-size circuit, we now define $E_{n,s}$. Each gate label from $1, \dots, n+s$ can be encoded using $\log(n+s)$ bits. First encode the n inputs using $n\log(n+s)$ bits. Next fix a topological ordering of the remaining gates. For each gate we can encode its two inputs (two previous gates) with $2\log(n+s)$ bits and the binary operation which requires 4 bits (since there 16 possible binary operations). So the length of our encoding is $n\log(n+s) + s(2\log(n+s)+4) \le 3s\log(2s) + 4s \le 5s\log s$ for all $n \ge 12$. ◀

▶ **Lemma 16.** *For $n, s \in \mathbb{N}$, and $s \ge n$, let $E_{n,s}$ be an encoding of an n-input, s-size circuit C using Lemma 15. Then there exists a multi-tape Turing machine M such that $M(E_{n,s}, x) = C(x)$ and it runs in $O(s^2 \log s)$ time.*

Proof. We utilize one tape (memory tape) to store all the intermediate values computed at each gate g_i using $n+s$ cells, and a second tape (evaluation tape) using 6 cells to compute the value at each g_i. We process each gate sequentially as it appears in the encoding scheme, and let g_{i_l} and g_{i_r} be the two gates feeding into g_i. Since Lemma 15 encodes the gates in a topological order, we can assume that when computing g_i, both g_{i_l} and g_{i_r} have already been computed. First copy the value of input bits of x onto the memory tape, and move the head of the input tape to the right by $n\log(n+s)$ steps in $O(s\log s)$ time. Now to compute a gate g_i we write the values of g_{i_l} and g_{i_r} along with the binary operation onto the evaluation tape. We can compute any binary operation with just constant overhead and write its value onto the ith cell of the memory tape. To output the evaluation of the circuit we output the value on the $(n+s)$th cell of the memory tape. The cost of evaluating each gate is dominated by the 2 read and 1 write operations on the memory tape that take $O(s)$ time. Since the size of the input upper bounds the number of gates we have that the simulation takes $O(s|E_{n,s}|) = O(s^2 \log s)$ time. ◀

Finally, we recall the famous Cook-Levin theorem that lets us convert a machine $M \in \mathsf{TIME}[t(n)]$ into a circuit $C \in \mathsf{SIZE}[t(n)\log t(n)]$.

▶ **Theorem 17** (Cook-Levin Theorem [4]). *Let $t : \mathbb{N} \to \mathbb{N}$ be a time constructible function. Then any multi-tape Turing machine running in $\mathsf{TIME}[t(n)]$ time can be simulated by a circuit-family of $\mathsf{SIZE}[t(n)\log t(n)]$.*

2.3 Range Avoidance

▶ **Definition 18.** *The Range Avoidance* (Avoid) *problem is defined as follows: given as input the description of a Boolean circuit* $C : \{0,1\}^n \to \{0,1\}^m$*, for* $m > n$*, find a* $y \in \{0,1\}^m$ *such that* $\forall x \in \{0,1\}^n : C(x) \neq y$*.*

An important object that connects Avoid and circuit lower bound is the truth table generator circuit.

▶ **Definition 19** ([13], Section 2.3)**.** *For* $n, s \in \mathbb{N}$ *where* $n \leq s \leq 2^n$*, the truth table generator circuit* $\mathsf{TT}_{n,s} : \{0,1\}^{L_{n,s}} \to \{0,1\}^{2^n}$ *maps a n-input size s circuit using* $L_{n,s} = (s+1)(7+\log(n+s))$ *bits of description*[12] *into its truth table. Moreover, such circuit can be uniformly constructed in time* $\text{poly}(2^n)$*.*

For the purpose of obtaining fixed polynomial circuit lower bound, we generalise the truth table generator circuit above into one that outputs the prefix of the truth table. We also use a different encoding scheme (with constant factor loss in the parameter) for the convenience of presentation.

▶ **Definition 20.** *For* $n, s \in \mathbb{N}$ *where* $12 \leq n \leq s \leq 2^n$ *and* $|E_{n,s}| = 5s\log s < 2^n$*, the prefix truth table generator circuit* $\mathsf{PTT}_{n,s} : \{0,1\}^{|E_{n,s}|} \to \{0,1\}^{|E_{n,s}|+1}$ *maps a n-input circuit of size s described with* $|E_{n,s}|$ *bits into the lexicographically first* $|E_{n,s}|+1$ *entries of its truth table.*

Since we want to prove lower bounds not just in the exponential regime, but also in the polynomial regime for any fixed polynomial, we need a more fine-grained analysis for the running time of uniformly generating $\mathsf{PTT}_{n,s}$

▶ **Lemma 21.** *The prefix truth table generator circuit* $\mathsf{PTT}_{n,s} : \{0,1\}^{|E_{n,s}|} \to \{0,1\}^{|E_{n,s}|+1}$ *has size* $O(|E_{n,s}|^3)$ *and can be uniformly constructed in time* $O(|E_{n,s}|^3)$*.*

Proof. Let M be the multi-tape Turing machine from Lemma 16 that takes as input an encoding of a circuit and a bitstring, and evaluates the circuit on that bitstring. Let C be the circuit generated from Theorem 17 that simulates M. Then $\mathsf{SIZE}(C) = O(s^2 \log^2 s) = O(|E_{n,s}|^2)$. Making $|E_{n,s}|+1$ copies of C for each output gate gives a circuit of size $O(|E_{n,s}|^3)$. ◀

▶ **Theorem 22** ([41, 13])**.** *There exists a single-valued* $\mathsf{FS_2P}$ *algorithm for* Avoid*. Moreover, on input circuit* $C : \{0,1\}^n \to \{0,1\}^{n+1}$*, the algorithm runs in time* $O(n|C|)$[13]*.*

▶ **Theorem 23** ([41, 13])**.** *There exists an explicit language* $L \in \mathsf{S_2E} \setminus$ *i.o.*-$\mathsf{SIZE}[2^n/n]$*.*

Proof. For any $n \in \mathbb{Z}$, let $\mathsf{TT}_n : \{0,1\}^{2^n-1} \to \{0,1\}^{2^n}$ be the truth table generator circuit. Let $f_n \in \{0,1\}^{2^n}$ be the canonical solution output by the single-valued algorithm from Theorem 22 on input TT_n.

The hard language L is defined as follows: for any $x \in \{0,1\}^*$, $x \in L$ if and only if the $(x+1)$-th bit of $f_{|x|} = 1$, treating x as an integer from 0 to $2^n - 1$. ◀

[12] in fact, it maps a stack program of description size $L_{n,s}$ and it is known that every n-input size s circuit has an equivalent stack program of size $L_{n,s}$ [23].

[13] the running time was implicit in the proof of [41], but easy to verify.

2.4 Sparse Languages

We define some notions of sparsity below, we first introduce natural definitions of sparsity and *sparse extensions* in the polynomial regime, and then give their generalizations in the fine-grained setting.

▶ **Definition 24.** *A language $L \in \mathsf{SPARSE}$ if for all n, $|L \cap \{0,1\}^n| \leq \mathsf{poly}(n)$. Moreover, L is called* uniformly-sparse *if $L \in \mathsf{P} \cap \mathsf{SPARSE}$.*

It is easy to see that $\mathsf{SPARSE} \subseteq \mathsf{P/poly}$. That is, one can identify the yes-instances efficiently, albeit in non-uniform fashion. The purpose of introducing the *uniform-sparsity* is to be able to identify these inputs efficiently in a uniform fashion. Unfortunately, we cannot expect any such language L to lie even in a modestly hard class as, by definition, $L \in \mathsf{P}$. The purpose of the *uniformly-sparse extensions*, on the other hand, is to bridge this gap. One can observe that unlike the *uniformly-sparse* languages, which are contained in P, languages with uniformly-sparse extension can even be undecidable! In particular, any unary language has uniformly-sparse extension in form of 1^*.

▶ **Definition 25.** *A language L has a* uniformly-sparse extension*, if there exists a L' s.t. :*

1. $L \subseteq L'$
2. L' *is* uniformly-sparse

Generalizing the above definitions in the fine-grained setting, we get:

▶ **Definition 26.** *Let $t : \mathbb{N} \to \mathbb{N}$ be a computable function, then a language L is $t(n)$-SPARSE if for all $n, |L \cap \{0,1\}^n| = O(t(n))$. Moreover we say that L is* $t(n)$-uniformly-sparse *if $L \in \mathsf{TIME}[t(n)] \cap t(n)$-$\mathsf{SPARSE}$.*

▶ **Definition 27.** *L has a* $t(n)$-uniformly-sparse extension*, if there exists a L' s.t.:*

1. $L \subseteq L'$
2. L' *is* $t(n)$-uniformly-sparse.

Observe that **every** binary language L is 2^n-SPARSE. Furthermore, every such L has a trivial 2^n-uniformly-sparse extension: $\{0,1\}^*$.

3 Lower Bounds & Hierarchy Theorem

In this section, we first present (Theorem 28) a fine-grained, parameterised version of Theorem 23. This allows us to use the Avoid framework and get circuit lower bounds in $\mathsf{S_2TIME}[t(n)]$ instead of $\mathsf{S_2E}$. We then observe that our $\mathsf{S_2TIME}[t(n)]$ witness is oblivious of the input, and hence the lower bounds we get are actually in $\mathsf{O_2TIME}[t(n)]$ as highlighted in Theorem 29.

In Theorem 32 we present the first time hierarchy theorem for $\mathsf{O_2P}$. In fact, we note to the best of our knowledge that this is the first known time hierarchy theorem for a semantic class.

▶ **Theorem 28.** *For $n \in \mathbb{N}$, let $t : \mathbb{N} \to \mathbb{N}$ be a time-constructible function, s.t. $t(n) > n \geq 12$ then*

$$\mathsf{S_2TIME}[t(n)] \not\subseteq i.o.\text{-}\mathsf{SIZE}\left[\frac{t(n)^{1/4}}{\log(t(n))}\right] .$$

Proof. We construct a language $L_t \in \mathsf{S_2TIME}[t(n)]$ and $L_t \not\subseteq i.o.\text{-}\mathsf{SIZE}\left[\frac{t(n)^{1/4}}{\log(t(n))}\right]$.

For any $n \in \mathbb{N}$, let $s = \lfloor \frac{t(n)^{1/4}}{\log(t(n))} \rfloor$ and $|E_{n,s}| = \lceil 5s \log s \rceil$. Let $\mathsf{PTT}_{n,s} : \{0,1\}^{|E_{n,s}|} \to \{0,1\}^{|E_{n,s}|+1}$ be the prefix truth table generator circuit as in Definition 20. Let $f_n \in \{0,1\}^{|E_{n,s}|+1}$ be the canonical solution to $\mathsf{Avoid}(\mathsf{PTT_{n,s}})$ as outputted by the single-valued algorithm from Theorem 22.

The hard language L_t is defined as follows: for any $n \in \mathbb{Z}$, the characteristic string of $L_t|_n$ is set to be $\mathcal{X}_{L_t|_n} := f_n || 0^{2^n - |E_{n,s}| - 1}$.

By definition of $\mathsf{PTT}_{n,s}$ and the fact that $f_n \notin \mathsf{Image}(\mathsf{PTT}_{n,s})$, we have that $L_t \notin i.o.\text{-}\mathsf{SIZE}[s]$. On the other hand, the single-valued algorithm for finding f_n runs in time $O(|E_{n,s}| \cdot |\mathsf{PTT}_{n,s}|) = O(t(n))$. Hence, $L_t \in \mathsf{S_2TIME}[t(n)]$. ◀

We make the observation that the witness in the $\mathsf{S_2TIME}$ machine above is oblivious to the actual input x.

▶ **Theorem 29.** *For $n \in \mathbb{N}$, let $t : \mathbb{N} \to \mathbb{N}$ be a time-constructible function, s.t. $t(n) > n \geq 12$ then*

$$\mathsf{O_2TIME}[t(n)] \not\subseteq i.o.\text{-}\mathsf{SIZE}\left[\frac{t(n)^{1/4}}{\log(t(n))}\right].$$

Proof. Consider the same language L_t in the proof of Theorem 28. Notice that for any input x of the same length n, the $\mathsf{FS_2P}$ algorithm is run on the same circuit $\mathsf{PTT}_{n,s}$ and hence the witness is the same for inputs of the same length. Thus, it follows that $L_t \in \mathsf{O_2TIME}[t(n)]$. ◀

We now get as a corollary a proof of Theorem 1.

▶ **Corollary 30.** *For all $k \in \mathbb{N}$, there exists an explicit language $L_k \in \mathsf{O_2P}$ s.t. $L_k \not\subseteq \mathsf{SIZE}[n^k]$.*

Proof. Fix $t(n) = n^{5k}$. Then there is an explicit hard language L_t as defined in the proof of Theorem 28, such that $L_t \not\subseteq \mathsf{SIZE}[n^k]$. Moreover, by Theorem 29 we have that $L_t \in \mathsf{O_2P}$. ◀

Before proving our hierarchy theorem for $\mathsf{O_2TIME}$, we prove a simple lemma that bounds from above the size of a circuit family computing languages in $\mathsf{O_2TIME}$.

▶ **Lemma 31.** $\mathsf{O_2TIME}[t(n)] \subseteq \mathsf{SIZE}[t(n)\log(t(n))]$.

Proof. Consider any language $L \in \mathsf{O_2TIME}[t(n)]$, and let $V(\cdot,\cdot,\cdot)$ be its $t(n)$-time verifier. For any integer $n \in \mathbb{N}$, let $y_n, z_n \in \{0,1\}^{t(n)}$ be the irrefutable proofs for input size n. By Theorem 17 we can convert $V(\cdot,\cdot,\cdot)$ into a circuit family $\{C_n\} \subseteq \mathsf{SIZE}[t(n)\log(t(n)]$. The values y_n and z_n can be hard-coded into C_n, and hence this circuit will decide L on all inputs of size n. ◀

Having both an upper bound on the size of circuits simulating an $\mathsf{O_2TIME}$ computation, and also a lower bound for $\mathsf{O_2TIME}$ against circuits, we can use the circuit size hierarchy (Theorem 14) to define a time hierarchy on $\mathsf{O_2TIME}$.

▶ **Theorem 32.** *For $n \in \mathbb{N}$, let $t : \mathbb{N} \to \mathbb{N}$ be a time constructible function, s.t. $t(n) > n \geq 12$ then:* $\mathsf{O_2TIME}[t(n)] \subsetneq \mathsf{O_2TIME}[t(n)^4 \log^9(t(n))]$.

Proof. Combining Theorem 29, Lemma 31, and Circuit Size Hierarchy (Theorem 14) we have:

$$\mathsf{O_2TIME}[t(n)] \subseteq \mathsf{SIZE}[t(n)\log t(n)] \subsetneq \mathsf{SIZE}[t(n)\log^{\frac{5}{4}} t(n)] ,$$

and

$$\mathsf{O_2TIME}[t(n)^4 \log^9(t(n))] \not\subseteq \mathsf{SIZE}[t(n)\log^{\frac{5}{4}} t(n)] .$$ ◀

▶ **Theorem 33.** *For $n \in \mathbb{N}$, let $t : \mathbb{N} \to \mathbb{N}$ be a time constructible function, s.t. $t(n) \geq n$ then: for all $\varepsilon > 0, \mathsf{O_2TIME}[t(n)] \subsetneq \mathsf{O_2TIME}[t(n)^{1+\varepsilon}]$.*

Proof. Let us assume that $\mathsf{O_2TIME}[t(n)^{1+\varepsilon}] \subseteq \mathsf{O_2TIME}[t(n)]$, then by translation we have that:

$$\mathsf{O_2TIME}[t(n)^{(1+\varepsilon)^2}] \subseteq \mathsf{O_2TIME}[t(n)^{1+\varepsilon}] \subseteq \mathsf{O_2TIME}[t(n)]$$

Inducting on any $k > 2$, we get that $\mathsf{O_2TIME}[t(n)^{(1+\varepsilon)^k}] \subseteq \mathsf{O_2TIME}[t(n)]$. Now setting $k = \lceil 3.1/\varepsilon \rceil$, by Bernoulli's inequality we have that $(1+\varepsilon)^k \geq (1+\varepsilon)^{3.1/\varepsilon} \geq 4.1$. Therefore, $\mathsf{O_2TIME}[t(n)^{4.1}] \subseteq \mathsf{O_2TIME}[t(n)]$, which contradicts Theorem 32. ◀

▶ Remark 34. We note here that while we are able to achieve a better gap in our hierarchy theorem in Theorem 33 over Theorem 32, there is a trade off. The hierarchy theorem defined in Theorem 32 is explicit, that is we have an explicit language not known to be contained in the smaller class. However, when we apply our translation argument to get better parameters in Theorem 33 we lose this explicitness.

4 Sparsity

In this section, we use *sparse extensions* to get various structural complexity results. We prove a more fine-grained statement of Theorem 3 which states that any language in $\mathsf{S_2TIME}[t(n)]$ with a *uniformly-sparse extension* is actually in $\mathsf{O_2TIME}[t(n)^2]$. This lets us extract as a corollary another proof of $\mathsf{S_2E} = \mathsf{O_2E}$. As another application of *sparse extensions*, we are able to recover the fixed polynomial lowerbounds for $\mathsf{O_2P}$ from the previous section as stated in Theorem 1. Finally we show connections between *sparse extensions* and open problems posed by [27].

▶ **Lemma 35.** *Let $L \in \mathsf{S_2TIME}[t(n)]$. If L has a $t(n)$-uniformly-sparse extension then $L \in \mathsf{O_2TIME}[t(n)^2]$.*

Proof. For any n, let L' be the $t(n)$-*uniformly-sparse extension* of L, and let $\mathcal{F}$ be the $\mathsf{TIME}[t(n)]$ predicate that decides membership in L'. We will now design an $\mathsf{O_2TIME}[t(n)^2]$ verifier V for $L|_n$. Since both L and L' are $t(n)$-SPARSE, we have that for most $x \in \{0,1\}^n$: $L|_n(x) = L'|_n(x) = 0$. V will first use $\mathcal{F}$ to efficiently filter out most non-membership in $L'|_n$, and hence $L|_n$ in $\mathsf{TIME}[t(n)]$. Now V only has to decide membership in L over $t(n)$ many inputs $X = \{x \in \{0,1\}^n : \mathcal{F}(x) = 1\}$. We will use the fact that since $L \in \mathsf{S_2TIME}[t(n)]$, for all $x \in X$, if $x \in L$ there is an irrefutable YES certificate y_x and if $x \notin L$ there is an irrefutable NO certificate z_x and a verifier V^*, running in $\mathsf{TIME}[t(n)]$ s.t.

- if $x \in L$, $\exists y_x$, $\forall z$ s.t. $V^*(x, y_x, z) = 1$
- if $x \notin L$, $\exists z_x$, $\forall y$ s.t. $V^*(x, y, z_x) = 0$

Consider the string Y^* which encodes a table of YES witnesses y_x^* for every input $x \in X$. When $x \in L$ we set $y_x^* = y_x$, and when $x \notin L$ we will set $y_x^* = 0^{t(n)}$. The size of Y^* is $O(t(n)^2)$, since there are at most $t(n)$ entries in the table each of length $t(n) + n$. For every $x \in X \cap \overline{L}$, let z_x be the irrefutable NO-certificate corresponding to x for V^*. We set Z^* to be the concatenation of all such z_x. The size of Z^* is also at most $t(n)^2$.

We now show that Y^* and Z^* will serve as oblivious irrefutable "YES" and "NO" certificates respectively for V. On input (x, Y^*, Z^*), V first parses Y^* to find the corresponding y_x^* in time $\mathsf{TIME}[t(n)^2]$. Then for each $z_i \in Z^*$ we run $V^*(x, y_x^*, z_i)$. If for all z_i, $V^*(x, y_x^*, z_i) = 1$ then V outputs 1, otherwise V will output 0. Since we are making at most $t(n)$ calls that each cost $\mathsf{TIME}[t(n)]$, V runs in $\mathsf{TIME}[t(n)^2]$.

```
$V(x, Y^*, Z^*)$ :
(1) Set output $= 1$.
(2) If $\mathcal{F}(x) = 0$, return 0.
(3) Parse $Y^*$ to get $y_x^*$.
(4) For $z_i \in Z^*$, do:
    (a) output $=$ output $\wedge V^*(x, y_x^*, z_i)$.
(5) Return output.
```

Figure 1 $\mathsf{O_2TIME}[t(n)^2]$ Verifier for Language in $\mathsf{S_2TIME}[t(n)]$ with $t(n)$-*uniformly-sparse extension.*

To see correctness, we first analyze the case when $x \in L$, then by construction Y^* includes $y_x^* = y_x$ and V will output 1. On the other hand if $x \notin L$ then there is an irrefutable no-certificate z_x in Z^* so there is no y_i such that $V(x, y_i, z_x) = 1$. Hence V outputs 0. ◀

By taking $t(n)$ to be a polynomial in Lemma 35 we directly get Corollary 36 (also Theorem 3) relating $\mathsf{O_2P}$ and $\mathsf{S_2P}$.

▶ **Corollary 36.** *If $L \in \mathsf{S_2P}$ and L has an uniformly-sparse extension, then $L \in \mathsf{O_2P}$*

Similarly, one can prove Theorem 4 by showing the same consequence for coNP vs coONP, thus making a partial progress towards the open questions posed by Goldreich and Meir in [27]. In the exponential regime, since all languages have the trivial 2^n-*uniformly-sparse extension* we get the equivalence between $\mathsf{O_2E}$ and $\mathsf{S_2E}$ as seen in Corollary 37.

▶ **Corollary 37.** $\mathsf{S_2E} = \mathsf{O_2E}$

Proof. As noted in Section 2.4, every language is 2^n-SPARSE, and has the trivial 2^n-*uniformly-sparse extension*: $\{0,1\}^*$. When $t(n) = 2^n$, by Lemma 35 we get that $\mathsf{S_2TIME}[2^n] \subseteq \mathsf{O_2TIME}[2^{2n}]$. ◀

In particular, the following lemma shows that the hard language in $\mathsf{S_2TIME}[t(n)]$ defined in Theorem 28 admits a $t(n)$-*uniformly-sparse extension*, giving another proof of Corollary 30.

▶ **Lemma 38.** *For $n \in \mathbb{N}$, let $t : \mathbb{N} \to \mathbb{N}$ be a time-constructible function, s.t. $t(n) > n \geq 12$ then, there is an explicit language $L_t \in \mathsf{S_2TIME}[t(n)]$ s.t. $L_t \notin \mathsf{SIZE}\left[\frac{t(n)^{1/4}}{\log(t(n))}\right]$. Moreover, L_t has a $t(n)$-uniformly-sparse extension L'_t.*

Proof. Let L_t be the $\mathsf{S_2TIME}[t(n)]$ language defined in the proof Theorem 28 with the characteristic string $\mathcal{X}_{L_t|_n} := f_n||0^{2^n-|E_{n,s}|-1}$. We now define the language L'_t whose characteristic string $\mathcal{X}_{L'_t|_n} := 1^{|E_{n,s}|+1}||0^{2^n-|E_{n,s}|-1}$. To see that this L'_t is a $t(n)$-*uniformly-sparse extension* of L_t, clearly $L_t \subseteq L'_t$. Moreover membership of $x \in L'_t$ can be decided by checking if the binary value of x is less than or equal to $|E_{n,s}| + 1$ which can be done in $\mathsf{TIME}[n] \subseteq \mathsf{TIME}[t(n)]$. ◀

Equipped with this lemma we have an alternative proof of fixed polynomial lower bounds for $\mathsf{O_2P}$ as stated in Theorem 1.

▶ **Corollary 39** (Theorem 1)**.** *For every $k \in \mathbb{N}$, $\mathsf{O_2P} \not\subseteq \mathsf{SIZE}[n^k]$. Moreover, for every k there is an explicit language L_k in $\mathsf{O_2P}$ s.t. $L_k \notin \mathsf{SIZE}[n^k]$.*

Proof. Fix $t(n) = n^{5k}$. Then by Lemma 38 there is an explicit language L_k such that $L_k \not\subseteq \mathsf{SIZE}[n^k]$, and L_k has an *uniformly-sparse extension.* Applying Lemma 35 we have that $L_k \in \mathsf{O_2TIME}[n^{10k}] \subseteq \mathsf{O_2P}$. ◀

5 Open Problems

We conclude with a few interesting open problems:

- Can we show that every sparse $\mathsf{S_2P}$ language is also in $\mathsf{O_2P}$?
- Can we show a non-trivial upper bound for $\mathsf{O_2P}$, for example $\mathsf{P^{NP}}, \mathsf{MA}, \mathsf{PP}$? This would imply explicit fixed-polynomial lower bounds for such classes. On the other hand, we do note that under reasonable derandomization assumptions, $\mathsf{O_2P} \subseteq \mathsf{S_2P} = \mathsf{P^{NP}}$.
- Can we arrive at something interesting about time hierarchy theorem for semantic classes where fixed-polynomial lower bounds are known e.g. $\mathsf{S_2P}$, $\mathsf{ZPP^{NP}}$, assuming $\mathsf{NP} \not\subseteq \mathsf{P/poly}$? For instance, if $\mathsf{NP} \subseteq \mathsf{P/poly}$, then it follows that $\mathsf{S_2P} \subseteq \mathsf{P/poly}$. One could then invoke the circuit size hierarchy theorem (Theorem 14) to establish a hierarchy theorem for $\mathsf{S_2TIME}$, similar to how we obtain the hierarchy theorem for $\mathsf{O_2TIME}$.

References

1 S. Aaronson. Oracles are subtle but not malicious. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 340–354. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.32. 5

2 S. Aaronson. The learnability of quantum states. In *Proceedings of the Royal Society A*, volume 463, pages 3089–3114, 2007. doi:10.1098/rspa.2007.0113. 3

3 D. Apon, W. Gasarch, and K. Lawler. The complexity of grid coloring. *Theory Comput. Syst.*, 67(3):521–547, 2023. doi:10.1007/S00224-022-10098-5. 3, 7

4 S. Arora and B. Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009. 8, 10

5 B. Barak. A probabilistic-time hierarchy theorem for “slightly non-uniform" algorithms. In *RANDOM*, pages 194–208, 2002. 4

6 N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996. doi:10.1006/JCSS.1996.0032. 5

7 J.-Y. Cai. $S_2P \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences*, 73(1):25–35, 2007. 2, 5, 6

8 J.-Y. Cai and O. Watanabe. On proving circuit lower bounds against the polynomial-time hierarchy. *SIAM J. Comput.*, 33(4):984–1009, 2004. doi:10.1137/S0097539703422716. 5

9 R. Canetti. More on BPP and the polynomial-time hierarchy. *Inf. Process. Lett.*, 57(5):237–241, 1996. doi:10.1016/0020-0190(96)00016-6. 2

10 V. T. Chakaravarthy and S. Roy. Oblivious symmetric alternation. In *STACS*, pages 230–241, 2006. 3, 5, 6

11 V. T. Chakaravarthy and S. Roy. Arthur and merlin as oracles. *Comput. Complex.*, 20(3):505–558, 2011. doi:10.1007/S00037-011-0015-3. 3, 5

12 E. Chattopadhyay and D. Zuckerman. Explicit two-source extractors and resilient functions. *Annals of Mathematics*, 189(3):653–705, 2019. doi:10.4007/annals.2019.189.3.1. 4

13 L. Chen, S. Hirahara, and H. Ren. Symmetric exponential time requires near-maximum circuit size. In *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2024, to appear. Association for Computing Machinery, 2024. 2, 4, 5, 7, 11

14 L. Chen and R. 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. 4

15 Y. Chen, Y. Huang, J. Li, and H. Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1058–1066, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3564246.3585147. 4, 5

16 Y. Chen and J. Li. Hardness of range avoidance and remote point for restricted circuits via cryptography. In *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2024, to appear. Association for Computing Machinery, 2024. 4

17 E. Chung, A. Golovnev, Z. Li, M. Obremski, S. Saraogi, and N. Stephens-Davidowitz. On the randomized complexity of range avoidance, with applications to cryptography and metacomplexity. *ECCC preprint*, 2023. https://eccc.weizmann.ac.il/report/2023/193/. 4

18 S. A. Cook. A hierarchy for nondeterministic time complexity. In *Proceedings of the fourth annual ACM symposium on Theory of computing - STOC '72*, STOC '72. ACM Press, 1972. doi:10.1145/800152.804913. 4

19 P. Dixon, A. Pavan, J. Vander Woude, and N. V. Vinodchandran. Pseudodeterminism: promises and lowerbounds. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1552–1565, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520043. 4

20 L. Fortnow and R. Santhanam. Hierarchy theorems for probabilistic polynomial time. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 316–324, 2004. 4

21 L. Fortnow, R. Santhanam, and L. Trevisan. Hierarchies for semantic classes. In *Proceedings of the 37th Annual ACM SIGACT Symposium on Theory of Computing*, pages 348–355. ACM, New York, 2005. doi:10.1145/1060590.1060642. 4

22 L. Fortnow, R. Santhanam, and R. Williams. Fixed-polynomial size circuit bounds. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 19–26. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.21. 3, 5, 7

23 G. S. Frandsen and P. B. Miltersen. Reviewing bounds on the circuit size of the hardest functions. *Information processing letters*, 95(2):354–357, 2005. doi:10.1016/J.IPL.2005.03.009. 11

24 K. Gajulapalli, A. Golovnev, S. Nagargoje, and S. Saraogi. Range avoidance for constant depth circuits: Hardness and algorithms. In Nicole Megow and Adam D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 65:1–65:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.APPROX/RANDOM.2023.65. 4

25 W. Gasarch. https://blog.computationalcomplexity.org/2010/07/spares-problems-in-np-thought-to-not-be.html, 2010. URL: https://blog.computationalcomplexity.org/2010/07/spares-problems-in-np-thought-to-not-be.html. 7

26 O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. 8

27 O. Goldreich and O. Meir. Input-oblivious proof systems and a uniform complexity perspective on p/poly. *ACM Transactions on Computation Theory (TOCT)*, 7(4):1–13, 2015. doi:10.1145/2799645. 3, 6, 7, 14, 15

28 V. Guruswami, X. Lyu, and X. Wang. Range avoidance for low-depth circuits and connections to pseudorandomness. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 20:1–20:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.20. 4

29 J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117(0):285–306, 1965. doi:10.1090/s0002-9947-1965-0170805-7. 4

30 F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *J. ACM*, 13(4):533–546, October 1966. `doi:10.1145/321356.321362`. 4

31 R. Ilango, J. Li, and R. Williams. Indistinguishability obfuscation, range avoidance, and bounded arithmetic. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1076–1089, New York, NY, USA, 2023. Association for Computing Machinery. `doi:10.1145/3564246.3585187`. 4

32 R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: derandomizing the xor lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997. 2

33 R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982. `doi:10.1016/S0019-9958(82)90382-5`. 2, 5

34 R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 302–309, 1980. `doi:10.1145/800141.804678`. 5

35 R. Kleinberg, O. Korten, D. Mitropolsky, and C. Papadimitriou. Total Functions in the Polynomial Hierarchy. In James R. Lee, editor, *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ITCS.2021.44`. 4

36 J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998. `doi:10.1137/S0097539795296206`. 5

37 O. Korten. The hardest explicit construction. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 433–444. IEEE, 2022. 4, 7

38 O. Korten and T. Pitassi. Strong vs. weak range avoidance and the linear ordering principle. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2024. URL: `https://eccc.weizmann.ac.il/report/2024/076/`. 4

39 J. Li and T. Yang. 3.1n- o (n) circuit lower bounds for explicit functions. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1180–1193, 2022. 2

40 X. Li. Two source extractors for asymptotically optimal entropy, and (many) more. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1271–1281, Los Alamitos, CA, USA, November 2023. IEEE Computer Society. `doi:10.1109/FOCS57990.2023.00075`. 4

41 Z. Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *Proceedings of the 56th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2024. Association for Computing Machinery, 2024. 2, 4, 5, 7, 11

42 Z. Lu, I. C. Oliveira, and R. Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, pages 303–316, New York, NY, USA, 2021. Association for Computing Machinery. `doi:10.1145/3406325.3451085`. 4

43 S. R. Mahaney. Sparse complete sets of NP: solution of a conjecture of berman and hartmanis. *J. Comput. Syst. Sci.*, 25(2):130–143, 1982. `doi:10.1016/0022-0000(82)90002-2`. 3

44 D. van Melkebeek and K. Pervyshev. A generic time hierarchy with one bit of advice. *Computational Complexity*, 16(2):139–179, 2007. `doi:10.1007/S00037-007-0227-8`. 4

45 P. B. Miltersen, N. V. Vinodchandran, and O. Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In *COCOON*, pages 210–220, 1999. 2

46 N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994. `doi:10.1016/S0022-0000(05)80043-1`. 2

47 S. P. Radziszowski. Small ramsey numbers. *The Electronic Journal of Combinatorics [electronic only]*, DS01, 2021. URL: `https://www.combinatorics.org/ojs/index.php/eljc/article/view/DS1`. 4

48 H. Ren, R. Santhanam, and Z. Wang. On the range avoidance problem for circuits. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 640–650, Los Alamitos, CA, USA, November 2022. IEEE Computer Society. doi:10.1109/FOCS54457.2022.00067. 4

49 A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998. doi:10.1007/S000370050007. 2

50 R. Santhanam. Circuit lower bounds for merlin–arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009. doi:10.1137/070702680. 5

51 J. I. Seiferas, M. J. Fischer, and A. R. Meyer. Separating nondeterministic time complexity classes. *Journal of the ACM*, 25(1):146–167, January 1978. doi:10.1145/322047.322061. 4

52 C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell Syst. Tech. J.*, 28(1):59–98, 1949. doi:10.1002/J.1538-7305.1949.TB03624.X. 2

53 N. V. Vinodchandran. A note on the circuit complexity of PP. *Theor. Comput. Sci.*, 347(1-2):415–418, 2005. doi:10.1016/J.TCS.2005.07.032. 5

54 R. Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014. doi:10.1145/2559903. 2

55 S. Žák. A turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, October 1983. doi:10.1016/0304-3975(83)90015-4. 4, 6

When Far Is Better: The Chamberlin-Courant Approach to Obnoxious Committee Selection

Sushmita Gupta
The Institute of Mathematical Sciences, HBNI, Chennai, India

Tanmay Inamdar
Indian Institute of Technology Jodhpur, Jodhpur, India

Pallavi Jain
Indian Institute of Technology Jodhpur, Jodhpur, India

Daniel Lokshtanov
University of California Santa Barbara, CA, USA

Fahad Panolan
School of Computer Science, University of Leeds, UK

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

Abstract

Classical work on metric space based committee selection problem interprets distance as “near is better”. In this work, motivated by real-life situations, we interpret distance as “far is better”. Formally stated, we initiate the study of “obnoxious” committee scoring rules when the voters’ preferences are expressed via a metric space. To accomplish this, we propose a model where *large distances imply high satisfaction* (in contrast to the classical setting where shorter distances imply high satisfaction) and study the egalitarian avatar of the well-known Chamberlin-Courant voting rule and some of its generalizations. For a given integer value λ between 1 and k, the committee size, a voter derives satisfaction from only the λth favorite committee member; the goal is to maximize the satisfaction of the least satisfied voter. For the special case of $\lambda = 1$, this yields the egalitarian Chamberlin-Courant rule. In this paper, we consider general metric space and the special case of a d-dimensional Euclidean space.

We show that when λ is 1 and k, the problem is polynomial-time solvable in $\mathbb{R}^2$ and general metric space, respectively. However, for $\lambda = k - 1$, it is NP-hard even in $\mathbb{R}^2$. Thus, we have “double-dichotomy” in $\mathbb{R}^2$ with respect to the value of λ, where the extreme cases are solvable in polynomial time but an intermediate case is NP-hard. Furthermore, this phenomenon appears to be “tight” for $\mathbb{R}^2$ because the problem is NP-hard for general metric space, even for $\lambda = 1$. Consequently, we are motivated to explore the problem in the realm of (parameterized) approximation algorithms and obtain positive results. Interestingly, we note that this generalization of Chamberlin-Courant rules encodes practical constraints that are relevant to solutions for certain facility locations.

2012 ACM Subject Classification Theory of computation → Facility location and clustering; Theory of computation → Fixed parameter tractability

Keywords and phrases Metric Space, Parameterized Complexity, Approximation, Obnoxious Facility Location

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.24

Related Version *Full Version*: https://arxiv.org/abs/2405.15372 [36]

Funding Sushmita Gupta and Pallavi Jain acknowledge support from SERB-SUPRA (grant number SPR/2021/000860).

Sushmita Gupta: The author acknowledges the support from SERB-MATRICS (grant number MTR/2021/000869).

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 24; pp. 24:1–24:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tanmay Inamdar: The author acknowledges support from IITJ Research Initiation Grant (grant number I/RIG/TNI/20240072).
Pallavi Jain: The author acknowledges support from IITJ Seed Grant (grant number I/SEED/PJ/20210119).
Daniel Lokshtanov: The author acknowledges support from NSF award CCF-2008838.
Saket Saurabh: The author acknowledges support from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416); and he also acknowledges the support of Swarnajayanti Fellowship (grant number DST/SJF/MSA-01/2017-18).

1 Introduction

Initiated in the 18th century, the multiwinner election problem, also known as the committee selection problem, has been central to social choice theory for over a century [16, 68, 53] and in the last decade and a half it has been among the most well-studied problems in computational social choice [3, 29, 9, 8]. In this problem, given a set of candidates $\mathcal{C}$, a profile $\mathcal{P}$ of voters' preferences, and an integer k; the goal is to find a k-sized subset of candidates (called a *committee*) using a multiwinner voting rule. The committee selection problem has many applications beyond parliamentary elections, such as selecting movies to be shown on a plane, making various business decisions, choosing PC members for a conference, choosing locations for fire stations in a city, and so on. For more details on the committee selection problem, we refer the reader to [29, 46].

The Chamberlin-Courant (CC) committee is a central solution concept in the world of committee selection. Named after Chamberlin and Courant [16], it is derived from a multiwinner voting rule where the voter's preference for a given k-sized committee is evaluated by adding the preference of each voter for its *representative*, the most preferred candidate in the committee.The CC committee is one with the highest value. There has been a significant amount of work in computational social choice centered around this concept and has to date engendered several *CC-type rules* that can be viewed as a generalization of the above. Specifically, *ordered weighted average*(OWA) operator-based rules such as the *median scoring rule*, defined formally later in [62, 4] can be seen as a direct generalization of CC. Moreover, there are other notions of generalization based on the preference aggregation principle: the original CC rule is *utilitarian*, that is, it takes the summation of each voter's preference value toward its representative, [62, 10, 37]. The *egalitarian* variant studied by Aziz et al. [4] and Gupta et al. [37] is one where only the least satisfied voter's preference value towards its representative is taken. Clearly, there could be many other variants where some other aggregation principle is considered. We refer to all these variants collectively as the CC-*type* rules and the egalitarian variants as the *egalitarian* CC-*type* rules. The egalitarian rules, also known as Rawlsian rules, are based on the highly influential Rawls's theory of "justice as fairness" [58, 59] that favors equality in some sense by maximising the minimum satisfaction. Egalitarian rules are very well-studied in voting theory [44, 4, 37, 24, 66].

In this paper, we consider the situation where the voters' preferences are expressed via a metric space, a natural setting in the facility location problem. Facility location can be viewed as an application of the committee selection (also known multi-winner voting) problem [17] and spatial voting [48]. Furthermore, we consider egalitarian scoring rules, which aim to maximize the "satisfaction" of the least satisfied voter. Moreover, Gupta et al. [37] study a wide range of egalitarian rules, called the *egalitarian median rule*, which is a generalization of the egalitarian CC rule and is defined as follows: for a voter v and a committee S, let pos_v^S (called a *position vector*) be the vector of positions of candidates in S in the ranking of v in increasing order. For example, for the voter $v\colon a \succ b \succ d \succ c$ and set $S = \{c, d\}$, the vector $pos_v^S = [3, 4]$. In the egalitarian median rule, given a value $1 \le \lambda \le k$, the satisfaction of a voter v for a committee S is given by $m - pos_v^S[\lambda]$, where m is the total number of candidates and $pos_v^S[\lambda]$ is the position value of the λ^{th} candidate in S according to v's preference list. Note that when $\lambda = 1$, we get the egalitarian CC scoring rule. Gupta et al. [37] proved that the egalitarian median rule is NP-hard for every $\lambda < k$.

In contrast to the aforementioned intractibility results for egalitarian CC-type rules, Betzler et al.[7] show that for $\lambda = 1$, the egalitarian median rule is polynomial-time solvable for 1-dimensional Euclidean preferences because such preferences are single-peaked. This motivates us to study egalitarian CC-type rules in the metric space setting that goes beyond dimension 1.

Preferences via a metric space. For a given metric space, preferences are encoded in the following manner: each candidate and each voter is represented by a point in the metric space. In earlier works, distance is viewed as being inversely proportional to preference, that is, a voter is said to have a higher preference for candidates who are closer to her than those that are farther in the metric space, [65, 13, 41, 34, 29, 64, 20]. In this paper, we consider the opposite scenario where *distance is directly proportional to preference*, that is, a candidate who is *farther away is more preferred* than the one who is closer. Inspired by obnoxious facility location [47, 67, 21], we call our problem *obnoxious committee selection.* Before we delve into the formal definition of our problem, we discuss the use of metric space to encode preference in earlier works.

- The facility location problem is actually equivalent to the committee selection problem, where we assume that the closer facility is more preferred. The well-known k-CENTER problem (also known as the MINIMAX FACILITY LOCATION problem in metric space) is equivalent to the egalitarian CC committee selection problem when the voters' preferences are encoded in a metric space, where higher preference is given to the candidate that is closer. For some applications, it is natural to demand more than one facility in the vicinity, e.g., convenience stores, pharmacies, healthcare facilities, playgrounds, etc. This is known as the *fault tolerant* k-CENTER problem and is captured by egalitarian median rules in a metric space [19]. Facility location is among the most widely studied topics in algorithms, and we point the reader to some recent surveys [1, 22, 35] on the topic and to [17] for a survey on facility location in mechanism design.
- In the spatial theory of voting, voters and candidates are embedded in the d-dimensional Euclidean space, and each voter ranks the candidates according to their distance from them [38, 48].

In the last few years, a fair amount of research centered on the theme of voting, committee selection, especially the CC rule, in metric spaces has appeared in theory and economics and computation venues [64, 55, 18, 71, 38, 54, 51, 2]. Motivated by the applications stated above, we consider the general metric spaces as well as the Euclidean space for our study. Next, we discuss our motivation for studying *obnoxious committee* selection before presenting the formal definition.

Why Obnoxious Committee? The committee selection problem has been studied for the metric space in literature [27, 34, 43, 63, 64]. All these papers use the "closer is better" perspective and thus candidates that are closer are preferred over those that are farther. Motivated by real-life scenarios where *every kind of facility is not desirable in the vicinity* such as is the case with factories, garbage dumps and so on, we want to study a problem which allows us to *restrict the number of facilities in the vicinity.* This is particularly relevant for facilities that bring some utility but too many lead to loss in value or even to negative utility. In order to design an appropriate solution concept for scenarios such as these we associate higher preference to facilities (i.e candidates) that are far and set the value of λ in a situation-specific way. For example, consider a situation where a local government is searching locations to build k factories, with the constraint that each of the k factories is located far from every neighborhood. This can be modeled by our problem by setting each of the neighborhoods as voters, each potential factory location as a candidate, and $\lambda = k$. Moreover, for facilities such as garbage recycling, we can set $\lambda = k - \mathcal{O}(1)$ so that all but few facilities are located far from any neighborhood. Since the value of λ can depend on k (which is part of the input), we take λ to be part of the input. Overall, we observe that as far as satisfaction is concerned, different facilities bring different levels of satisfaction depending on how many of them are in the vicinity. Consequently, it is desirable to have a model which is robust enough to capture this nuance. This translates to λ being user defined, and is thus specified as part of the input to the problem.

Formal definition. We introduce some notation before giving a formal definition of the problem studied in this paper. For a given metric space $\mathscr{M} = (X, d)$, a point $x \in X$, a subset $S \subseteq X$, and $\lambda \in [k]$, we define $d^\lambda(x, S)$ to be the distance of x to the λ^{th} farthest point in S. To define this notion formally, we may sort the distances of a point x to each $s \in S$ in non-increasing order (breaking ties arbitrarily, if needed), and let these distances be $d(x, s_1) \geq d(x, s_2) \geq \ldots \geq d(x, s_k)$. Then, $s_\lambda \in S$ is said to be the λ^{th} farthest point from x in S, and $d^\lambda(x, S) = d(x, s_\lambda)$. Note that $d^1(x, S)$ is the distance of x from a farthest point in S. For a point $p \in X$ and a non-negative real r, $B(p, r) := \{q \in X : d(p, q) \leq r\}$ denotes the ball of radius r centered at p.

Obnoxious Egalitarian Median Committee Selection
(Obnox-Egal-Median-CS, in short)
Input: A metric space $\mathscr{M}$ consisting of a set of voters, $\mathscr{V}$, a set of candidates, $\mathcal{C}$; positive integers k and $\lambda \in [k]$; and a positive real t.
Question: Does there exist a subset $S \subseteq \mathcal{C}$ such that $|S| = k$ and for each $v \in \mathscr{V}$, $d^\lambda(v, S) \geq t$?

When $\lambda = 1$, we give the problem a special name, Obnoxious-Egal-CC, due to its similarity with the egalitarian CC rule. Note that the egalitarian (resp. utilitarian) CC rule itself is the special case of the egalitarian (resp. utilitarian) median rule when $\lambda = 1$.

Our Contributions. In the following, we discuss the highlights of our work in this paper and the underlying ideas used to obtain the result.

- We begin with studying Obnoxious-Egal-CC, that is, Obnox-Egal-Median-CS with $\lambda = 1$, and show that it is polynomial-time solvable when voters and candidates are embedded in $\mathbb{R}^2$ with Euclidean distances, Theorem 1. To design this algorithm, we first observe that the above setting can be equivalently reformulated as the following geometric problem. Given $\mathscr{V}$, and a set of equal-sized disks $\mathcal{D}$, find a k-size subset $\mathcal{D}' \subseteq \mathcal{D}$ such that no point of $\mathscr{V}$ belongs to the common intersection region of $\mathcal{D}'$. Following that we use geometric properties of equal-sized disks to design an algorithm that uses dynamic programming to inductively build such a region. This algorithmic result contrasts with the intractability of the non-obnoxious version (the k-Center problem) which is known to be NP-hard in $\mathbb{R}^2$.

- In Theorem 7, we consider Obnoxious-Egal-CC in general metric spaces. We show that it is NP-hard, and in fact, the optimization variant is also W[2]-hard to approximate beyond a factor of 1/3, parameterized by k, the committee size. Informally speaking, this implies that no algorithm with running time $f(k)n^{\mathcal{O}(1)}$ is likely to exist, assuming widely believed complexity-theoretic assumptions. For more background on parameterized complexity, the reader may refer to the full version [36], or more generally, a textbook on the topic [23].
- Notwithstanding these negative results, we show that Obnoxious-Egal-CC admits a factor 1/4 approximation algorithm that runs in polynomial time, Theorem 9. In this algorithm, we first compute a "$t/2$-net" $S \subseteq \mathcal{C}$, i.e., S satisfies the following two properties: (1) $d(c, c') > t/2$ for any distinct $c, c' \in S$, and (2) for any $c \notin S$, there exists some $c' \in \mathscr{M}$ such that $d(c, c') \leq t/2$. Now, consider a point $p \in \mathscr{V}$ and a $c^* \in \mathcal{C}$, such that $d(p, c^*) \geq t$. Then, by using the two properties of S, we argue that there exists a point in $c' \in S$ that is "near" c, and hence, "far from" p. More specifically, we can show that $d(p, c') \geq t/4$, leading to a 1/4-approximate solution.
- Our work on Obnox-Egal-Median-CS for $\lambda > 1$ reveals that for $\lambda = k$, the problem can be solved in polynomial time due to the fact that every committee member needs to be at least t distance away from every voter. So, if possible, we can choose any k candidates that are t-distance away from every voter; otherwise, a solution does not exist. The algorithm is same as the one in Proposition 3 in [4], but here we can have ties. We show that for $\lambda = k - 1$, Obnox-Egal-Median-CS is NP-hard (Theorem 11) even when the voters and candidates are points in $\mathbb{R}^2$. Furthermore, we show that the intractability results we have for Obnoxious-Egal-CC in Theorem 7 carry forward to $\lambda > 1$, as shown in Theorem 13.
- For an arbitrary value of λ in $\mathbb{R}^d$ space, we exhibit a *fixed-parameter tractable approximation scheme*, that is, an algorithm that returns a solution of size k, in time FPT in (ϵ, λ, d), such that for every point $v \in \mathscr{V}$ there are at least λ points in the solution that are at distance at least $(1-\epsilon)t$ from v, Theorem 22. Note that $\lambda \leq k$, thus, this algorithm is also FPT in (ϵ, k, d). To obtain this result, we first observe that it is possible to further refine the idea of $t/2$-net, and define a set of "representatives", if the points belong to a Euclidean space. In this setting, for any $0 < \epsilon < 1$, we can compute a candidate set $\mathcal{R}$ of representatives, such that for every relevant $c \in \mathcal{C}$, there exists a $c' \in \mathcal{R}$ such that $d(c, c') \geq \epsilon/2$. Moreover, $\mathcal{R}$ is bounded by a function of λ, d, and ϵ. Thus, we can find an $(1 - \epsilon)$-approximation by enumerating all size-k subsets of $\mathcal{R}$.

Related works. Much of the research on multiwinner voting is concentrated on the computational complexity of computing winners under various rules, because for many applications it is crucial to be able to efficiently compute exact winners. As might be expected, computing winners under some committee scoring rules can be done in polynomial time (e.g., k-Borda [29]), while for many of the others the decision problem is NP-hard.

Effort towards applying the framework of parameterized complexity to these problems has primarily focused on parameters such as the committee size k and the number of voters, n. Indeed, this line of research has proven to be rather successful (see, e.g., [11, 10, 28, 31, 30, 4, 7, 6, 32, 70, 72, 49, 5, 52, 37, 69]). The problem has also been studied through the perspectives of approximation algorithms [55, 12] and parameterized approximation algorithms [60, 61, 10].

It is worth noting the similarities between our model and that of the *fault tolerant* versions of clustering problems, such as k-Center or k-Median [45, 39, 14], also [15]. In the latter setting, the clustering objective incorporates the distance of a point to its λ^{th} closest chosen center. Here, $\lambda \geq 1$ is typically assumed to be a small constant. Thus, even if $\lambda - 1$ centers chosen in the solution undergo failure, and if they all happen to be nearby a certain point p, we still have some (upper) bound on the distance of p to its now-closest center. Note that this motivation of fault tolerance translates naturally into our setting, where we want some (lower) bound on the distance of a voter to its λ^{th} *farthest* candidate, which may be useful of the $\lambda - 1$ farthest *candidates* are unable to perform their duties.

Preliminaries. In the optimization variant of Obnox-Egal-Median-CS, the input consists of $(\mathscr{M}, \mathscr{V}, \mathcal{C}, k, \lambda)$ as defined above, and the goal is to find the largest t^* for which the resulting instance is a yes-instance of Obnox-Egal-Median-CS, and we call such a t^* the optimal value of the instance. We say that an algorithm has an approximation guarantee of $\alpha \leq 1$, if for any input $(\mathscr{M}, \mathscr{V}, \mathcal{C}, k, \lambda)$, the algorithm finds a subset $S \subseteq \mathcal{C}$ of size k such that for each $v \in \mathscr{V}$, $d^\lambda(v, S) \geq \alpha \cdot t^*$.

For more details on parameterized complexity, we refer the reader to the textbooks [23, 33, 25, 56].

2 Obnoxious Egalitarian Chamberlin-Courant (CC)

We begin our study with Obnoxious-Egal-CC. Recall that Obnox-Egal-Median-CS with $\lambda = 1$ is Obnoxious-Egal-CC. We begin with the Euclidean space, followed by the general metric space.

2.1 Polynomial Time algorithm in $\mathbb{R}^2$

In this section, we design a polynomial time algorithm when the voters and candidates are embedded in $\mathbb{R}^2$. In particular, we prove the following result.

▶ **Theorem 1.** *There exists a polynomial-time algorithm to solve an instance of* Obnoxious-Egal-CC *when* $\mathscr{V} \cup \mathcal{C} \subset \mathbb{R}^2$*, and the distances are given by Euclidean distances.*

Overview. Before delving into a formal description of the polynomial-time algorithm, we start with a high-level overview of the result. For simplicity of the exposition, we assume that $t = 1$ (this can be easily achieved by scaling $\mathbb{R}^2$, and thus all points in the input, by a factor of t). For each $c \in \mathcal{C}$, let $D(c)$ denote a *unit disk* (i.e., an open disk of *diameter* 1) with c as its center. In the new formulation, we want to find a subset $S \subseteq \mathcal{C}$ of size k, such that for each $v \in \mathscr{V}$, the solution S contains at least one candidate c, such that v is outside $D(c)$ (which is equivalent to saying that the euclidean distance between v and c is larger than 1, which was exactly the original goal). This is an equivalent reformulation with a more geometric flavor, thus enabling us to use techniques from computational geometry.

First, we perform some basic preprocessing steps, that will be help us in the main algorithm. First, if there is a disk $D(c)$ that does not contain a voter, then any set containing c is a solution. Similarly, if we have two disjoint unit disks $D(c)$ and $D(c')$ centered at distinct $c, c' \in \mathcal{C}$, then any superset of $\{c, c'\}$ of size k is a valid solution, which can be found and returned easily. We check this condition for all subsets of size 2. In the final step of preprocessing, we iterate over all subsets of candidates of size 3, and check whether the common intersection of the corresponding three disks is empty – if we find such a set, then it is easy to see that any of its k-sized superset forms a solution. Now, assuming that the preprocessing step does not already give the solution, we know that each subset of unit disks of size at most 3 have a common intersection. By a classical result in discrete geometry called Helly's theorem [50], this also implies that each non-empty subset must have a common intersection. Our goal is to find a smallest such subset S, for which, the common intersection region is devoid of all voters $v \in \mathscr{V}$. We design a dynamic programming algorithm to find such a subset. Note that each subset is in one-to-one correspondence with a convex region defining the boundary of the common intersection, and the boundary of the common intersection consists of portions of boundaries of the corresponding unit disks (also known as "arcs"). The dynamic programming algorithm considers partial solutions defined by a consecutive sequence of arcs that can be attached end-to-end, while at the same time, ensuring that the common intersection does not contain any voter $v \in \mathscr{V}$. When we are trying to add another arc to the boundary, we have to make sure that (i) one of the endpoints of the arc is the same as one of the endpoints of the last arc defining the partial boundary, and (ii) the new area added to the "partial common intersection" does not contain a voter. We need to introduce several defintions and handle several special cases in order to formally prove the correctness of this strategy, which we do next.

Formal description. We work with the rescaled and reformulated version of the problem, as described above. Further, we assume, by infinitesimally perturbing the points if required (see, e.g., [26]), that the points $\mathcal{C} \cup \mathscr{V}$ satisfy the following general position assumption: no three unit disks centered at distinct candidates intersect at a common point. Note that this assumption is only required in order to simplify the algorithmic description.

For each candidate $c \in \mathcal{C}$, let $D(c)$ denote the unit disk (i.e., an open disk of radius 1) with c as center. In the following, we will often omit the qualifier *unit*, since all disks are assumed to be open unit disks unless explicitly mentioned otherwise. Note that our original problem is equivalent to determining whether there exists a subset $S \subseteq \mathcal{C}$ of size k such that for every $v \in \mathscr{V}$, there exists a candidate $c \in S$ such that $v \notin D(c)$. Equivalently, we want to find a set $S \subseteq \mathcal{C}$ such that $\left(\bigcap_{c \in S} D(c)\right) \cap \mathscr{V} = \emptyset$. For a subset $S' \subseteq \mathcal{C}$, we let $I(S') := \bigcap_{c \in S'} D(c)$, and let $D(S') = \{D(c) : c \in S'\}$. We design a polynomial-time algorithm to find a smallest-sized subset $S' \subseteq \mathcal{C}$ such that $I(S') \cap \mathscr{V} = \emptyset$. For any two points $x, y \in \mathbb{R}^2$, let $\overline{xy}$ be the straight-line segment joining x and y.

We first perform the following preprocessing steps to handle easy cases. For $k = 1$, we try each $c \in \mathcal{C}$ and check whether $d(v, c) \geq 1$ for all voters $v \in \mathscr{V}$. Now suppose $k \geq 2$. First, we check whether there exists a pair of disks centered at distinct $c_1, c_2 \in \mathcal{C}$ such that the distance between c_1, c_2 is at least 2. Then, for any voter $v \in \mathscr{V}$, if $d(v, c_1) < 1$, then $d(v, c_2) > 1$ by triangle inequality. Therefore, $\{c_1, c_2\}$ can be augmented by adding arbitrary set of $k - 2$ candidates in $\mathcal{C} \setminus \{c_1, c_2\}$ to obtain a solution. Now suppose that neither of the previous two steps succeeds. Then, we try all possible subsets $S' \subseteq \mathcal{C}$ of size at most 3, and check whether $I(S') = \emptyset$, that is, no point in $\mathbb{R}^2$ belongs to $I(S')$ (note that this specifically implies that $I(S') \cap \mathscr{V} = \emptyset$). If we find such a set S', then we can add an arbitrary subset of $\mathcal{C} \setminus S'$ of size $k - |S'|$ to obtain a set S of size k. Thus, we can make the following assumptions, given that the preprocessing step does not solve the problem.

1. $k \geq 4$,
2. For every $c, c' \in \mathcal{C}$, $D(c) \cap D(c') \neq \emptyset$, and the two disks intersect at two distinct points (this is handled in the second step of preprocessing), and
3. For any subset $\emptyset \neq S \subseteq \mathcal{C}$, $I(S) \neq \emptyset$. In particular, this also holds for sets S with $|S| > 3$ – otherwise by Helly's theorem [50], there would exist a subset $S' \subseteq S$ of size 3 such that $I(S') = \emptyset$, a case handled in the preprocessing step.

Let $\mathcal{P}$ be a set of intersection points of the boundaries of the disks $\{D(c) : c \in \mathcal{C}\}$. Note that since the boundaries of every pair of disks intersect exactly twice (this follows from the item (2) above), $|\mathcal{P}| = 2\binom{|\mathcal{C}|}{2}$. Furthermore, for $c \in \mathcal{C}$, let $\mathcal{P}(c) \subset \mathcal{P}$ be the set of intersection points that lie on the boundary of $D(c)$. For $c \in \mathcal{C}$ and distinct $p, q \in \mathcal{P}(c)$, we define $\mathsf{arc}(p, q, c)$ as the *minor arc* (i.e., the portion of the boundary of $D(c)$ that is smaller than a semicircle) of disk $D(c)$ with p and q as its endpoints. Note that p and q are interchangeable in the definition, and $\mathsf{arc}(p, q, c) = \mathsf{arc}(q, p, c)$. For a subset $S' \subseteq \mathcal{C}$, let $\mathcal{A}(S')$ be the set of arcs defining the boundary of the region $I(S')$ – note that since $I(S') \neq \emptyset$ for any $S' \neq \emptyset$, $\mathcal{A}(S')$ is well-defined and is a non-empty set of arcs. We first have the following proposition, the proof of which follows from arguments in planar geometry.

▶ **Proposition 2.** *Fix a set $S \subseteq \mathcal{C}$ with $|S| \geq 2$. Furthermore, assume that S is a* minimal *set with intersection equal to $I(S)$, i.e., there is no subset $S' \subset S$ such that $I(S') = I(S)$. Then, for every $c \in S$, $\mathcal{A}(S)$ contains exactly one arc of the form $\mathsf{arc}(p, q, c)$ for some $p, q \in \mathcal{P}(c)$.*

Proof. First we prove that every arc in $\mathcal{A}(S)$ is a minor arc. Suppose for contradiction that $\mathcal{A}(S)$ contains a non-minor arc A on the boundary of some $D(c)$, $c \in S$. Consider any $c' \in S$ with $c' \neq c$, and let $S' = \{c, c'\}$. Note that $I(S) \subseteq I(S')$ as $S' \subseteq S$ and intersection of disk can only decrease by adding more points to the set. Thus, $\mathcal{A}(S')$ contains an arc A' that is a superset of A. Let p and q denote the endpoints of A', and note that A' is also a non-minor arc. Note that $p \in I(S') = D(c) \cap D(c')$. Let p' denote the point on $D(c)$ that is diametrically opposite to p, and since A' is a major arc, it follows that $p' \in A' \subseteq I(S') = D(c) \cap D(c')$. To summarize, both p and p' belong to both $D(c)$ and $D(c')$. However, since both $D(c)$ and $D(c')$ are unit disks, $\overline{pp'}$ is a common diameter of $D(c)$ and $D(c')$, which contradicts that c and c' are distinct.

Now we prove the second part of the claim, that is, for each $c \in S$, $\mathcal{A}(S)$ contains exactly one minor arc of the form $\mathsf{arc}(\cdot, \cdot, c)$. Suppose there exists some c such that there exist two arcs $A_1 = \mathsf{arc}(p_1, q_1, c)$ and $A_2 = \mathsf{arc}(p_2, q_2, c)$ in $\mathcal{A}(S)$. Note that A_1 and A_2 must be disjoint, otherwise we can concatenate them to obtain a single arc. Suppose, without loss of generality, traversing clockwise along the boundary of $D(c)$, the ordering of the points is p_1, q_1, q_2, p_2. Let $c_1 \in S$ (resp. $c_2 \in S$) be the candidate such that q_1 (resp. q_2) belongs on the boundaries of $D(c)$ and $D(c_1)$ (resp. $D(c)$ and $D(c_2)$). It is clear that $c \neq c_1$ and $c \neq c_2$. We further claim that $c_1 \neq c_2$ – suppose this is not the case. Then, q_1 and q_2 belong to the boundaries of $D(c)$ and $D(c_1)$. In this case, p_1 (or p_2) cannot belong to $D(c) \cap D(c_1) \subseteq I(S)$, which contradicts the assumption that p_1 (or p_2) lie on the boundary of $I(S)$. Thus, we have that c, c_1, c_2 are all distinct. However, again we reach a contradiction since p_1 is outside $D(c) \cap D(c_2) \subseteq I(S)$. It follows that each arc appears at most once in $\mathcal{A}(S)$.

Finally, we consider the case when there is some $c \in S$ such that no arc of the form $\mathsf{arc}(\cdot, \cdot, c)$ belongs to $\mathcal{A}(S)$. In this case, the region bounded by $\mathcal{A}(S)$, i.e., $I(S)$, is completely contained inside $D(c)$. However, this implies that $I(S) = I(S \setminus \{c\})$, which contradicts the minimality of S. ◀

Next, we proceed towards designing our dynamic programming algorithm.

Algorithm. For any $x, y, p \in \mathcal{P}$, $c_1, c \in \mathcal{C}$, and an integer $i \geq 2$, we define a table entry $A[x, y, p, c_1, c, i]$ that denotes whether there exists a region $R(x, y, p, c_1, c) \subset \mathbb{R}^2$ with the following properties:

- $R = R(x, y, p, c_1, c)$ is a convex region bounded by a set $\mathcal{A}(R)$ of $i - 1$ circular arcs, and straight-line segment $\overline{xy}$, such that $\mathcal{A}(R)$ contains:
  - At most one arc of the form $\mathsf{arc}(\cdot, \cdot, c')$ for every $c' \in \mathcal{C}$.
  - Exactly one arc of the form $\mathsf{arc}(x, \cdot, c_1)$, which is the *first* arc traversed along the boundary of R in clockwise direction, starting from x. Note that c_1 is the center of this arc.
  - $\mathsf{arc}(y, p, c)$
- $R \cap \mathscr{V} = \emptyset$.

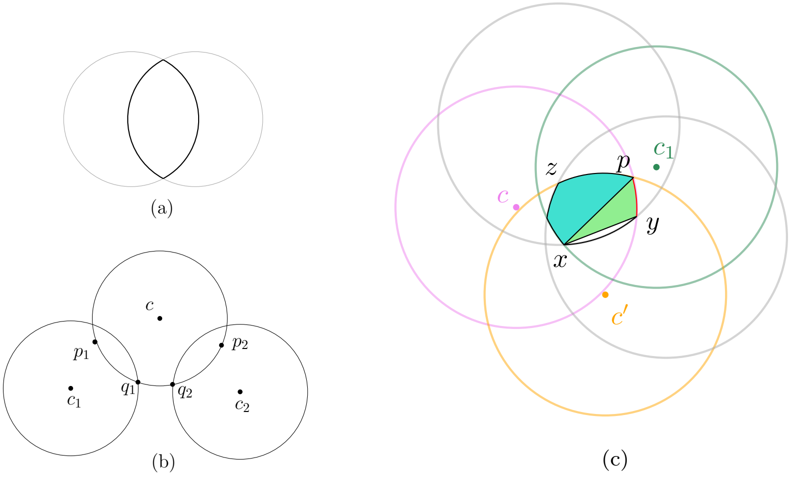


Figure 1 Illustration for the proof of Proposition 2 and the algorithm. Fig (a): intersection of boundaries of two unit disks is defined by two minor arcs. Fig (b): Two disjoint arcs $\mathsf{arc}(p_1, q_1, c)$ and $\mathsf{arc}(p_2, q_2, c)$ cannot appear on the boundary of a common intersection, since they correspond to disjoint regions. Fig (c): Illustration for the dynamic program. A region formed by intersection of 5 disks is shown. $\mathsf{arc}(p, y, c)$ is shown in red. Blue region corresponds to the entry $A[x, p, z, c_1, c', 3]$, and green region corresponds to the newly added region to the blue region, corresponding to the entry $A[x, y, p, c_1, c, 4]$.

Note that if $\mathsf{arc}(y, p, c)$ is not defined, or any of the other conditions do not hold, then a region with the required properties does not exist.

First, we compute all entries $A[x, y, p, c, c_1, i]$ with $i \leq 3$. Note that the number of arcs of the form $\mathsf{arc}(\cdot, \cdot, \cdot)$ is bounded by $O(|\mathcal{P}|^2 \cdot |\mathcal{C}|) = O(|\mathcal{C}|^3)$, and since $i \leq 3$, we can explicitly construct all such candidate regions in polynomial time. Thus, we can correctly populate all such table entries with `true` or `false`.

Now, we discuss how to fill a table entry $A[x, y, p, c_1, c, i]$ with $i \geq 4$. We fix one such entry and its arguments. If the region $R(x, p, y, c_1, c)$ bounded by $\overline{xp}, \overline{xy}$ and $\mathsf{arc}(y, p, c)$ contains a point from $\mathscr{V}$, then the entry $A[x, p, y, c_1, c, i]$ is defined to be `false`. Note that this can easily be checked in polynomial time. Otherwise, suppose that $R(x, p, y, c_1, c) \cap \mathscr{V} = \emptyset$. In this case, let $\mathcal{T}$ be a set of tuples of the form $(x, p, z, c_1, c', i-1)$, where $z \in \mathcal{P}$, and $c' \in \mathcal{C}$ such that the following conditions are satisfied. (1) $c' \notin \{c, c_1\}$, (2) The minor arc $\mathsf{arc}(p, z, c')$ exists, and (3) When traversing along this arc from z to p, the arc $\mathsf{arc}(p, y, c)$ is a "right turn". Formally, consider the tangents $\ell_c, \ell_{c'}$ to the disks $D(c)$ and $D(c')$ at point p respectively. Let H_c (resp. $H_{c'}$) be the closed halfplane defined by the line ℓ_c ($\ell_{c'}$) that contains $D(c)$ ($D(c')$). Then, arcs $\mathsf{arc}(p, z, c')$ and $\mathsf{arc}(y, p, c)$ must belong to $H_c \cap H_{c'}$. See Figure 1(c). Then,

$$A[x, y, p, c_1, c, i] = \bigvee_{(x,p,z,c_1,c',i-1) \in \mathcal{T}} A[x, p, z, c_1, c', i-1].$$

Since we take an or over at most $|\mathcal{C}| \times |\mathscr{V}|$ many entries, each such entry can be computed in polynomial time. Furthermore, since the number of entries is polynomial in $|\mathcal{C}|$ and $|\mathscr{V}|$, the entire table can be populated in polynomial time.

Now, we iterate over all entries $A[x, y, p, c_1, c, i]$ such that the following conditions hold.

- $A[x, y, p, c_1, c, i] =$ `true`,
- There exists some $c'' \in \mathcal{C} \setminus \{c, c_1\}$ such that $\mathsf{arc}(x, y, c'')$ exists, and
- The region bounded by $\mathsf{arc}(x, y, c'')$ and segment $\overline{xy}$ does not contain any point from $\mathcal{C}$.

The meaning here is that $\mathsf{arc}(x, y, c'')$ is the last arc bounding the required region.

If such an entry exists with $i \leq k-1$, then we conclude that the given instance of Obnox-Egal-Median-CS is a yes-instance. Otherwise, it is a no-instance. Finally, using standard backtracking strategy in dynamic programming, it actually computes a set $S \subseteq \mathcal{C}$ such that $I(S) \cap \mathscr{V} = \emptyset$. Next, we establish the proof of correctness of this dynamic program.

A Proof of Correctness.

▶ **Lemma 3 (♣).**[1] *Consider an entry $A[x,y,p,c_1,c,i]$, corresponding to some $x,y,p \in \mathcal{P}, c_1, c \in \mathcal{C}$. Then, $A[x,y,p,c_1,c,i] = \texttt{true}$ if and only if the corresponding region R, as in the definition of the table entry, contains no point of $\mathscr{V}$.*

▶ **Lemma 4.** *This algorithm correctly decides whether the given instance of* Obnox-Egal-Median-CS *is a* yes-*instance.*

Proof. First, it is easy to see that the preprocessing step correctly finds a minimum-size subset of at most 3 whose intersection contains no point of $\mathscr{V}$, if such a subset exists. Thus, we now assume that the preprocessing step does not find a solution, and the algorithm proceeds to the dynamic programming part.

Recall that due to Proposition 1, if S is a minimal subset of centers, such that $I(S) \cap \mathscr{V} = \emptyset$ (if any), then $\mathcal{A}(S)$ contains exactly one minor arc that is part of the circle centered at each $c \in S$. In particular, this holds for the optimal set S^* of centers (if any), and let $i = |S^*|$. Pick an arbitrary arc in $\mathcal{A}(S^*)$, and let c_1 be the center of this arc, and x be one of the endpoints of this arc. By traversing the arcs in $\mathcal{A}(S^*)$ in clockwise manner, let the last two arcs be $\mathsf{arc}(y,p,c)$, and $\mathsf{arc}(y,x,c')$. Then, by Lemma 3, it follows that $A[x,y,p,c_1,c',i-1] = \texttt{true}$, and the region bounded by $\overline{xy}$ and $\mathsf{arc}(y,x,c')$ does not contain a point from $\mathcal{C}$. Thus, the algorithm outputs the correct solution corresponding to the entry $A[x,y,p,c_1,c',i-1]$.

In the other direction, if the algorithm finds an entry $A[x,y,p,c_1,c',i-1] = \texttt{true}$, such that (1) $\mathsf{arc}(x,y,c)$ exists, (2) $c \notin \{c', c_1\}$, and (3) the region bounded by $\mathsf{arc}(x,y,c)$ and $\overline{xy}$ does not contain a point from $\mathcal{C}$, then using Lemma 3, we can find a set of i disks whose intersection does not contain a point from $\mathcal{C}$. Therefore, if for all entries it holds that at least one of the conditions does not hold, then the algorithm correctly concludes that the given instance is a no-instance. ◀

The algorithm as it is does not work for $\lambda > 1$. We do not know whether the problem is polynomial time solvable for $\lambda > 1$.

2.2 Hardness in Graph Metric

In this section, we show the intractability of the problem when the voters and candidates are embedded in the graph metric space, which implies the intractability in the general metric space. The metric space defined by the vertex set of a graph as points and distance between two points as the shortest distance between the corresponding vertices in the graph is called the *graph metric space.*

We present a reduction from the Hitting Set problem, defined below, which is known to be NP-hard [42] and W[2]-parameterized by k [23].

[1] The proofs of the statements marked with ♣ can be found in the full version [36].

Hitting Set
Input: Set system $(\mathcal{U}, \mathcal{F})$, where $\mathcal{U}$ is the ground set of n elements, $\mathcal{F}$ is a family of subsets of $\mathcal{U}$, and a positive integer k
Question: Does there exist $H \subseteq \mathcal{U}$ of size k such that for any $S \in \mathcal{F}$, $H \cap S \neq \emptyset$?

Reduction. Define a graph G with vertex set $\mathscr{V} \cup \mathcal{C}$ as follows. For every element $e \in \mathcal{U}$, we add a candidate c_e to $\mathcal{C}$, and for every set $S \in \mathcal{F}$, we add a voter v_S to $\mathscr{V}$. We add an edge (c_e, v_S) in G if and only if $e \notin S$. The weight of all such edges is equal to 1. Also, for any $c_e, c'_e \in \mathcal{C}$, we add an edge of weight 2. The distance function $d : (\mathscr{V} \cup \mathcal{C}) \to \mathbb{R}^+$ is given by the shortest path distances in G.

▶ **Observation 5.** *For any $e \in \mathcal{U}$, and $S \in \mathcal{F}$, $d(c_e, v_S) = 1$ if and only if $e \notin S$. Otherwise $d(c_e, v_S) = 3$ if and only if $e \in S$.*

▶ **Lemma 6 (♣).** *$(\mathcal{U}, \mathcal{F})$ admits a hitting set of size k if and only if there exists a set $H \subseteq \mathcal{C}$ of size k such that for any $v_S \in \mathscr{V}$, $\max_{c_e \in H} d(c_e, v_S) = 3$.*

In fact, this construction shows that it is NP-hard to approximate the problem within a factor of $1/3 + \epsilon$ for any $\epsilon > 0$. Indeed, suppose there existed such a $\beta = (1/3 + \epsilon)$-approximation for some $\epsilon > 0$. Then, if $(\mathcal{U}, \mathcal{F})$ is a yes-instance of Hitting Set, then Lemma 6 implies that $\mathsf{OPT} = 3$ – here OPT denotes the largest value of t for which we have a yes-instance for the decision version. In this case, the β-approximation returns a solution S of size k and of cost at least $\beta \cdot 3 = 1 + 3\epsilon > 1$. This implies that for each $v_S \in \mathscr{V}$, there exists some $c_e \in S$ with $d(v_S, c_e) > 1$. However, such a c_e must correspond to an element $e \in S$ – otherwise $d(v_S, c_e) = 1$ by construction. Therefore, the solution S corresponds to a hitting set of size k. Alternatively, if $(\mathcal{U}, \mathcal{F})$ is a no-instance of Hitting Set, then Lemma 6 implies that the there is no solution of size k with cost 3. Thus, a β-approximation can be used to distinguish between yes- and no-instances of Hitting Set. Hence, we have the following result.

▶ **Theorem 7.** Obnoxious-Egal-CC *is* NP*-hard. Furthermore, for any $\alpha > 1/3$,* Obnoxious-Egal-CC *does not admit a polynomial time α-approximation algorithm, unless* P = NP*. Furthermore,* Obnoxious-Egal-CC *does not admit an* FPT*-approximation algorithm parameterized by k with an approximation guarantee of $\alpha > 1/3$, unless* FPT $= W[2]$.

2.3 Approximation Algorithm in General Metric Space

In this section, we design a polynomial time 1/4-approximation algorithm when voters and candidates are embedded in a general metric space. Since the problem is trivial for $k = 1$ (we can simply iterate over all solutions of size 1, i.e., all $c \in \mathcal{C}$, and check whether it forms a solution), we assume in the rest of the section that $k > 1$.

We first guess a voter $p' \in \mathscr{V}$ and a candidate $c' \in \mathcal{C}$ such that c' is the farthest candidate from p' in an optimal solution S^*. Let $t = d(p', c')$. We know that all candidates in S^* are within a distance t from p', i.e., $S^* \subseteq B$, where $B = B(p', t) \cap \mathcal{C}$. Let M be a $(t/2)$-net of B, i.e., M is a *maximal* set of candidates with the following properties: (1) $d(c_i, c_j) > t/2$ for any distinct $c_i, c_j \in M$, and (2) for any $c \notin M$, there exists some $c' \in M$ such that $d(c, c') \leq t/2$. Note that such a $(t/2)$-net can be found in polynomial-time using a simple greedy algorithm.

Now there are two cases. (1) If $|M| \geq k$, let M' be an arbitrary subset of M of size exactly k. (2) If $|M| < k$, then let $M' = M \cup Q$ where Q is an arbitrary subset of candidates from $B \setminus M$ such that $|M'| = k$.

▶ **Lemma 8.** *Fix some $v \in \mathscr{V}$, and let $c \in \mathcal{C}$ be the farthest center from v in M'. Then, $d(v, c) \geq t/4$.*

Proof. We consider two cases based on the size of M, and the way we obtain M' from M.

Case 1: $|M| \geq k$, and M' is an arbitrary subset of M. Suppose for contradiction $d(v, c) < t/4$. Since c is the farthest candidate from v, the same is true for any candidate $c' \in M'$. Then, $d(p, c) < t/4$ and $d(v, c') < t/4$, which implies that $d(c, c') \leq d(v, c) + d(v, c') < t/4 + t/4 = t/2$, which contradicts property 1 of M.

Case 2: $|M| < k$ and M' is obtained by adding arbitrary candidates to M. If $d(v, c) \geq t/2 \geq t/4$, we are done. So assume that $d(v, c) < t/2$. Let c^* be the farthest center from v in an optimal solution. Then, $d(v, c^*) \geq t$. Also, $c^* \notin M \subseteq M'$, otherwise $d(v, c) \geq d(v, c^*) \geq t$, since c is the farthest candidate from v. Therefore, by property 2, there exists some $c' \in M$ such that $d(c', c^*) \leq t/2$. Again, $d(v, c') \leq d(v, c) < t/2$. Then, $d(v, c^*) \leq d(v, c') + d(c', c^*) < t/2 + t/2 = t$. This contradicts $d(v, c^*) \geq t$. ◀

Thus, we conclude with the following theorem.

▶ **Theorem 9.** *There is a polynomial-time $1/4$-approximation algorithm for the optimization variant of* Obnoxious-Egal-CC *when the voters and candidates belong to an arbitrary metric space $\mathscr{M}$.*

3 Obnoxious Egalitarian Median Committee Selection for $\lambda > 1$

In this section, we move our study to the case when $\lambda > 1$. We first show that $\lambda = k - 1$ is NP-hard in $\mathbb{R}^2$. But the extreme cases of $\lambda = 1$ or $\lambda = k$ are tractable: In fact, the $\lambda = 1$ case is polynomial-time solvable for $\mathbb{R}^2$ but the $\lambda = k$ is polynomial-time solvable even in a general metric space. Furthermore, we show that similar to $\lambda = 1$, the problem is hard to approximate for any value of λ in graph metric. Finally, contrary to Theorem 13 we give an FPT-approximation scheme for arbitrary value of λ in $\mathbb{R}^d$.

3.1 Hardness

In this section, we present results pertaining to NP-hardness and approximation hardness.

NP-hardness for $\lambda = k - 1$ in $\mathbb{R}^2$. To exhibit this we give a reduction from the 2-Independent Set problem in unit disk graphs (UDGs). We give a formal definition of UDGs below, followed by the definition of the aforementioned problem.

▶ **Definition 10.** *Given a set $\mathscr{P} = \{p_1, p_2, \ldots, p_n\}$ of points in the plane, a unit disk graph (UDG, in short) corresponding to the set $\mathscr{P}$ is a graph $G = (\mathscr{P}, E)$ satisfying $E = \{(p_i, p_j) | d(p_i, p_j) \leq 1\}$, where $d(p_i, p_j)$ denotes the Euclidean distance between p_i and p_j.*

2-Independent Set In Unit Disk Graph
Input: Given a set $V \subset \mathbb{R}^2$ of n points, and a positive integer k.
Question: Let $G = (V, E)$ be a unit disk graph defined on V. Does there exist a subset $S \subseteq V$ such that $|S| = k$, and for any distinct $u, w \in S$, $d_G(u, w) > 2$?

This problem is shown to be NP-hard in [40].

▶ **Theorem 11.** Obnox-Egal-Median-CS *is* NP*-hard when $\lambda = k - 1$, even in the special case where $\mathscr{V} \cup \mathcal{C} \subset \mathbb{R}^2$ and the distances are given by standard Euclidean distances.*

Proof. Let (V, k) be the given instance of 2-Independent Set In Unit Disk Graph, where $V \subset \mathbb{R}^2$. We create an instance of Obnox-Egal-Median-CS as follows. For every point $p \in V$, we add a voter and a candidate co-located at the point in $\mathbb{R}^2$ at the point p. Let $\mathscr{V}$ and $\mathcal{C}$ be the resulting sets of n voters and n candidates, and the value of k remains unchanged. Without loss of generality, we assume that $k \geq 2$. We set $\lambda = k-1$. We prove the following lemma. Note that due to the strict inequality, this does not quite fit the definition of Obnox-Egal-Median-CS. Subsequently, we discuss how to modify the construction so that this issue is alleviated. In the following proof, we use $d_e(\cdot, \cdot)$ to denote the Euclidean distance and $d_G(\cdot, \cdot)$ to denote the shortest-path distance in the unit disk graph G.

▶ **Lemma 12.** *S is a 2-independent set of size k in G if and only if for the corresponding set $S' \subseteq \mathcal{C}$, it holds that, for every voter $v \in \mathscr{V}$, $d_e^\lambda(v, S') > 2$.*

Proof. In the forward direction, let S be a 2-independent set of size k in G, and let S' be as defined above. Suppose for the contradiction that there exists a voter v for which $d_e^\lambda(v, S') \leq 2$. That is, there exists two distinct candidates $c_1, c_2 \in S'$ such that $d_e(v, c_1) \leq 2$, and $d_e(v, c_2) \leq 2$. We consider two cases, depending on whether v is co-located with either of c_1 or c_2, or not. Suppose v is co-located with c_1 (w.l.o.g., the c_2 case is symmetric). Then, let p_1 and p_2 be the points in $S \subseteq P$ corresponding to v and c_2 respectively. However, since $d_e(p_1, p_2) \leq 2$, (p_1, p_2) is an edge in G, which contradicts the 2-independence of S. In the second case, v is not co-located with c_1 as well as c_2. Even in this case, let q, p_1, p_2 be the points in P corresponding to v, c_1, and c_2 respectively. Note that q, p_1, p_2 are distinct, and $p_1, p_2 \in S$. However, since $d_e(q, p_1) \leq 2, d_e(q, p_2) \leq 2$, (q, p_1) and (q, p_2) are edges in G, which again contradicts the 2-independence of S, as $d_G(p_1, p_2) = 2$.

In the reverse direction, let $S' \subseteq \mathcal{C}$ be a subset of candidates such that for each voter $v \in \mathscr{V}$, $d_e^\lambda(v, S') > 2$. Let $S \subseteq P$ be the corresponding points of S', and suppose for contradiction that S is not a 2-independent set in G, which implies that there exist two distinct $p_1, p_2 \in S$ such that $d_G(p_1, p_2) \leq 2$. Let c_1, c_2 be the candidates in S' corresponding to p_1 and p_2 respectively. Again, we consider two cases. First, suppose that $d_G(p_1, p_2) = 1$, i.e., (p_1, p_2) is an edge in G. Then, let v_1 be the voter co-located at p_1. Then, for v_1, $d_e(v_1, c_1) = 0$, and $d_e(v_1, c_2) \leq 2$, since (p_1, p_2) is an edge. This contradicts that $d_e^\lambda(v_1, S') > 2$. In the second case, suppose $d_G(p_1, p_2) = 2$, then let $q \in P$ be a common neighbor of p_1 and p_2 in G, and let $v_q \in \mathscr{V}$ be the voter co-located to q. Again, note that $d_e(v_q, c_1) \leq 2$ and $d_e(v_q, c_2) \leq 2$, which contradicts that $d_e^\lambda(v_q, S') > 2$. ◀

Let $t \coloneqq \min_{p,q \in P : d_e(p,q) > 2} d_e(p, q)$. That is, t is the smallest Euclidean distance between non-neighbors in G. By definition, for any $p', q' \in P$ such that $d_e(p', q') > 2$, it holds that $d_e(p', q') \geq t$. Now, we observe that the proof of Lemma 12 also works after changing the condition $d_e^\lambda(v, S') > 2$ to $d_e^\lambda(v, S') \geq t$. Note that there exists points p, q such that $d_G(p, q) > 2$, and hence $d_e(p, q) > 2$, otherwise, it is a trivial no-instance of 2-Independent Set In Unit Disk Graph. ◀

Approximation Hardness in Graph Metric. The reduction is same as in Section 2.2. Here, instead of Hitting Set, we give a reduction from the Multi-Hitting Set problem, where each set needs to be hit at least $\lambda \geq 1$ times for some constant λ. It can be easily seen that this is a generalization of Hitting Set and is also NP-complete [57] (for an easy reduction from Hitting Set, simply add $\lambda - 1$ "effectively dummy" sets that contain all the original elements) Thus, we have the following result.

▶ **Theorem 13.** *For any fixed $1 \leq \lambda < k$,* Obnox-Egal-Median-CS *is* NP*-hard. Furthermore, for any fixed $1 \leq \lambda < k$, and for any $\alpha \geq 1/3$,* Obnox-Egal-Median-CS *does not admit a polynomial time α-approximation algorithm, unless* P = NP*. Furthermore,* Obnox-Egal-Median-CS *does not admit an FPT-approximation algorithm parameterized by k with an approximation guarantee of $\alpha \geq 1/3$, unless $FPT = W[2]$.*

3.2 FPT-AS in Euclidean and Doubling Spaces

In this section, we design an FPT approximation scheme for the inputs in $\mathbb{R}^d$, parameterized by λ, d, and ϵ. In fact, the same arguments can be extended to metric spaces of doubling dimension d. However, we focus on $\mathbb{R}^d$ for the ease of exposition, and discuss the case of doubling spaces at the end.

In the subsequent discussions, we say that $S \subseteq \mathcal{C}$ is a *solution* if it satisfies the following two properties: (i) $|S| \geq \lambda$, and (ii) for each $v \in \mathscr{V}$, $d^\lambda(v, S) \geq t$. For any given instance of Obnox-Egal-Median-CS, we state the following simple observations.

▶ **Observation 14.** *If there exists $S \subseteq \mathcal{C}$ of size at least λ, such that each $c \in S$ is at distance at least t from each $v \in \mathscr{V}$, then S is a solution.*

▶ **Observation 15 (♣).** *A subset $S \subseteq \mathcal{C}$ of $\lambda + 1$ points that are pairwise $2t$ distance away from each other is a solution.*

First, note that we can assume $\lambda + 1 \leq k$ – otherwise $\lambda = k$ case can be easily solved in polynomial-time using the argument mentioned in the preliminaries. Now, if a set $S \subseteq \mathcal{C}$ with $|S| \geq \lambda + 1$ satisfying the conditions of Observation 15 exists, then we can immediately augment it with arbitrary $k - (\lambda + 1)$ candidates from $\mathcal{C} \setminus S$, yielding a solution of size k. Thus, henceforth, we may assume that any subset $S \subseteq \mathcal{C}$ consisting of candidates that are pairwise $2t$ distance away from each other, has size at most λ.

Let us fix N to be one such maximal subset – note that we can compute N in polynomial time using a greedy algorithm. The following observation follows from the maximality of N.

▶ **Observation 16.** *Any point $p \in \mathcal{C}$ must be in $\bigcup_{c \in N} B(c, 2t)$. In other words, each $p \in \mathcal{C}$ is inside a ball of radius $2t$ centered at one of the points in N.*

This simple observation, combined with the following covering-packing property of the underlying Euclidean space will allow our algorithm to pick points from the vicinity of those chosen by an optimal algorithm.

▶ **Proposition 17 (♣).** *In $\mathbb{R}^d$, for any $0 < r_1 < r_2$, a ball of radius r_2 can be covered by $\alpha_d \cdot (r_2/r_1)^d$ balls of radius r_1. Here, α_d is a constant that depends only on the dimension d.*

Next, for each $c \in N$, we find an "$\epsilon t/4$-net" inside the ball $B(c, 2t)$, i.e., a maximal subset $Q \subseteq B(c, 2t) \cap \mathcal{C}$, such that (i) for any distinct $c_1, c_2 \in Q$, $d(c_1, c_2) > \epsilon t/4$, and (ii) For each $c_1 \in \mathcal{C} \setminus Q$, there exists some $c_2 \in Q$, such that $d(c_1, c_2) \leq \epsilon t/4$. Note that Q can be computed using a greedy algorithm. Next, we iterate over each $c' \in Q$, and mark the $\lambda - 1$ closest unmarked candidates to c' that are not in Q (if any). Let $R_c := Q \cup M$, where M denotes the set of marked candidates during the second phase.

▶ **Observation 18 (♣).** *For each $c \in N$, $|R_c| \leq \mathcal{O}_d(\lambda \cdot (1/\epsilon)^d)$, where $\mathcal{O}_d(\cdot)$ hides a constant that depends only on the dimension d.*

Let $S' = \bigcup_{c \in N} R_c$. Finally, let $S := N \cup S'$, and note that $|S| \leq \mathcal{O}_d(\lambda^2 \cdot (1/\epsilon)^d)$, where $\mathcal{O}_d(\cdot)$ notation hides constants that depend only on d. Now we consider two cases.

- If $|S| \leq k$, then we augment it with arbitrary $k - |S|$ candidates from $\mathcal{C} \setminus S$, and output the resulting set.
- If $|S| > k$, then try all possible k-sized subsets of S to see if it constitutes a solution. There can be at most $\binom{|S|}{k} < 2^{|S|} = 2^{\mathcal{O}_d(\lambda^2(1/\epsilon)^d)}$ sets to check resulting in time $2^{\mathcal{O}_d(\lambda^2(1/\epsilon)^d)} \cdot (|\mathscr{V}| + |\mathcal{C}|)^{\mathcal{O}(1)}$.

The next lemma completes the proof. We prove it by comparing S to an optimal solution, and show that for every point in the latter there is a point in the vicinity that is present in S.

▶ **Lemma 19.** *If $|S| > k$, then there is a subset $Q \subseteq S$ of size k that constitutes a solution.*

Proof. Suppose that there is an optimal solution, denoted by O, that contains k points and for each point $v \in \mathscr{V}$ there exist at least λ points in O (called *representatives*, $\mathcal{R}(v)$) that are at least t distance away from v. Let $\mathcal{R} = \bigcup_{v \in \mathscr{V}} \mathcal{R}(v)$ denote the set of all representatives.

First, due to Observation 16, each $c \in \mathcal{R}$ is inside some $B(c', 2t)$ for some $c' \in N$. Let $\tilde{c} \in Q$ be the closest (breaking ties arbitrarily) candidate to c from Q. By construction, $d(\tilde{c}, c) \leq t\epsilon/4$. Let $A(\tilde{c}) \subseteq \mathcal{R}$ be the points for which $\tilde{c}$ is the closest point in $R_{c'}$ (breaking ties arbitrarily).

Case 1: $|A(\tilde{c})| \leq \lambda$. In this case, we claim that for each $c_1 \in A(\tilde{c})$, we have added a unique c_2 to $R_{c'} \subseteq S'$ such that $d(c_1, c_2) \leq \epsilon t$. First, if $A(\tilde{c}) \subseteq R_{c'}$, then the claim is trivially true (the required bijection is the identity mapping). Otherwise, there exists some $c_1 \in A(\tilde{c})$ such that $c_1 \notin R_c$. In particular, this means that c_1 was not marked during the iteration of the marking phase corresponding to $\tilde{c} \in Q$. This means that at least $\lambda - 1$ other candidates with distance at most $\epsilon t/4$ from $\tilde{c}$ were marked. For any of these marked candidates c_2, it holds that $d(c_1, c_2) \leq d(c_1, \tilde{c}) + d(\tilde{c}, c_2) \leq \epsilon t/2 \leq \epsilon t$. Accounting for $\tilde{c}$, this implies that, for each $c_1 \in A(\tilde{c})$, there are at least $\lambda \geq |A(\tilde{c})|$ distinct candidates in R_c within distance ϵt. Let $Q(\tilde{c}) \subseteq S'$ denote an arbitrary such subset of size λ in this case.

Case 2: $|A(\tilde{c})| > \lambda$. In this case, let $A'(\tilde{c}) \subset A(\tilde{c})$ be an arbitrary subset of size λ. We claim that $A'(\tilde{c})$ is sufficient for any solution. In particular, consider a $v \in \mathscr{V}$ and $c \in A(\tilde{c}) \setminus A'(\tilde{c})$ such that c is a representative of v. We claim that for all $c' \in A'(\tilde{c})$, $d(\tilde{c}, c') \geq (1 - \epsilon/2)t$, which follows from $d(c, c') \leq \epsilon t/2$. Thus, the λ points of $A'(\tilde{c})$ constitute an approximate set of representatives for v. Now, by using the argument from the previous paragraph w.r.t. $A'(\tilde{c})$, we can obtain a set $Q(\tilde{c})$ of size λ, such that for any voter $v \in V$ such that $\mathcal{R}(v) \cap A(\tilde{c}) \neq \emptyset$, every point in $Q(\tilde{c})$ is at distance at least $(1 - \epsilon)t$ from v.

Finally, let Q denote the union of all sets $Q(\tilde{c})$ defined in this manner (note that $Q(\tilde{c})$ is defined only if $A(\tilde{c}) \neq \emptyset$). First, by construction, for each $v \in \mathscr{V}$, Q contains at least λ points at distance at least $(1 - \epsilon)t$. Next, $Q \subseteq R'$ and $|Q| \leq k$ since for each point in $\mathcal{R}$, we add at most one point to Q. Now, if $|Q| < k$, then we can simply add arbitrary $k - |Q|$ points to obtain the desired set. ◀

In fact, the covering-packing properties of the underlying metric space that are crucial in our algorithm are abstracted in the following well-known notion.

▶ **Definition 20** (Doubling dimension and doubling spaces). *Let $\mathscr{M} = (P, d)$ be a metric space, where P is a set of points and d is the distance function. We say that $\mathscr{M}$ has* doubling dimension *δ, if for any $p \in P$, and any $r \geq 0$, the ball $B(p, r) := \{q \in P : d(p, q) \leq r\}$ can be covered using at most 2^δ balls of radius $r/2$. If the doubling dimension of a metric space $\mathscr{M}$ is a constant, then we say that $\mathscr{M}$ is a* doubling space.

Note that Euclidean space of dimension d has doubling dimension $O(d)$. By a simple repeated application of the above definition, we obtain the following Proposition 21 that is an analogue of Proposition 17.

▶ **Proposition 21.** *Let $\mathscr{M} = (P, d)$ be a metric space of doubling dimension δ. Then, any ball $B(p, r_2)$ can be covered with $\left(\lceil \frac{r_2}{r_1} \rceil\right)^{\delta}$ balls of radius r_1, where $0 < r_1 \leq r_2$.*

Our algorithm generalizes to metric spaces of doubling dimension δ in a straightforward manner, resulting in the following theorem.

▶ **Theorem 22.** *For any ϵ, $0 < \epsilon < 1$, we have an algorithm that given an instance of* Obnox-Egal-Median-CS *in a metric space of doubling dimension δ, computes a solution of size k such that for every point $v \in \mathscr{V}$ there are at least λ points in the solution that are at distance at least $(1-\epsilon)t$ from v in time* FPT *in $(\epsilon, \lambda, \delta)$. In particular, we obtain this result in Euclidean spaces of dimension d, in time* FPT *in (ϵ, λ, d).*

4 Outlook

In this paper we studied a committee selection problem, where preferences of voters towards candidates was captured via a metric space. In particular, we studied a variant where larger distance corresponds to higher preference for a candidate in comparison to a candidate who is nearer. We showed that our problem is NP-hard in general, and designed some polynomial time algorithms, as well as (parameterized) approximation algorithms. We conclude with some research directions for future study. One of our concrete open question is that Is Obnox-Egal-Median-CS in $\mathbb{R}^2$ for $\lambda > 1$ polynomial-time solvable? In this paper, we considered median scoring rules. It would be interesting to study other scoring rules as well, when the voters and candidates are embedded in a metric space.

Moreover, we note that situations where, for each neighborhood we want exactly λ facilities nearby, and the remaining $k - \lambda$ to be far away, is not handled by this model. This would be the "exact" variant of our problem Obnox-Egal-Median-CS and would be of natural interest.

References

1 Hyung-Chan An and O. Svensson. Recent developments in approximation algorithms for facility location and clustering problems. In *Combinatorial Optimization and Graph Algorithms: Communications of NII Shonan Meetings*, pages 1–19. Springer, 2017.

2 Nima Anari, Moses Charikar, and Prasanna Ramakrishnan. Distortion in metric matching with ordinal preferences. In Kevin Leyton-Brown, Jason D. Hartline, and Larry Samuelson, editors, *Proceedings of the 24th ACM Conference on Economics and Computation, EC 2023, London, United Kingdom, July 9-12, 2023*, pages 90–110. ACM, 2023. doi:10.1145/3580507.3597740.

3 Haris Aziz, Felix Brandt, Edith Elkind, and Piotr Skowron. Computational social choice: The first ten years and beyond. In Bernhard Steffen and Gerhard J. Woeginger, editors, *Computing and Software Science - State of the Art and Perspectives*, volume 10000 of *Lecture Notes in Computer Science*, pages 48–65. Springer, 2019. doi:10.1007/978-3-319-91908-9_4.

4 Haris Aziz, Piotr Faliszewski, Bernard Grofman, Arkadii Slinko, and Nimrod Talmon. Egalitarian committee scoring rules. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 56–62. ijcai.org, 2018. doi:10.24963/IJCAI.2018/8.

5 Haris Aziz, Serge Gaspers, Joachim Gudmundsson, Simon Mackenzie, Nicholas Mattei, and Toby Walsh. Computational aspects of multi-winner approval voting. In Gerhard Weiss, Pinar Yolum, Rafael H. Bordini, and Edith Elkind, editors, *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 107–115. ACM, 2015. URL: http://dl.acm.org/citation.cfm?id=2772896.

6 Nadja Betzler, Robert Bredereck, Jiehua Chen, and Rolf Niedermeier. Studies in computational aspects of voting - A parameterized complexity perspective. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 318–363. Springer, 2012. doi:10.1007/978-3-642-30891-8_16.

7 Nadja Betzler, Arkadii Slinko, and Johannes Uhlmann. On the computation of fully proportional representation. *J. Artif. Intell. Res.*, 47:475–519, 2013. doi:10.1613/JAIR.3896.

8 Steven J Brams, D Marc Kilgour, and M Remzi Sanver. A minimax procedure for electing committees. *Public Choice*, 132:401–420, 2007.

9 Felix Brandt, Markus Brill, and Bernhard Harrenstein. Tournament Solutions. In F Brandt, V Conitzer, U Endriss, J Lang, and A. D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 453–474. Cambridge University Press, 2016.

10 Robert Bredereck, Piotr Faliszewski, Andrzej Kaczmarczyk, Dusan Knop, and Rolf Niedermeier. Parameterized algorithms for finding a collective set of items. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1838–1845. AAAI Press, 2020. doi:10.1609/AAAI.V34I02.5551.

11 Robert Bredereck, Piotr Faliszewski, Andrzej Kaczmarczyk, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Robustness among multiwinner voting rules. *Artif. Intell.*, 290:103403, 2021. doi:10.1016/J.ARTINT.2020.103403.

12 Markus Brill, Piotr Faliszewski, Frank Sommer, and Nimrod Talmon. Approximation algorithms for balancedcc multiwinner rules. In *AAMAS '19*, pages 494–502, 2019. URL: http://dl.acm.org/citation.cfm?id=3331732.

13 Laurent Bulteau, Gal Shahaf, Ehud Shapiro, and Nimrod Talmon. Aggregation over metric spaces: Proposing and voting in elections, budgeting, and legislation. *J. Artif. Intell. Res.*, 70:1413–1439, 2021. doi:10.1613/JAIR.1.12388.

14 Deeparnab Chakrabarty, Luc Côté, and Ankita Sarkar. Fault-tolerant k-supplier with outliers. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science, STACS 2024, March 12-14, 2024, Clermont-Ferrand, France*, volume 289 of *LIPIcs*, pages 23:1–23:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.STACS.2024.23.

15 Deeparnab Chakrabarty and Chaitanya Swamy. Interpolating between k-median and k-center: Approximation algorithms for ordered k-median. 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 29:1–29:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.29.

16 John R Chamberlin and Paul N Courant. Representative deliberations and representative decisions: Proportional representation and the borda rule. *American Political Science Review*, 77(3):718–733, 1983.

17 Hau Chan, Aris Filos-Ratsikas, Bo Li, Minming Li, and Chenhao Wang. Mechanism design for facility location problems: A survey. In *Proceedings of IJCAI*, pages 4356–4365, 2021. doi:10.24963/IJCAI.2021/596.

18 Moses Charikar, Kangning Wang, Prasanna Ramakrishnan, and Hongxun Wu. Breaking the metric voting distortion barrier. 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 1621–1640. SIAM, 2024. doi:10.1137/1.9781611977912.65.

19 Shiri Chechik and David Peleg. The fault-tolerant capacitated k-center problem. *Theor. Comput. Sci.*, 566:12–25, 2015. doi:10.1016/J.TCS.2014.11.017.

20 Jiehua Chen and Sven Grottke. Small one-dimensional euclidean preference profiles. *Soc. Choice Welf.*, 57(1):117–144, 2021. doi:10.1007/S00355-020-01301-Y.

21 Richard L Church and Zvi Drezner. Review of obnoxious facilities location problems. *Comput. Oper. Res.*, 138:105468, 2022. doi:10.1016/J.COR.2021.105468.

22 Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, and Chris Schwiegelshohn. Breaching the 2 LMP approximation barrier for facility location with applications to k-median. In *Proceedings of SODA*, pages 940–986, 2023. doi:10.1137/1.9781611977554.CH37.

23 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.

24 Eva Michelle Deltl, Till Fluschnik, and Robert Bredereck. Algorithmics of egalitarian versus equitable sequences of committees. In *IJCAI 2023*, pages 2651–2658, 2023. doi:10.24963/IJCAI.2023/295.

25 Robert G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

26 Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990. doi:10.1145/77635.77639.

27 Edith Elkind, Piotr Faliszewski, Jean-François Laslier, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. What do multiwinner voting rules do? an experiment over the two-dimensional euclidean domain. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 494–501. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.10612.

28 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner rules on paths from k-borda to chamberlin-courant. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 192–198. ijcai.org, 2017. doi:10.24963/IJCAI.2017/28.

29 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner voting: A new challenge for social choice theory. *Trends in computational social choice*, 74(2017):27–47, 2017.

30 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Multiwinner analogues of the plurality rule: axiomatic and algorithmic perspectives. *Soc. Choice Welf.*, 51(3):513–550, 2018. doi:10.1007/S00355-018-1126-4.

31 Piotr Faliszewski, Piotr Skowron, Arkadii Slinko, and Nimrod Talmon. Committee scoring rules: Axiomatic characterization and hierarchy. *ACM Trans. Economics and Comput.*, 7(1):3:1–3:39, 2019. doi:10.1145/3296672.

32 Piotr Faliszewski, Piotr Skowron, and Nimrod Talmon. Bribery as a measure of candidate success: Complexity results for approval-based multiwinner rules. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 6–14. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3091133.

33 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer, 2006. doi:10.1007/3-540-29953-X.

34 Michal Tomasz Godziszewski, Pawel Batko, Piotr Skowron, and Piotr Faliszewski. An analysis of approval-based committee rules for 2d-euclidean elections. In *Proceedings of AAAI*, pages 5448–5455, 2021. doi:10.1609/AAAI.V35I6.16686.

35 Kishen N. Gowda, Thomas W. Pensyl, Aravind Srinivasan, and Khoa Trinh. Improved bi-point rounding algorithms and a golden barrier for k-median. In *Proceedings of SODA*, pages 987–1011, 2023. doi:10.1137/1.9781611977554.CH38.

36 Sushmita Gupta, Tanmay Inamdar, Pallavi Jain, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. When far is better: The chamberlin-courant approach to obnoxious committee selection. *CoRR*, abs/2405.15372, 2024. doi:10.48550/arXiv.2405.15372.

37 Sushmita Gupta, Pallavi Jain, Saket Saurabh, and Nimrod Talmon. Even more effort towards improved bounds and fixed-parameter tractability for multiwinner rules. In *Proceedings of IJCAI*, pages 217–223, 2021. doi:10.24963/IJCAI.2021/31.

38 Aviram Imber, Jonas Israel, Markus Brill, Hadas Shachnai, and Benny Kimelfeld. Spatial voting with incomplete voter information. In *AAAI 2024*, pages 9790–9797, 2024. doi:10.1609/AAAI.V38I9.28838.

39 Tanmay Inamdar and Kasturi R. Varadarajan. Fault tolerant clustering with outliers. In Evripidis Bampis and Nicole Megow, editors, *Approximation and Online Algorithms - 17th International Workshop, WAOA 2019, Munich, Germany, September 12-13, 2019, Revised Selected Papers*, volume 11926 of *Lecture Notes in Computer Science*, pages 188–201. Springer, 2019. doi:10.1007/978-3-030-39479-0_13.

40 Sangram Kishor Jena, Ramesh K. Jallu, Gautam K. Das, and Subhas C. Nandy. The maximum distance-d independent set problem on unit disk graphs. In Jianer Chen and Pinyan Lu, editors, *Frontiers in Algorithmics - 12th International Workshop, FAW 2018, Guangzhou, China, May 8-10, 2018, Proceedings*, volume 10823 of *Lecture Notes in Computer Science*, pages 68–80. Springer, 2018. doi:10.1007/978-3-319-78455-7_6.

41 Yusuf Hakan Kalayci, David Kempe, and Vikram Kher. Proportional representation in metric spaces and low-distortion committee selection. In *AAAI 2024*, pages 9815–9823, 2024. doi:10.1609/AAAI.V38I9.28841.

42 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of CCC*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

43 Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k-center problems. *Theor. Comput. Sci.*, 242(1-2):237–245, 2000. doi:10.1016/S0304-3975(98)00222-9.

44 Steven Klein. On the egalitarian value of electoral democracy. *Political Theory*, page 00905917231217133, 2023.

45 Nirman Kumar and Benjamin Raichel. Fault tolerant clustering revisited. In *Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG 2013, Waterloo, Ontario, Canada, August 8-10, 2013*. Carleton University, Ottawa, Canada, 2013. URL: http://cccg.ca/proceedings/2013/papers/paper_36.pdf.

46 Martin Lackner and Piotr Skowron. *Multi-winner voting with approval preferences*. Springer Nature, 2023.

47 Alexander Lam, Haris Aziz, Bo Li, Fahimeh Ramezani, and Toby Walsh. Proportional fairness in obnoxious facility location. In Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum, editors, *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, pages 1075–1083. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2024. doi:10.5555/3635637.3662963.

48 Jean-François Laslier. Spatial approval voting. *Political Analysis*, 14(2):160–185, 2006.

49 Hong Liu and Jiong Guo. Parameterized complexity of winner determination in minimax committee elections. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 341–349. ACM, 2016. URL: http://dl.acm.org/citation.cfm?id=2936975.

50 Jirí Matousek. *Lectures on discrete geometry*, volume 212 of *Graduate texts in mathematics*. Springer, 2002.

51 Ivan-Aleksandar Mavrov, Kamesh Munagala, and Yiheng Shen. Fair multiwinner elections with allocation constraints. In *Proceedings of EC*, pages 964–990, 2023. doi:10.1145/3580507.3597685.

52 Neeldhara Misra, Arshed Nabeel, and Harman Singh. On the parameterized complexity of minimax approval voting. In Gerhard Weiss, Pinar Yolum, Rafael H. Bordini, and Edith Elkind, editors, *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 97–105. ACM, 2015. URL: http://dl.acm.org/citation.cfm?id=2772895.

53 Burt L Monroe. Fully proportional representation. *American Political Science Review*, 89(4):925–940, 1995.

54 Kamesh Munagala, Yiheng Shen, Kangning Wang, and Zhiyi Wang. Approximate core for committee selection via multilinear extension and market clearing. In *Proceedings of SODA*, pages 2229–2252, 2022. doi:10.1137/1.9781611977073.89.

55 Kamesh Munagala, Zeyu Shen, and Kangning Wang. Optimal algorithms for multiwinner elections and the chamberlin-courant rule. In *EC '21*, pages 697–717. ACM, 2021. doi:10.1145/3465456.3467624.

56 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

57 Sridhar Rajagopalan and Vijay V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM J. Comput.*, 28(2):525–540, 1998. doi:10.1137/S0097539793260763.

58 John Rawls. *A Theory of Justice: Original Edition*. Harvard University Press, 1971. URL: http://www.jstor.org/stable/j.ctvjf9z6v.

59 John Rawls. *Justice as Fairness: Political Not Metaphysical*, pages 145–173. Palgrave Macmillan UK, London, 1991. doi:10.1007/978-1-349-21763-2_10.

60 Piotr Skowron. FPT approximation schemes for maximizing submodular functions. *Inf. Comput.*, 257:65–78, 2017. doi:10.1016/J.IC.2017.10.002.

61 Piotr Skowron and Piotr Faliszewski. Chamberlin-courant rule with approval ballots: Approximating the maxcover problem with bounded frequencies in FPT time. *J. Artif. Intell. Res.*, 60:687–716, 2017. doi:10.1613/JAIR.5628.

62 Piotr Skowron, Piotr Faliszewski, and Jérôme Lang. Finding a collective set of items: From proportional multirepresentation to group recommendation. *Artif. Intell.*, 241:191–216, 2016. doi:10.1016/J.ARTINT.2016.09.003.

63 Piotr Krzysztof Skowron and Edith Elkind. Social choice under metric preferences: Scoring rules and STV. In *Proceedings of AAAI*, pages 706–712. AAAI Press, 2017. doi:10.1609/AAAI.V31I1.10591.

64 Chinmay Sonar, Subhash Suri, and Jie Xue. Multiwinner elections under minimax chamberlin-courant rule in euclidean space. In *Proceedings of IJCAI*, pages 475–481, 2022. doi:10.24963/IJCAI.2022/68.

65 Chinmay Sonar, Subhash Suri, and Jie Xue. Fault tolerance in euclidean committee selection. In *ESA 2023*, volume 274, pages 95:1–95:14, 2023. doi:10.4230/LIPICS.ESA.2023.95.

66 Gogulapati Sreedurga, Mayank Ratan Bhardwaj, and Yadati Narahari. Maxmin participatory budgeting. In *IJCAI 2022*, pages 489–495, 2022. doi:10.24963/IJCAI.2022/70.

67 Arie Tamir. Obnoxious facility location on graphs. *SIAM J. Discret. Math.*, 4(4):550–567, 1991. doi:10.1137/0404048.

68 Thorvald N. Thiele. Om flerfoldsvalg. *Oversigt over det Kongelige Danske Videnskabernes Selskabs Forhandlinger*, pages 415–441, 1895.

69 Yongjie Yang. On the complexity of calculating approval-based winners in candidates-embedded metrics. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 585–591, 2022. doi:10.24963/IJCAI.2022/83.

70 Yongjie Yang and Jianxin Wang. Parameterized complexity of multi-winner determination: More effort towards fixed-parameter tractability. In Elisabeth André, Sven Koenig, Mehdi Dastani, and Gita Sukthankar, editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 2142–2144. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018. URL: `http://dl.acm.org/citation.cfm?id=3238099`.

71 Vikram Kher Yusuf Hakan Kalayci, David Kempe. Proportional representation in metric spaces and low-distortion committee selection. In *Proceedings of AAAI*, 2024.

72 Aizhong Zhou, Yongjie Yang, and Jiong Guo. Parameterized complexity of committee elections with dichotomous and trichotomous votes. In Edith Elkind, Manuela Veloso, Noa Agmon, and Matthew E. Taylor, editors, *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 503–510. International Foundation for Autonomous Agents and Multiagent Systems, 2019. URL: `http://dl.acm.org/citation.cfm?id=3331733`.

Better Boosting of Communication Oracles, or Not

Nathaniel Harms
EPFL, Lausanne, Switzerland

Artur Riazanov
EPFL, Lausanne, Switzerland

Abstract

Suppose we have a two-party communication protocol for f which allows the parties to make queries to an oracle computing g; for example, they may query an EQUALITY oracle. To translate this protocol into a randomized protocol, we must replace the oracle with a randomized subroutine for solving g. If q queries are made, the standard technique requires that we boost the error of each subroutine down to $O(1/q)$, leading to communication complexity which grows as $q \log q$. For which oracles g can this naïve boosting technique be improved?

We focus on the oracles which can be computed by constant-cost randomized protocols, and show that the naïve boosting strategy can be improved for the EQUALITY oracle but not the 1-HAMMING DISTANCE oracle. Two surprising consequences are (1) a new example of a problem where the cost of computing k independent copies grows superlinear in k, drastically simplifying the only previous example due to Blais & Brody (CCC 2019); and (2) a new proof that EQUALITY is not complete for the class of constant-cost randomized communication (Harms, Wild, & Zamaraev, STOC 2022; Hambardzumyan, Hatami, & Hatami, Israel Journal of Mathematics 2022).

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases oracles, error reduction, communication complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.25

Funding *Nathaniel Harms*: Supported by an NSERC postdoctoral fellowship and the Swiss State Secretariat for Education, Research, and Innovation (SERI) under contract number MB22.00026.
Artur Riazanov: Supported by the Swiss State Secretariat for Education, Research, and Innovation (SERI) under contract number MB22.00026.

1 Introduction

We typically require that randomized algorithms succeed with probability 2/3, since the probability can be boosted to any $1 - \delta$ by taking a majority vote of $O(\log(1/\delta))$ repetitions. If many randomized subroutines are used within an algorithm, the probability of error may accumulate, and one may apply standard boosting to each subroutine to bring the error probability down to an acceptable level. We wish to understand when this is necessary, in the setting of communication complexity.

Suppose two parties, Alice and Bob, wish to compute a function $f(x, y)$ on their respective inputs x and y, using as little communication as possible, and they have access to a shared (i.e. public) source of randomness. A convenient way to design a randomized communication protocol to compute $f(x, y)$ is to design a *deterministic* protocol, but assume that Alice and Bob have access to an *oracle* (in other words, a subroutine) which computes a certain problem g that itself has an efficient randomized protocol.

▶ **Example 1.** The EQUALITY problem is the textbook example of a problem with an efficient randomized protocol [19, 23]: Given inputs $a, b \in [N]$, two parties can decide (with success probability 3/4) whether $a = b$, using only 2 bits of (public) randomized communication, regardless of the domain size N. So, to design a randomized protocol for solving another problem $f(x, y)$, we may assume that the two parties have access to an EQUALITY oracle.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 25; pp. 25:1–25:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

For example, suppose Alice and Bob have vertices x and y in a shared tree T, and wish to decide whether x and y are adjacent in T. If $p(x)$ denotes the parent of x in T, then Alice and Bob can decide adjacency using two EQUALITY queries: "$x = p(y)$?" and "$y = p(x)$?"

▶ **Example 2.** The 1-HAMMING DISTANCE communication problem is denoted HD_1 and defined as $\mathrm{HD}_1^n(x, y) = 1$ if $x, y \in \{0,1\}^n$ differ on exactly 1 bit, and 0 otherwise. It has a constant-cost randomized protocol, but unlike adjacency in trees, this protocol *cannot* be expressed as a deterministic protocol using the EQUALITY oracle [12, 13].

Using oracles makes the protocol simpler, and also makes it clearer how and why randomness is used in the protocol, which provides more insight into randomized communication (see e.g. [5, 12, 13, 8] for recent work using oracles to understand randomized communication). But when we replace the *oracle* for g with a randomized protocol for g, we must compensate for the probability that the randomized protocol produces an incorrect answer. Write $\mathsf{D}^g(f)$ for the optimal cost of a deterministic communication protocol for f using an oracle for g (where the players pay cost 1 to query the oracle). Write $\mathsf{R}_\delta(f)$ for the optimal cost of a randomized protocol for f with error δ. Then the inequality

$$\forall f\ , \qquad \mathsf{R}_\delta(f) = O\left(\mathsf{D}^g(f) \cdot \mathsf{R}_{1/4}(g) \cdot \log\left(\frac{\mathsf{D}^g(f)}{\delta}\right)\right) \tag{1}$$

follows from standard boosting: if there are $q = \mathsf{D}^g(f)$ queries made by the protocol in the worst case, then we obtain a randomized protocol by simulating each of the q queries to g using a protocol for g with error $\approx \delta/q$, sending $\mathsf{R}_{\delta/q}(g) = O(\mathsf{R}_{1/4}(g) \cdot \log(q/\delta))$ bits of communication for each query. But is it possible to improve on this naïve bound? The main question of this paper is:

▶ **Question 3.** *For which oracle functions g can Equation* (1) *be improved?*

We focus on the oracles g which have *constant-cost* randomized communication protocols, like EQUALITY. Randomized communication is quite poorly understood, with many fundamental questions remaining open even when restricted to the surprisingly rich class of constant-cost problems. Many recent works have focused on understanding these extreme examples of efficient randomized computation; see [12, 13, 11, 6, 16, 14, 8] and the survey [15]. And indeed some of these works [12, 15] use Equation (1) specifically for the EQUALITY oracle. So this is a good place to begin studying Question 3. Our main result is:

▶ **Theorem 4** (Informal; see Theorems 11 and 14)**.** *Equation* (1) *can be improved for the EQUALITY oracle, but it is (nearly) tight for the 1-HAMMING DISTANCE oracle.*

This has some unexpected consequences, described below, and also answers Question 3 for all *known* constant-cost problems.

Every known constant-cost problem g satisfies either $\mathsf{D}^{\mathrm{EQ}}(g) = O(1)$ or $\mathsf{D}^g(\mathrm{HD}_1) = O(1)$ ([7] gives a survey of all known problems). Therefore we answer Question 3 for all *known* constant-cost oracles. Towards an answer for *all* constant-cost oracles, we show that the technique which allows us to improve Equation (1) works *only* for the EQUALITY oracle (Proposition 21).

Our main proof also has two other surprising consequences:

Direct sums. Direct sum questions ask how the complexity of computing k copies of a problem grows with k (see e.g. [9, 18, 1, 3]). Recently, [3] answered a long-standing question of [9] by providing the first example of a problem where the communication complexity of computing k independent copies grows *superlinearly* with k. Their example is specially designed to exhibit this behaviour and goes through the query-to-communication lifting technique. In our investigation of Question 3, we show that computing k independent copies of the drastically simpler, constant-cost 1-Hamming Distance problem requires $\Omega\left(\frac{k \log k}{\log \log k}\right)$ bits of communication (Theorem 14). As a corollary, we also show a similar direct sum theorem for randomized parity decision trees (Corollary 15).

Oracle separations. In an effort to better understand the power of randomness in communication, recent works have studied the relative power of different oracles. [5] show that the Equality oracle is not powerful enough to simulate the standard communication complexity class BPP (i.e. $N \times N$ communication matrices with cost $\operatorname{poly} \log \log N$), i.e. Equality is not *complete* for BPP. [12, 13] showed that Equality is also not complete for the class BPP^0 of *constant-cost* communication problems, because 1-Hamming Distance does not reduce to it; and [8] show that there is *no* complete problem for BPP^0. There are many lower-bound techniques for communication complexity, but not many lower bounds for communication with *oracles*. Our investigation of Question 3 gives an unexpected new proof of the separation between the Equality and 1-Hamming Distance oracles; our proof is "algorithmic", and arguably simpler than the Ramsey-theoretic proof of [13] or the Fourier-analytic proof of [12].

Further Motivation, Discussion & Open Problems

Let's say a constant-cost oracle function g has *better boosting* if

$$\forall f \ : \qquad \mathsf{R}_\delta(f) = O(\mathsf{D}^g(f) + \log(1/\delta)) \,.$$

We showed that among the *currently-known* constant-cost oracle functions g, better boosting is possible if and only if $\mathsf{D}^{\mathrm{EQ}}(g) = O(1)$, and we observed that among *all* constant-cost oracles, only the Equality oracle satisfies the properties used to prove Theorem 11. So, permit us the following conjecture:

▶ **Conjecture 5.** *An oracle function* $g \in \mathsf{BPP}^0$ *has better boosting if and only if* $\mathsf{D}^{\mathrm{EQ}}(g) = O(1)$.

To disprove this conjecture, we need a new example of a constant-cost (total) communication problem that is not somehow a generalization of 1-Hamming Distance. Such an example would be very interesting, so in that regard we hope the conjecture is false.

One more motivation of the current study is to find an approach towards a question of [14] about the intersection between communication complexity classes $\mathsf{UPP}^0 \cap \mathsf{BPP}^0$, where UPP^0 denotes the class of problems with bounded sign-rank, or equivalently, constant-cost *unbounded-error* randomized protocols [22]. Writing EQ^0 for the class of problems g where $\mathsf{D}^{\mathrm{EQ}}(g) = O(1)$, [14] asks:

▶ **Question 6** ([14]). *Is* $\mathsf{UPP}^0 \cap \mathsf{BPP}^0 = \mathsf{EQ}^0$*?*

This question seems challenging; as noted in [14], a positive answer would imply other conjectures about $\mathsf{UPP}^0 \cap \mathsf{BPP}^0$, notably the conjecture of [16] that 1-Hamming Distance does not belong to UPP^0, which would be the first example of a problem in $\mathsf{BPP}^0 \setminus \mathsf{UPP}^0$.

[16] showed that all known lower-bound techniques against UPP^0 fail to prove this. But a positive answer to Question 6 implies that all oracles in $\mathsf{UPP}^0 \cap \mathsf{BPP}^0$ have better boosting, so a weaker question is:

▶ **Question 7.** *Do all oracles in* $\mathsf{UPP}^0 \cap \mathsf{BPP}^0$ *have better boosting?*

Because of Theorem 4, this weaker question would also suffice to prove that 1-Hamming Distance does not belong to UPP^0. It is not clear to us whether Question 7 is easier to answer than Question 6. If the answer to Question 7 is negative (i.e. there is an oracle in $\mathsf{UPP}^0 \cap \mathsf{BPP}^0$ which does not have better boosting), then either Conjecture 5 or Question 6 is false.

Similar questions about probability boosting were studied recently for query complexity in [2] who focused on the properties of the *outer* function f of which allow for better boosting to compute $f \circ g^{\otimes k}$, whereas one may think of our oracles as the *inner* functions. We may rephrase Theorem 11 as a "composition theorem" which says that for any function $f\colon \{0,1\}^k \to \{0,1\}$, the composed function $f \circ (\textsc{Eq})^{\otimes k}$ which applies f to the result of k instances of Equality has communication cost

$$\mathsf{R}_\delta(f \circ (\textsc{Eq})^{\otimes k}) = O(\mathsf{DT}(f) + \log(1/\delta)) \tag{2}$$

where $\mathsf{DT}(f)$ is the decision-tree depth of f. We prefer the statement in Theorem 11 because it more clearly differentiates between the *protocol* and the *problem*. To see what we mean, consider taking f to be the And function; the immediate consequence of Equation (2) is that $\mathsf{R}_{1/4}(\textsc{And} \circ (\textsc{Eq})^{\otimes k}) = O(k)$, whereas the immediate consequence of Theorem 11 is that $\mathsf{R}_{1/4}(\textsc{And} \circ (\textsc{Eq})^{\otimes k}) = O(1)$ because this function can be computed using 1 Equality query. To get the same result from Equation (2) one must rewrite the *problem* $\textsc{And} \circ (\textsc{Eq})^{\otimes k}$ as a different decision tree over different inputs.

2 Definitions: Communication Problems and Oracles

We will use some non-standard definitions that are more natural for constant-cost problems. These definitions come from e.g. [5, 13, 12, 14, 8].

It is convenient to define a *communication problem* as a set $\mathcal{P}$ of Boolean matrices, closed under row and column permutations. The more standard definition has one fixed function $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ for each input size n, with communication matrix $M_f \in \{0,1\}^{2^n \times 2^n}$, whereas we will think of a communication problem $\mathcal{P}$ as possibly containing many different communication matrices $M \in \{0,1\}^{N \times N}$ on each domain size N. (In the adjacency-in-trees problem, Example 1, there are many different trees on N vertices, which define many different communication matrices.)

For a fixed matrix $M \in \{0,1\}^{N \times N}$ and parameter $\delta < 1/2$, we write $\mathsf{R}_\delta(M)$ for the two-way, public-coin randomized communication complexity of M. For a communication problem $\mathcal{P}$, we write $\mathsf{R}_\delta(\mathcal{P})$ as the function

$$N \mapsto \max\left\{\mathsf{R}_\delta(M) \ : \ M \in \mathcal{P}, M \in \{0,1\}^{N \times N}\right\} .$$

Then the class BPP^0 is the collection of communication problems $\mathcal{P}$ which satisfy $\mathsf{R}_{1/4}(\mathcal{P}) = O(1)$.

To define communication with oracles, we require the notion of a *query set*:

▶ **Definition 8** (Query Set). *A* query set $\mathcal{Q}$ *is a set of matrices closed under (1) taking submatrices; (2) permuting rows and columns; and (3) copying rows and columns. For any set of matrices* $\mathcal{M}$*, we write* $\mathsf{QS}(\mathcal{M})$ *for the closure of* $\mathcal{M}$ *under these operations.*

Observe that if $\mathsf{R}_{1/4}(\mathcal{P}) = O(1)$ then $\mathsf{R}_{1/4}(\mathsf{QS}(\mathcal{P})) = O(1)$, since constant-cost protocols are preserved by row and column copying as well as taking submatrices.

▶ **Definition 9** (Communication with oracles). *Let $\mathcal{P}$ be any communication problem, i.e. set of Boolean matrices. For any $N \times N$ matrix M with values in a set Λ, write $\mathsf{D}^{\mathcal{P}}(M)$ for the minimum cost of a two-way deterministic protocol computing M as follows. The protocol is a binary tree T where each leaf node v is assigned a value $\ell(v) \in \Lambda$, and each inner node v is assigned a query matrix $Q \in \{0,1\}^{N \times N}$ where $Q \in \mathsf{QS}(\mathcal{P})$. On any pair of inputs $(i,j) \in [N] \times [N]$, the protocol proceeds as follows: the current pointer c is initiated as the root of T, and at every step, if $Q_c(i,j) = 1$ then the pointer c moves to its left child, and otherwise if $Q_c(i,j) = 0$ then the pointer c moves to the right. Once the pointer c reaches a leaf, the output of the protocol is the value $\ell(c)$ assigned to the leaf c. It is required that $\ell(c) = M(i,j)$. The cost of the protocol is the depth of T.*

This definition differs from the standard definition of oracle communication because we do not restrict the input size of the oracle. Specifically, each oracle query is represented by an $N \times N$ matrix $Q \in \mathsf{QS}(\mathcal{P})$, obtained by taking a submatrix of an *arbitrarily large* instance of $P \in \mathcal{P}$ and then copying rows and columns. This is the natural definition because this preserves constant-cost randomized protocols, whereas preserving non-constant cost functions usually requires restricting the size of the instance $P \in \mathcal{P}$.

▶ Remark 10. For constant-cost communication problems, i.e. problems $\mathcal{P} \in \mathsf{BPP}^0$, we will simply identify the problem $\mathcal{P}$ with its query set $\mathsf{QS}(\mathcal{P})$ since this does not change the communication complexity of $\mathcal{P}$. For example, $\mathsf{D}^{\mathrm{EQ}}(\cdot)$ is $\mathsf{D}^{\mathcal{Q}}(\cdot)$ where $\mathcal{Q}$ is taken to be the closure $\mathsf{QS}(\{I_{N,N}\})$ of the identity matrices.

3 Better Boosting of Equality Protocols

We prove the first part of Theorem 4, that Equation (1) can be improved for the EQUALITY oracle. This theorem will also be applied in the later sections of the paper.

The proof uses the "noisy-search-tree" argument of [10]. This is a well-known idea that was previously applied in [21] to get an upper bound on the communication complexity of GREATER-THAN; see also the textbook exercise in [23]. We only require the observation that the argument works for arbitrary EQUALITY queries, not just the binary search queries used in those papers. Also, we did not find any complete exposition of the proof of the GREATER-THAN upper-bound: the application of [10] in [21] is black-box and informal, and the models of computation in these two works do not match up, which causes some very minor gaps in the proof[1], so we make an effort to give a complete exposition here.

Informal protocol sketch. The idea of the protocol is that an EQUALITY-oracle protocol is a binary tree T, where each node is a query to the oracle. On any given input, there is one "correct" path through T. The randomized protocol keeps track of a current node c in the tree T. In each round, the node c either moves down to one of its children, or, if it detects that a mistake has been made in an earlier round, it moves back up the tree. There are two main ideas:

[1] The gap is that the outputs of the EQUALITY subroutine are *not* independent random variables. As far as we can tell, this very minor issue persists in the textbook exercise in [23] devoted to the GREATER-THAN problem.

1. At every node c, the protocol can "double-check" the answers in *all* ancestor nodes with only $O(1)$ communication overhead, which *implicitly* reduces the error of all previous queries. This uses a special property of the Equality oracle, that a conjunction of equalities $(a_1 = b_1) \wedge (a_2 = b_2) \wedge \cdots \wedge (a_t = b_t)$ is equivalent to a single equality $(a_1, a_2, \ldots, a_t) = (b_1, b_2, \ldots, b_t)$. We can use this property to check if the current node c is on the "correct" path. (This simple observation is our contribution to this argument.)
2. The random walk of the node c through the tree is likely to stay close to the "correct" path; this is essentially the argument of [10].

▶ **Theorem 11.** *For any $M \in \Lambda^{N \times N}$ with values in an arbitrary set Λ,*

$$\mathsf{R}_\delta(M) = O\left(\mathsf{D}^{\mathrm{EQ}}(M) + \log\frac{1}{\delta}\right).$$

Proof. Let T be the tree of depth $d = \mathsf{D}^{\mathrm{EQ}}(M)$ as in Definition 9. For a node v in T let $a_v, b_v \colon [N] \to \mathbb{N}$ be the functions defining the oracle query at the node v with $Q_v(i,j) = \textsc{Eq}(a_v(i), b_v(j))$. Let $R := 4 \cdot \max\{d, C\log(1/\delta)\}$ where C is a sufficiently large constant, and construct a tree T' by replacing each leaf node v of T with another tree L_v of depth $C\log(1/\delta)$ (where C is a sufficiently large constant), with each node v' of L_v being a copy of the parent node of v in T (i.e. the functions $a_{v'}, b_{v'} \colon [N] \to \mathbb{N}$ are identical to those of the parent of v). We then simulate the protocol defined by T using Algorithm 1.

Algorithm 1 Noisy-Tree Protocol.

```
Input: Row $i$, column $j$ of communication matrix $M$.
Initialize pointer $c \leftarrow \mathrm{root}(T')$.
for $r \in [R]$ do
    Let $P = (p_1, p_2, \ldots, p_k)$ be the path in $T'$ from $\mathrm{root}(T')$ to $c$.
    Let $(q_1, q_2, \ldots, q_t)$ be the subsequence of $P$ where the protocol has taken the left branch.
        ▷ (i.e. the nodes where the protocol previously detected "equality".)
    Use the Equality protocol with error probability $1/4$ to check
        $(a_{q_1}(i), a_{q_2}(i), \ldots, a_{q_t}(i)) = (b_{q_1}(j), b_{q_2}(j), \ldots, b_{q_t}(j))$?
        ▷ Re-check all previous "equality" answers simultaneously.
    if inequality is detected on the sequence $q_1, \ldots, q_t$ then
        ▷ A mistake was detected in an earlier round; go back up.
        Update $c$ to be the parent of $c$ in $T'$.
    else
        ▷ Check the current node and continue.
        Use the Equality protocol with error probability $1/4$ to check $a_c(i) = b_c(j)$?
        if Equality is detected then move $c$ to its left child, otherwise move $c$ to its right child.
if $c$ belongs to a subtree $L_v$ (replacing leaf $v$ of $T$) then
    return $\ell(v)$. Otherwise return 0.
```

Since Algorithm 1 performs R rounds using in each round at most 2 instances of the randomized Equality protocol with error $1/4$, the total amount of communication is at most $O(R) = O(\max\{d, \log(1/\delta)\})$ as desired. Let us now verify correctness.

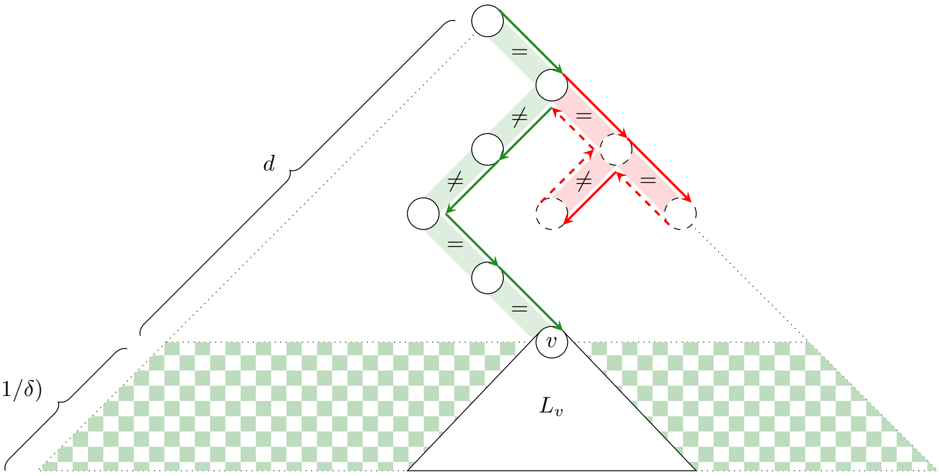


Figure 1 The picture represents the runtime of Algorithm 1. The thick green path is $P'_{i,j}$ for some i and j. The walk corresponding to the runtime of Algorithm 1 is represented with thin arrows: green arrows represent *good* rounds, solid red arrows represent bad rounds where protocol makes a mistake, and dashed red arrows represent bad rounds where protocol backtracks.

For any inputs i, j there is a unique root-to-leaf path $P_{i,j}$ taken in T ending at some leaf v, and a corresponding unique path $P'_{i,j}$ in T' which terminates at the subtree L_v. For any execution of Algorithm 1, we say a round r is "good" if the pointer c starts the round on a vertex in $P'_{i,j} \cup L_v$ and also ends the round on a vertex in $P'_{i,j} \cup L_v$. We say round r is "bad" otherwise. Write $\boldsymbol{g}$ for the number of good rounds and $\boldsymbol{b}$ for the number of bad rounds, which are random variables satisfying $R = \boldsymbol{b} + \boldsymbol{g}$.

▷ Claim 12. If $\boldsymbol{g} > d$ then the protocol produces a correct output.

Proof of claim. Observe that, if $c \in P'_{i,j}$ at the start of round r, then the counter cannot move back up, because the Equality protocol has one-sided error and will correctly report that the concatenated strings are equal with probability 1. So the protocol must have terminated with the counter c at a descendent of the g^{th} node of $P'_{i,j}$. Since $\boldsymbol{g} > d$, the protocol terminated with c in the subtree of T' that replaced the final node v of $P_{i,j}$, meaning that it will output the correct value. ◁

We say that the protocol *makes a mistake* in round r if the randomized Equality protocol erroneously outputs "equal" in Line 5 when $(a_{q_1}(i), \dots, a_{q_t}(i)) \neq (b_{q_1}(j), \dots, b_{q_t}(j))$, or if these tuples are truly equal but the protocol erroneously reports "equal" in Line 9 when $a_c(i) \neq b_c(j)$. Define the random variable $\boldsymbol{m}_r := 1$ if the protocol makes a mistake in round r and 0 otherwise, and define $\boldsymbol{m} = \sum_{r=1}^{R} \boldsymbol{m}_r$ for the total number of rounds where the protocol makes a mistake.

▷ Claim 13. $\boldsymbol{b} \leq 2\boldsymbol{m}$.

Proof of claim. Consider any bad round r. Either the counter c moves up or down the tree. If the counter c moves up to its parent c', then we charge the bad round to the most recent round $r' < r$ where the counter started at c' and observe that the protocol must have made a mistake at round r'. Otherwise, if the counter c moves down the tree, we charge the bad round to r itself and observe that the protocol makes a mistake in round r. Then we see that each round where a mistake is made is charged for at most 2 bad rounds (one for itself, if the counter moves down; and one for the earliest round where the counter returns to its current position). ◁

If the protocol outputs the incorrect value then we must have $\boldsymbol{g} = R - \boldsymbol{b} \leq d$ and therefore $R - d \leq \boldsymbol{b} \leq 2\boldsymbol{m}$, so $\boldsymbol{m} \geq \frac{R-d}{2}$. It remains to bound the number of mistakes $\boldsymbol{m}$; we write $\boldsymbol{m} = \sum_{r=1}^{R} \boldsymbol{m}_r$ where $\boldsymbol{m}_r$ indicates whether the protocol makes a mistake in round r.

In any round r, conditional an all previous rounds, the probability that the protocol makes a mistake is at most $1/4$: either there is an ancestor node in P where a mistake was made in an earlier round, in which case a mistake is made in round r only if it makes an error in Line 5; or the path P is entirely correct and the protocol makes a mistake only if there is an error in Line 9. So $\mathbb{P}[\boldsymbol{m}_r = 1 \mid \boldsymbol{m}_1, \ldots, \boldsymbol{m}_{r-1}] \leq 1/4$ for every r and $\mu := \mathbb{E}[\boldsymbol{m}] \leq R/4$. Using known concentration bounds (e.g. Theorem 3.1 of [17]), for any $\frac{1}{4} \leq \gamma \leq 1$ we have $\mathbb{P}[\boldsymbol{m} \geq \gamma R] \leq e^{-R \cdot D(\gamma \| \delta)}$; in particular, since $R = 4 \cdot \max\{d, C \log(1/\delta)\}$, we have $\frac{R-d}{2} \geq \frac{3R}{8}$, so for constant $\kappa := D\left(\frac{3}{8} \middle\| \frac{1}{4}\right) > 0$,

$$\mathbb{P}\left[\boldsymbol{m} \geq \frac{R-d}{2}\right] \leq \mathbb{P}\left[\boldsymbol{m} \geq \frac{3}{8} \cdot R\right] \leq e^{-R \cdot \kappa} \leq e^{-4C \log(1/\delta) \cdot \kappa} \leq \delta \,,$$

when we choose C to be a sufficiently large constant. ◀

4 No Better Boosting for Hamming Distance, and Consequences

We now complete the proof of Theorem 4 by showing that Equation (1) is nearly tight for the 1-Hamming Distance oracle. We prove this with a direct-sum result, showing that computing k independent copies of 1-Hamming Distance cannot be computed without the $\log k$-factor loss from boosting. Let us define the direct sum problems.

For any function $f \colon X \times Y \to Z$ and any $k \in \mathbb{N}$, we define function $f^{\otimes k}$ as the function which computes k copies of f, i.e. $f^{\otimes k} \colon X^k \times Y^k \to Z^k$ where on inputs $x \in X^k$ and $y \in Y^k$,

$$f^{\otimes k}(x, y) = (f(x_1, y_1), f(x_2, y_2), \ldots, f(x_k, y_k)) \,.$$

It is easy to see that $\mathsf{D}^{\mathrm{HD}_1}((\mathrm{HD}_1^n)^{\otimes k}) = k$ for $n > 1$ since we can compute each copy of HD_1^n with one query. In this section we prove:

▶ **Theorem 14.** *For all $n \geq 4k^2$, $\mathsf{R}_{1/4}((\mathrm{HD}_1^n)^{\otimes k}) = \Omega(k \log k / \log\log k)$. Consequently, there exist matrices M such that*

$$\mathsf{R}_{1/4}(M) = \Omega\left(\frac{\mathsf{D}^{\mathrm{HD}_1}(M) \cdot \log \mathsf{D}^{\mathrm{HD}_1}(M)}{\log\log \mathsf{D}^{\mathrm{HD}_1}(M)}\right) .$$

Our proof has two further consequences. The first is about randomized parity decision trees (see e.g. [4] for definitions and background on parity decision trees): it is not hard to see that the randomized parity decision tree complexity of the 1-Hamming Weight function $\mathrm{HW}_1^n \colon \{0,1\}^n \to \{0,1\}$ defined by $\mathrm{HW}_1^n(x) = 1$ iff $|x| = 1$ is $\mathsf{RPDT}(\mathrm{HW}_1^n) = O(1)$. Since $\mathrm{HD}_1^n(x, y) = \mathrm{HW}_1^n(x \oplus y)$ and one can simulate each parity query with two bits of communication, we get $\mathsf{R}_{1/4}((\mathrm{HD}_1^n)^{\otimes k}) = O(\mathsf{RPDT}_{1/4}((\mathrm{HW}_1^n)^{\otimes k}))$. Together these statements imply:

▶ **Corollary 15.** *For $n \geq 4k^2$, $\mathsf{RPDT}((\mathrm{HW}_1^n)^{\otimes k}) = \Omega(k \log k / \log\log k)$.*

The second consequence of our proof, explained in Section 4.3, is the optimal $\Omega(\log n)$ lower bound on the number of Equality queries required to compute HD_1^n. All of these results come from our main lemma, a randomized reduction from HD_k^n to $(\mathrm{HD}_1^n)^{\otimes O(k)}$.

4.1 Randomized Reduction Lemma

▶ **Lemma 16.** *For $c = 9/10$, and for all $k \in \mathbb{N}$, let $R = \log_{1/c} k$ and $\delta := \frac{1}{10R}$. Then*

$$\mathsf{R}_{1/4}(\mathrm{HD}_k^n) = O\left(\sum_{i=0}^{R} \mathsf{R}_\delta((\mathrm{HD}_1^n)^{\otimes(4k \cdot c^i)})\right).$$

Proof. Our protocol for $\mathrm{HD}_k^n(x,y)$ is Algorithm 2. Let $c = 9/10$ and let C be some constant to be determined later. For a string $x \in \{0,1\}^n$ and a set $S \subseteq [n]$, we will write $x_S \in \{0,1\}^{|S|}$ for the substring of x on coordinates S.

Algorithm 2 Hamming Distance Reduction.

```
Input: $x, y \in \{0,1\}^n$.
 1: Initialize $T \leftarrow [n]$; $\ell \leftarrow k$.
 2: while $\ell > C$ do
 3:     Let $S_1, \dots, S_{4\ell}$ be a uniformly random partition of $T$.
 4:     Let $u_i = x_{S_i}$; $v_i = y_{S_i}$ be the substrings of $x, y$ on subsets $S_i$ for all $i \in [4\ell]$.
 5:     Run $\delta$-error protocols for $(\mathrm{HD}_1^n)^{\otimes 4\ell}((u_1,v_1),\dots,(u_{4\ell},v_{4\ell}))$ and
        $\mathrm{EQ}_n^{\otimes 4\ell}((u_1,v_1),\dots,(u_{4\ell},v_{4\ell}))$.
        ▷ Assuming these subroutines are correct, we know $\mathsf{dist}(u_i,v_i)$ exactly, if $\mathsf{dist}(u_i,v_i) \in \{0,1\}$.
 6:     $w_i \leftarrow \mathsf{dist}(u_i,v_i)$ if $\mathsf{dist}(u_i,v_i) \le 1$ and 2 otherwise.
        ▷ We can safely output 0 if we see more than $\ell$ differences:
 7:     if $\sum_{i \in [4\ell]} w_i > \ell$ then return 0.
        ▷ In the next step, isolate the sets $S_i$ where the protocol finds exactly one difference.
 8:     $s \leftarrow |\{i \in [4\ell] \mid w_i = 1\}|$.
        ▷ If $\mathsf{dist}(x_T,y_T) > \ell$, we should see many sets with exactly one difference; output 1 otherwise:
 9:     if $s < \ell/10$ then return 1.
        ▷ Throw out sets $S_i$ with at most one difference; update the number $\ell$ of remaining differences.
10:     $T \leftarrow \bigcup_{i \in [4\ell]:\, w_i = 2} S_i$.
11:     $\ell \leftarrow \ell - s$.
12: return $\mathrm{HD}_\ell^{|T|}(x_T, y_T)$.
```

First, let us calculate the cost of the protocol. As guaranteed by Line 9, at each iteration the value of ℓ is reduced to at most $\frac{9}{10}\ell = c\ell$, so there are at most $R = \log_{1/c} k$ iterations, and in the i-th iteration (indexed from zero), $\ell \le kc^i$. Hence, at each iteration, the communication cost is at most

$$\mathsf{R}_\delta((\mathrm{HD}_1^n)^{\otimes 4kc^i}) + \mathsf{R}_\delta((\mathrm{EQ})^{\otimes 4kc^i}) \le 2 \cdot \mathsf{R}_\delta((\mathrm{HD}_1^n)^{\otimes 4kc^i}).$$

Since C is a constant, the cost of the final step with $\ell \le C$ is $O(1)$.

Now let us estimate the error. Since there are at most $R = \log_{1/c} k$ iterations and the 2 protocols in Line 5 each have error at most $\delta = 1/10R$, the total probability of an error occurring in Line 5 is at most $1/5$. We may therefore assume from now on the perfect correctness of the values w_i.

Under this assumption, the protocol maintains the invariant that the number of bits outside T where x, y differ is $\mathsf{dist}(x_{[n]\setminus T}, y_{[n]\setminus T}) = k - \ell$, so it cannot output the incorrect value in Line 7. Let us consider the probability that the protocol outputs the incorrect value in Line 9. This only occurs if $\mathsf{dist}(x_T, y_T) > \ell$ and $s < \ell/10$. We need to estimate $\mathbb{P}[|\{i \in [4\ell] \mid w_i = 1\}| \geq \ell/10]$. The size of the set $\{i \in [4\ell] \mid w_i > 0\}$ is the number of unique colors we get when coloring each element i of the set $\Delta_T := \{i \in T : x_i \neq y_i\}$ of cardinality $|\Delta_T| = \mathsf{dist}(x_T, y_T)$ uniformly with color $\boldsymbol{\chi}_i \sim [4\ell]$; call this number $\boldsymbol{\chi} := |\{\boldsymbol{\chi}_i : i \in \Delta_T\}|$. We know that Line 9 does not halt, so $|\{i \in [4\ell] \mid w_i = 2\}| < \ell/2$. Then, if $\boldsymbol{\chi} \geq (6/10)\ell$, it must be that $|\{i \in [4\ell] : w_i = 1\}| \geq \boldsymbol{\chi} - \ell/2 \geq \ell/10$, so Line 9 does not halt. For simplicity, since $|\Delta_T| \geq \ell$, in the next expression we consider only the first ℓ elements of Δ_T and identify them with the set $[\ell]$. The probability we need to estimate is

$$\begin{aligned}\mathbb{P}[|\{\boldsymbol{\chi}_i \mid i \in [\ell]\}| \leq 0.6\ell] &\leq \sum_{S \in \binom{[\ell]}{0.6\ell}} \mathbb{P}[\{\boldsymbol{\chi}_i \mid i \in [\ell]\} \subseteq \{\boldsymbol{\chi}_i \mid i \in S\}] \\ &\leq \binom{\ell}{0.6\ell} \cdot \left(\frac{6}{10 \cdot 4}\right)^{0.4\ell} < 2^{\ell} \cdot 2^{\log_2(3/20)\cdot 0.4\ell} \leq 2^{-.01\ell}.\end{aligned}$$

We have that the total error is bounded by $\sum_{\ell=C}^{\infty} 2^{-.01\ell} \leq 2^{-.01C}/(1-2^{-.01}) \leq 100 \cdot 2^{-.01C}$, so choosing C to be large enough we get arbitrarily small constant error. ◀

4.2 Direct Sum Theorem for 1-Hamming Distance

We require the lower bound on the communication cost of HD_k^n:

▶ **Theorem 17** ([24]). *For all $k^2 \leq \delta n$, $\mathsf{R}_\delta(\mathrm{HD}_k^n) = \Omega(k \log(k/\delta))$.*

Now we can prove Theorem 14.

Proof of Theorem 14. Assume for contradiction that $\mathsf{R}_{1/4}((\mathrm{HD}_1^n)^{\otimes k}) = o(k \log k / \log\log k)$, so by standard boosting,

$$\mathsf{R}_\delta((\mathrm{HD}_1^n)^{\otimes k}) = o\left(\frac{k \log k}{\log\log k} \cdot \log \frac{1}{\delta}\right).$$

Then by Lemma 16, with $c = 9/10$ and $R = \log_{1/c} k$,

$$\begin{aligned}\mathsf{R}_{1/4}(\mathrm{HD}_k^n) = O\left(\sum_{i=0}^{R} \mathsf{R}_\delta((\mathrm{HD}_1^n)^{4kc^i})\right) &= \sum_{i=0}^{R} o\left(\frac{kc^i \log(kc^i)}{\log\log(kc^i)} \log\log k\right) \\ &= \sum_{i=0}^{R} o(c^i k \log k) = o(k \log k),\end{aligned}$$

which contradicts Theorem 17 when $n \geq 4k^2$. ◀

Our Corollary 15 for randomized parity decision trees follows easily from this theorem since a randomized parity decision tree for 1-Hamming Weight, (or k copies of it), can be simulated by a randomized communication protocol to compute 1-Hamming Distance (or k copies of it).

4.3 Lower Bound on Computing 1-Hamming Distance with Equality Queries

Recently, [12, 13] showed that Equality is not complete for the class BPP^0 of constant-cost communication problems, and [8] showed that there is *no* complete problem for this class. The independent and concurrent proofs of [12, 13] both showed that $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = \omega(1)$.

We showed above that functions which reduce to Equality have better boosting, while 1-Hamming Distance does not, so 1-Hamming Distance cannot reduce to Equality – this gives a new and unexpected proof that Equality is not complete for BPP^0:

▶ **Corollary 18.** $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = \omega(1)$*. Therefore,* Equality *is not a complete problem for* BPP^0*.*

There is an easy upper bound of $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = O(\log n)$ obtained using binary search. With a more careful argument we can strengthen the above result and get a new proof that this is optimal, matching the lower bound already given in [12] by Fourier analysis.

▶ **Theorem 19.** $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = \Theta(\log n)$*.*

Proof. Assume for the sake of contradiction that $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = o(\log n)$, which immediately implies $\mathsf{D}^{\mathrm{EQ}}((\mathrm{HD}_1^n)^{\otimes k}) = o(k \log n)$. By Theorem 11 we then have $\mathsf{R}_\delta((\mathrm{HD}_1^n)^{\otimes k}) \le o(k\log n) + O(\log 1/\delta)$. Applying Lemma 16 we get, for $c = 9/10$ and $\delta = \frac{1}{10 \log_{1/c} k}$,

$$\begin{aligned}\mathsf{R}_{1/4}(\mathrm{HD}_k^n) &= O\left(\sum_{i=0}^{\log_{1/c} k} \mathsf{R}_\delta((\mathrm{HD}_1^n)^{\otimes 4kc^i})\right) \\ &= \sum_{i=0}^{\log_{1/c} k} (o(kc^i \log n) + O(\log\log k)) = o(k\log n) + O(\log k \log\log k).\end{aligned}$$

Applying this inequality with $n = k^4$ we get $\mathsf{R}_{1/4}(\mathrm{HD}_k^{k^4}) = o(k \log k)$, which contradicts Theorem 17. ◀

▶ Remark 20. It is interesting that the additive $O(\log(1/\delta))$ in Theorem 11 is required for this proof. If the $\log(1/\delta)$ term was multiplicative, we would get a bound of $o(k \log n \cdot \log\log k)$ in the sum, giving $o(k \log k \log\log k)$ when we set $n = k^4$, which is not in contradiction with Theorem 17. So the weaker (but still non-trivial) bound $\mathsf{R}_{1/4}(M) = O(\mathsf{D}^{\mathrm{EQ}}(M))$ would not suffice, although it would still allow us to conclude $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = \omega(1)$. The trivial bound of $\mathsf{R}_{1/4}(M) = O(\mathsf{D}^{\mathrm{EQ}}(M) \log \mathsf{D}^{\mathrm{EQ}}(M))$ would not allow us to prove even $\mathsf{D}^{\mathrm{EQ}}(\mathrm{HD}_1^n) = \omega(1)$.

5 Noisy-Tree Fails for Other Oracles

At this point we cannot determine whether better boosting is possible *only* for the constant-cost protocols which reduce to Equality. But we can make some progress towards this question by observing that the "noisy-tree" protocol in Theorem 11 does not work for any other oracles in BPP^0. To state this formally, we must define a reasonable generalization of that protocol.

The noisy-tree protocol relied on two properties of the Equality oracle. The first is that it has one-sided error (the protocol for Equality will output the correct answer with probability 1 when the inputs are equal). The second property is what we will call the *conjunction property*:

A query set $\mathcal{Q}$ has the *conjunction property* if there exists a constant c such that for all $d \in \mathbb{N}$ and all $Q_1, \dots, Q_d \in \mathcal{Q}$, $\mathsf{D}^{\mathcal{Q}}\left(\bigwedge_{i=1}^d Q_i\right) \le c$ where $\bigwedge_{i=1}^d Q_i$ denotes the problem of computing

$$Q_1(x_1, y_1) \wedge Q_2(x_2, y_2) \wedge \cdots \wedge Q_d(x_d, y_d)\,.$$

on d pairs of inputs $(x_1, y_1), \dots, (x_d, y_d)$. For example, Equality has the conjunction property because computing

$$\mathrm{EQ}(x_1, y_1) \wedge \mathrm{EQ}(x_2, y_2) \wedge \cdots \wedge \mathrm{EQ}(x_d, y_d)$$

can be done with the single query $\text{Eq}\left(((x_1, x_2, \ldots, x_d), (y_1, y_2, \ldots, y_d)\right)$. Following the proof of Theorem 11, we could claim the following result, which would hold even for oracles $\mathcal{Q}$ that have non-constant cost (but still using arbitrary-size oracle queries[2]):

▶ **"Theorem".** Let $\mathcal{Q}$ be any query set satisfying the conjunction property, and whose elements $Q \in \mathcal{Q}$ admit one-sided error randomized communication protocols with cost $O(\mathsf{R}(\mathcal{Q}))$. Then for any $M \in \{0,1\}^{N \times N}$, $\mathsf{R}_\delta(M) = O(\mathsf{D}^{\mathcal{Q}}(M) \cdot \mathsf{R}_{1/4}(\mathcal{Q}) + \log \frac{1}{\delta})$.

But it turns out that this does not really generalize Theorem 11, even if we require only the conjunction property (i.e. ignore one-sided error):

▶ **Proposition 21.** *If $\mathcal{Q}$ is a query set that satisfies the conjunction property, then it is either a subset of the query set of* Equality*, or it is the set of all matrices.*

To prove this, we use VC dimension. The VC dimension of a Boolean matrix M is the largest d such that there are d columns of M, where the submatrix of M restricted to those columns contains all 2^d possible distinct rows. A family $\mathcal{F}$ of matrices has *bounded VC dimension* if there is a constant d such that all $M \in \mathcal{F}$ have VC dimension at most d. If $\mathcal{F}$ is closed under taking submatrices (and permutations), then it has bounded VC dimension if and only if it is not the family of all matrices.

Proof of Proposition 21. If $\mathcal{Q}$ does not contain the matrix $\left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, then it is not hard to see that $\mathcal{Q}$ is a subset of the query set of Equality. So we suppose that $\mathcal{Q}$ contains the matrix $\left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and satisfies the conjunction property. We first show:

▷ Claim 22. Every matrix $M \in \{0,1\}^{N \times N}$ is a submatrix of $\bigwedge_{i=1}^N Q_i$ where each $Q_i = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

Proof of claim. Let each Q_i be a copy of $\left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right]$, so that $Q = \bigwedge_{i=1}^N Q_i$ has row space $[2]^N$ and column space $[2]^N$. Let $M \in \{0,1\}^{N \times N}$ and map each row $x \in [N]$ of M to the row $v(x) \in [2]^N$ of Q with

$$\forall j \in [N] : v(x)_j = \begin{cases} 1 & \text{if } M(x,j) = 1 \\ 2 & \text{if } M(x,j) = 0\,, \end{cases}$$

and map each column $y \in [N]$ of M to the column $w(y) \in [2]^N$ of Q with

$$\forall j \in [N] : w(y)_j = \begin{cases} 1 & \text{if } j \neq y \\ 2 & \text{if } j = y\,. \end{cases}$$

For any row x and column y of M, if $M(x,y) = 1$ then

$$Q_i(v(x)_j, w(y)_j) = \begin{cases} Q_i(1,1) = 1 & \text{if } M(x,j) = 1 \text{ and } j \neq y \\ Q_i(1,2) = 1 & \text{if } M(x,j) = 1 \text{ and } j = y \\ Q_i(2,1) = 1 & \text{if } M(x,j) = 0 \text{ and } j \neq y\,. \end{cases}$$

This covers all the cases, since we never have $M(x,j) = 0$ and $j = y$, so $Q(v(x), w(y)) = 1$. Finally, if $M(x,y) = 0$ then

$$Q_i(v(x)_y, w(y)_y) = Q_i(2,2) = 0\,,$$

so $Q(v(x), w(y)) = 0$. Therefore M is a submatrix of Q. ◁

[2] Arbitrary-size oracle queries may be sensible for non-constant cost problems that still have bounded VC dimension, e.g. Greater-Than oracles as in [5].

By the conjunction property, there is a constant c such that $\mathsf{D}^{\mathcal{Q}}(M) \leq \mathsf{D}^{\mathcal{Q}}(\bigwedge_{i=1}^{N} Q_i) \leq c$ for all $M \in \{0,1\}^{N \times N}$. Therefore, there is a constant C and a function $f \colon \{0,1\}^C \to \{0,1\}$ such that all matrices M can be written as

$$M(x,y) := f(Q_1(x,y), Q_2(x,y), \dots, Q_C(x,y))$$

where each $Q_i \in \mathcal{Q}$ (think of f as the function which simulates the protocol for $\mathsf{D}^{\mathcal{Q}}(M)$ using the answers to each query Q_i; see e.g. [14] for the simple proof of this fact). Let $f(\mathcal{Q})$ denote the set of all matrices which can be achieved in this way, which we have argued is the set of all matrices. For the sake of contradiction, assume that $\mathcal{Q}$ is not the set of all matrices, so that the VC dimension $\mathsf{VC}(\mathcal{Q})$ is bounded. Then standard VC dimension arguments (see e.g. [20]) show that the VC dimension of $f(\mathcal{Q})$ is at most $O(\mathsf{VC}(\mathcal{Q}) \cdot C \log C)$. Since C is constant, the VC dimension of $f(\mathcal{Q})$ is therefore also bounded, but $f(\mathcal{Q})$ contains all matrices, so this is a contradiction and $\mathcal{Q}$ must contain all matrices. ◀

▶ Remark 23. If one is interested only in constant-cost oracles, we may replace the conjunction property $\mathsf{D}^{\mathcal{Q}}(\bigwedge_i Q_i) \leq c$ with the property $\mathsf{R}_{1/4}\left(\bigwedge_{i=1}^{d} Q_i\right) = O(1)$, but the same proof rules out this generalization as well.

References

1 Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. *SIAM Journal on Computing*, 42(3):1327–1363, 2013. doi:10.1137/100811969.

2 Shalev Ben-David, Mika Göös, Robin Kothari, and Thomas Watson. When is amplification necessary for composition in randomized query complexity? In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2020.

3 Eric Blais and Joshua Brody. Optimal separation and strong direct sum for randomized query complexity. In *34th Computational Complexity Conference (CCC 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019.

4 Arkadev Chattopadhyay, Ankit Garg, and Suhail Sherif. Towards stronger counterexamples to the log-approximate-rank conjecture. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2021.

5 Arkadev Chattopadhyay, Shachar Lovett, and Marc Vinyals. Equality alone does not simulate randomness. In *34th Computational Complexity Conference (CCC 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

6 Louis Esperet, Nathaniel Harms, and Andrey Kupavskii. Sketching distances in monotone graph classes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

7 Yuting Fang, Mika Göös, Nathaniel Harms, and Pooya Hatami. Constant-cost communication does not reduce to k-hamming distance, 2024. arXiv:2407.20204.

8 Yuting Fang, Lianna Hambardzumyan, Nathaniel Harms, and Pooya Hatami. No complete problem for constant-cost randomized communication. In *Proceedings of the Symposium on Theory of Computing (STOC 2024)*, 2024.

9 Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM Journal on computing*, 24(4):736–750, 1995. doi:10.1137/S0097539792235864.

10 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

11 Lianna Hambardzumyan, Hamed Hatami, and Pooya Hatami. A counter-example to the probabilistic universal graph conjecture via randomized communication complexity. *Discrete Applied Mathematics*, 322:117–122, 2022. doi:10.1016/J.DAM.2022.07.023.

12 Lianna Hambardzumyan, Hamed Hatami, and Pooya Hatami. Dimension-free bounds and structural results in communication complexity. *Israel Journal of Mathematics*, 253(2):555–616, 2022.

13 Nathaniel Harms, Sebastian Wild, and Viktor Zamaraev. Randomized communication and implicit graph representations. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1220–1233, 2022. doi:10.1145/3519935.3519978.

14 Nathaniel Harms and Viktor Zamaraev. Randomized communication and implicit representations for matrices and graphs of small sign-rank. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*. SIAM, 2024.

15 Hamed Hatami and Pooya Hatami. Guest column: Structure in communication complexity and constant-cost complexity classes. *ACM SIGACT News*, 55(1):67–93, 2024. doi:10.1145/3654780.3654788.

16 Hamed Hatami, Pooya Hatami, William Pires, Ran Tao, and Rosie Zhao. Lower bound methods for sign-rank and their limitations. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

17 Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 617–631. Springer, 2010. doi:10.1007/978-3-642-15369-3_46.

18 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5:191–204, 1995. doi:10.1007/BF01206317.

19 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1996.

20 Jiri Matousek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.

21 Noam Nisan. The communication complexity of threshold gates. *Combinatorics, Paul Erdos is Eighty*, 1:301–315, 1993.

22 Ramamohan Paturi and Janos Simon. Probabilistic communication complexity. *Journal of Computer and System Sciences*, 33(1):106–123, 1986. doi:10.1016/0022-0000(86)90046-2.

23 Anup Rao and Amir Yehudayoff. *Communication Complexity and Applications*. Cambridge University Press, 2020.

24 Mert Sağlam. Near log-convexity of measured heat in (discrete) time and consequences. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 967–978. IEEE, 2018.

Two Views on Unification: Terms as Strategies

Furio Honsell
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Marina Lenisa
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Ivan Scagnetto
Department of Mathematics, Computer Science and Physics, University of Udine, Italy

Abstract

In [19], the authors have shown that *linear application* in Geometry of Interaction (GoI) models of λ-calculus amounts to *resolution* between *principal types* of linear λ-terms. This analogy also works in the reverse direction. Indeed, an alternative definition of *unification* between algebraic terms can be given by viewing the terms to be unified as *strategies*, *i.e.* sets of pairs of occurrences of the same variable, and verifying the termination of the GoI interaction obtained by playing the two strategies. In this paper we prove that such a criterion of unification is equivalent to the standard one. It can be viewed as a local, bottom-up, definition of unification. Dually, it can be understood as the GoI interpretation of unification.

The proof requires generalizing earlier work to arbitrary algebraic constructors and allowing for multiple occurrences of the same variable in terms. In particular, we show that two terms σ and τ unify if and only if $\mathcal{R}(\sigma) \mathrel{\widehat{\subseteq}} \mathcal{R}(\tau) \mathbin{\widehat{;}} (\mathcal{R}(\sigma) \mathbin{\widehat{;}} \mathcal{R}(\tau))^*$ and $\mathcal{R}(\tau) \mathrel{\widehat{\subseteq}} \mathcal{R}(\sigma) \mathbin{\widehat{;}} (\mathcal{R}(\tau) \mathbin{\widehat{;}} \mathcal{R}(\sigma))^*$, where $\mathcal{R}(\sigma)$ denotes the set of pairs of paths leading to the same variable in the term σ, $\widehat{\subseteq}$ denotes "inclusion up to substitution" and $\widehat{;}$ denotes "composition up to substitution".

2012 ACM Subject Classification Theory of computation → Program semantics; Theory of computation → Linear logic

Keywords and phrases unification, geometry of interaction, games

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.26

Funding Work supported by project PNRR SERICS PE00000014 M4C2I1.3 – SECCO – "SecCo-OC – Secure Containers Open Call" – CUP D33C22001300002, and by project PNRR SERICS M4C2I1.3 – COVERT OpenCall In searCh Of eVidence of stEalth cybeR Threats – CUP G23C24000790006.

Acknowledgements The authors express their gratitude to the referees for their useful comments.

1 Introduction

Geometry of Interaction (GoI) is a general programme originated by J.-Y. Girard in the late 80's, which aims at giving language-independent mathematical models of algorithms. An extraordinary amount of results by dozens of logicians and computer scientists has been triggered by Girard's seminal papers [13, 14, 15] on the GoI-dynamics of cut-elimination in Linear Logic. Among these arose Game Semantics, as initiated by Abramsky and Hyland, which led to far reaching connections through notions such as traced monoidal categories. Inspired by [2, 3] and in particular by [1], in [19] the authors gave a simple explanation of *GoI-linear application* in GoI linear models of λ-calculus in terms of *unification*, more specifically *resolution*, between *principal types* of linear λ-terms (see also [10, 9, 18, 11, 17]). The notion of GoI-application was hitherto justified in terms of more complex mathematical notions, such as proof nets, C*-algebras, and categorical trace operators.

In this paper we show that the correspondence between GoI linear application and unification is intrisic. Indeed, we introduce an alternative criterion for *unification* between algebraic terms, by viewing the terms to be unified as *strategies*, *i.e.* sets of pairs of occurrences

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 26; pp. 26:1–26:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of the same variable, taken as moves, and checking at the termination of the GoI interaction between the two strategies. We prove that such a criterion of unification is equivalent to the standard one. It can be viewed as a local, bottom-up definition of unification w.r.t. the more top-down approach used in ordinary unification algorithms. But more importantly, this can be viewed, dually, as a step in Girard's GoI programme, namely as the GoI-interpretation of the unification algorithm.

The proof requires generalizing earlier work to arbitrary algebraic constructors and allowing for multiple occurrences of the same variable. In particular, we show that two terms σ and τ unify if and only if

$$\mathcal{R}(\sigma) \ \widehat{\subseteq}\ \mathcal{R}(\tau) \mathbin{\widehat{;}} (\mathcal{R}(\sigma) \mathbin{\widehat{;}} \mathcal{R}(\tau))^* \quad \text{and} \quad \mathcal{R}(\tau) \ \widehat{\subseteq}\ \mathcal{R}(\sigma) \mathbin{\widehat{;}} (\mathcal{R}(\tau) \mathbin{\widehat{;}} \mathcal{R}(\sigma))^*$$

where $\mathcal{R}(\sigma)$ denotes the set of pairs of paths leading to the same variable, *i.e.* occurrences of the same variable, in the term σ, $\widehat{\subseteq}$ denotes "inclusion up to substitution", and $\widehat{;}$ denotes "composition up to substitution".

Since we view terms as strategies, in the form of irreflexive binary relations on variable occurrences, it is easier for expository purposes to discuss first the case of terms with no constants and no *hapax variables*, *i.e.* variables which occur only once. Moreover, since variables merely play the role of placeholders within a term, it is easier to discuss first the case where the terms to be unified do not share variables. Later we remove all these restrictions, by showing that the case of general terms can be reduced to the initial restricted one.

Summing up, unification can be carried out either in a traditional *top down* fashion, taking the view of terms, as in *e.g.* [20], or *bottom up* taking the view of variable occurrences. We will call the former simply *unification*, while we will call the latter *GoI-unification.*

Related work. The use of unification and resolution to build or explain GoI models has a long history. J.-Y. Girard, in [14], first pointed out the close connection between GoI and resolution. A number of authors pursued this investigation, *e.g.* [8], and most notably M.Bagnol in his thesis [7], and subsequent papers on the more recent line of investigation of "Transcendental syntax", which has been introduced by Girard (see [16]) and further investigated in B.Eng's thesis [12]. While in these works unification and resolution are used to build GoI models, in the present paper the point of view is different. Namely, here the goal is to do the converse, *i.e.* to provide a new perspective on unification through a GoI-like mechanism.

Finally, we mention the line of research in Geometry of Interaction which has been probably the most active and fruitful in the last one or two decades, *i.e.* that of GoI based abstract machines (see *e.g.* [4, 6, 5]), which have been studied in particular in connection to time and space efficiency of the computational model of λ-calculus.

Summary. The paper is organized as follows. In Section 2 we introduce preliminary definitions. In Section 3 we discuss standard unification. In Section 4 we give the main definition of when two terms *GoI-unify*, and state the main results for terms with no constants and no hapax variables, and under the assumption that the terms to unify do not share variables. In Section 5 we give examples. In Section 6 we outline the proofs of the main results. In Section 7 we discuss the extension of GoI-unification to general terms. Final remarks and possible directions for future work appear in Section 8.

2 Preliminaries

Throughout the paper we assume the following definitions.

▶ **Definition 1** (Terms and Variable Occurrences).

(i) Terms T_Σ *are given by a* signature $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$*, where* $n > 0$*, and* Σ_i *are sets of* i*-ary constructors* f_i^j*, for* $j = 1, \dots, |\Sigma_i|$*. Terms are trees whose leaves are variables* $\alpha, \beta, \dots \in \mathit{TVar}$*, and nodes are operations in* Σ*, i.e.*

$$(T_\Sigma \ni)\ \sigma, \tau ::= \ \alpha \mid \beta \mid \ \dots \ \mid f_i^j(\sigma_1, \dots, \sigma_i).$$

(ii) *Let* T_Σ^- *be the subset of terms in* T_Σ *with no 0-ary constructors and no* hapax variables*, i.e. variables which occur only once.*

(iii) Occurrences *of variables in terms, or* occurrence terms*, are given by the following grammar:*

$$(O_\Sigma \ni)\ u[\alpha] ::= \alpha \mid (f_i^j, k)u[\alpha] \qquad\qquad \textit{for } k \leq i,$$

where

- α *denotes the occurrence of the variable* α *in the term* α*,*
- *if* $u[\alpha]$ *denotes an occurrence of* α *in* σ*, then* $(f_i^j, k)u[\alpha]$ *denotes the corresponding occurrence of* α *in* $f_i^j(\underbrace{\sigma_1, \dots, \sigma_{k-1}}_{k-1}, \sigma, \underbrace{\sigma_{k+1}, \dots, \sigma_i}_{i-k})$.

(iv) *The (possibly empty)* path *of an occurrence* $u[\alpha]$ *is* u*.*

(v) *Let* $\mathcal{R}(O_\Sigma)$ *denote the set of binary relations on* O_Σ *containing only pairs of the shape* $\langle u[\alpha], v[\alpha]\rangle$*, for* $\alpha \in \mathit{TVar}$*, and* u, v *occurrence paths.*

(vi) *Given a relation* $\mathcal{R}$*, we denote by* $\mathcal{R}^+$ *the maximal irreflexive subrelation included in the symmetric and transitive closure of* $\mathcal{R}$ *.*

▶ **Notation 2.**

- *Syntactical identity of terms is denoted by* $\equiv$*.*
- *We consider terms up to injective renaming of variables and denote term equality by* $=$*.*

▶ **Example 3.** Let f_1^1 and f_2^1 be a unary and a binary constructor, respectively, and let $\sigma = f_2^1(\alpha, f_1^1(\alpha))$. Then σ has two occurrences of the variable α, *i.e.* $(f_2^1, 1)\,\alpha$ and $(f_2^1, 2)(f_1^1, 1)\,\alpha$, with corresponding paths $u = (f_2^1, 1)$ and $v = (f_2^1, 2)(f_1^1, 1)$. ⌟

Terms give rise to *non-deterministic strategies* for a game where *moves* are variable occurrences, in the following sense:

▶ **Definition 4** (From Terms to Strategies).

(i) *A term* τ *gives rise to a set of* variable occurrences (moves)*:*

$$\mathcal{O}(\tau) = \{u[\alpha] \ \mid \ u[\alpha] \textit{ is an occurrence of } \alpha \textit{ in } \tau\}.$$

(ii) *A term* τ *gives rise to a symmetric and transitive irreflexive relation on* O_Σ (strategy)*:*

$$\mathcal{R}(\tau) = \{\langle u[\alpha], v[\alpha]\rangle \ \mid \ u[\alpha], v[\alpha] \textit{ are different occurrences of } \alpha \textit{ in } \tau\}.$$

(iii) *Two different variable occurrences* $u[\alpha], v[\beta]$ *are* compatible *if* $u = w(f_i^j, k)u'$ *and* $v = w(f_i^j, h)v'$*, for some paths* w, u', v'*, and indexes* i, j*,* $k \neq h$*.*

▶ **Example 5.** Let f_1^1 and f_2^1 be a unary and a binary constructor, respectively. Then the variable occurrences $(f_2^1, 1)(f_2^1, 1)(f_1^1, 1)\,\alpha$ and $(f_2^1, 1)(f_2^1, 2)\,\beta$ are compatible, while the occurrences $(f_2^1, 1)(f_2^1, 1)(f_1^1, 1)\,\alpha$ and $(f_2^1, 1)(f_2^1, 1)\,\beta$ are not. ⌟

For a given term τ, it is immediate that all variable occurrences in $\mathcal{O}(\tau)$ are pairwise compatible. Vice versa, the compatibility condition permits us to build a term from a set of pairwise compatible variable occurrences, by tagging some nodes of the term tree with fresh variables, if needed.

▶ **Proposition 6.**

(i) *Given a set $\mathcal{S}$ of pairwise compatible variable occurrences, we can build the tree of a term, by tagging possible missing leaves with fresh variables in $Z = \{\zeta_1, \dots, \zeta_i, \dots\}$, i.e.*

$$\mathcal{T}_Z(\mathcal{S}) = \begin{cases} \zeta & \textit{if } \mathcal{S} = \emptyset, \zeta \in Z \textit{ fresh} \\ \alpha & \textit{if } \mathcal{S} = \{\alpha\} \\ f_i^j(\mathcal{T}_Z(\{u[\alpha] \mid (f_i^j, 1)u[\alpha] \in \mathcal{S}\}), \dots, \mathcal{T}_Z(\{u[\alpha] \mid (f_i^j, i)u[\alpha] \in \mathcal{S}\})) & \textit{if } \forall v[\alpha] \in \mathcal{S}\ \exists k, v' \textit{s.t.} \\ & \qquad v = (f_i^j, k)v' \end{cases}$$

(ii) *Let σ be a term with no 0-ary constructors. Then we have $\mathcal{T}_Z(\mathcal{O}(\sigma)) = \sigma$.*

Proof. By induction on the structure of terms. The compatibility condition ensures that at any step all occurrence paths start with the same constructor f_i^j. ◀

Notice that, in Proposition 6(ii) above we consider terms without constants. In principle, one could extend this result to the whole set of terms, by suitably generalizing Definition 4 and introducing a notion of *constant occurrence*. However, we prefer to follow a different approach, which allows for a simpler theory: we first introduce and study the notion of GoI-unification for terms without constants, and then, in Section 7, we show how the general case can be dealt with via a simple encoding into the restricted case.

3 The Top-down Perspective on Unification

First we fix notations for standard notions.

▶ **Definition 7** (Terms Unifiers). *Let σ and τ be terms.*

(i) *A* substitution *is a function $U : T_\Sigma \to T_\Sigma$ defined inductively on the structure of terms from a* variable substitution *$U_{Var} : TVar \to T_\Sigma$, i.e.*

- $U(\alpha) = U_{Var}(\alpha)$
- $U(f_i^j(\sigma_1, \dots, \sigma_i)) = f_i^j(U(\sigma_1), \dots, U(\sigma_i))$.

(ii) *A* unifier *for σ and τ is a substitution U which differs from the identity on a finite number of variables and such that $U(\sigma) = U(\tau)$. We call* domain *of U, $dom(U)$, the finite set of variables on which U is not the identity.*

(iii) *Given two substitutions, U and V, the composition $U;V$ is defined as usual, i.e. $U;V = V \circ U$.*

(iv) *Given two substitutions, U and V, we define $U \leq V$ if there exists a substitution U' such that $U;U' = V$, i.e. U is more general than V.*

(v) *Given terms σ, τ, a* most general unifier (m.g.u.) *of σ, τ is a unifier $\overline{U}$ of σ and τ such that, for any unifier U of σ and τ, $\overline{U} \leq U$.*

We introduce now a unification algorithm *à la* Martelli Montanari, [20], which unifies simultaneously sets of pairs of terms.

▶ **Definition 8** (Martelli-Montanari's Algorithm). *Let E be a set of pairs of the form $\langle \sigma, \tau \rangle$, for $\sigma, \tau \in T_\Sigma$. The unification algorithm is defined by the following rules for deriving judgements of the form $\mathcal{U}(E)$:*

$$\begin{array}{lll}
\textbf{(i)} & \mathcal{U}(\{\langle f_i^j(\sigma_1,\ldots,\sigma_i), f_i^j(\tau_1,\ldots,\tau_i)\rangle\}\cup E) & \to\ \mathcal{U}(\{\langle\sigma_1,\tau_1\rangle,\ldots,\langle\sigma_i,\tau_i\rangle\}\cup E)\\
\textbf{(ii)} & \mathcal{U}(\{\langle\alpha,\alpha\rangle\}\cup E) & \to\ \mathcal{U}(E)\\
\textbf{(iii)} & \mathcal{U}(\{\langle f_i^j(\sigma_1,\ldots,\sigma_i),\alpha\rangle\}\cup E) & \to\ \mathcal{U}(\{\langle\alpha, f_i^j(\sigma_1,\ldots,\sigma_i)\rangle\}\cup E)\\
\textbf{(iv)} & \mathcal{U}(\{\langle\alpha,\sigma\rangle\}\cup E) & \to\ \mathcal{U}(\{\langle\alpha,\sigma\rangle\}\cup E[\sigma/\alpha]),\ \mathit{if}\ \alpha\notin\sigma\wedge\alpha\in Var(E)\\
\textbf{(v)} & \mathcal{U}(\{\langle\alpha,\sigma\rangle\}\cup E) & \to\ \mathit{fail},\ \mathit{if}\ \alpha\in\sigma\ \wedge\ \alpha\neq\sigma\\
\textbf{(vi)} & \mathcal{U}(\{\langle f_i^j(\sigma_1,\ldots,\sigma_i), f_k^l(\tau_1,\ldots,\tau_k)\rangle\}\cup E & \to\ \mathit{fail},\ \mathit{if}\ i\neq k\ \mathit{or}\ j\neq l
\end{array}$$

The algorithm in Definition 8 is non-deterministic. Moreover, notice that rule (iii) is not symmetric and that rule (iv) produces a set of pairs of terms to be unified, where there is only one occurrence of the variable α left.

In particular, termination of the algorithm is guaranteed by the fact that, by applying any of the rules, the complexity of the set of pairs E decreases, according to the following measure:

▶ **Definition 9.** *Let E be a set of pairs of the form $\langle\sigma,\tau\rangle$. We define a complexity measure of E by $m(E) = (m_t, m_v)$, where*

- *m_t is the sum of the complexities of terms in the lefthand parts of the pairs,*
- *m_v is the number of variables appearing in E, except variables α's for which there is only one pair $\langle\alpha,\sigma\rangle$ such that $\alpha\notin\sigma$ and $\alpha\notin E\setminus\{\langle\alpha,\sigma\rangle\}$.*

Then, the following lemma is straightforward.

▶ **Lemma 10.** *Given terms $\sigma,\tau\in T_\Sigma$, Martelli-Montanari's Algorithm terminates either with failure or with a set of pairs yielding a substitution.*

The following proposition is well-known.

▶ **Proposition 11** (Most General Unifier). *Let $\sigma,\tau\in T_\Sigma$. The terms σ and τ unify if and only if Martelli-Montanari's algorithm on $\{\langle\sigma,\tau\rangle\}$ terminates with a substitution E. Then $\{\alpha\mapsto\rho_\alpha \mid \langle\alpha,\rho_\alpha\rangle\in E\}$ is a m.g.u. of σ and τ, which is unique up to appropriate injective renaming of variables.*

4 GoI-unification: the Bottom-up Perspective of Paths

In this section we provide a GoI account of unification. Namely, as done in [19], we utilize the machinery underlying process interaction in *game semantics* to explain the dynamics of unification. In this perspective, terms are viewed as non-deterministic strategies over the language of moves consisting of variable occurrences (see Definition 4). The proofs of the results in this section appear in Section 6. More specifically, we provide a GoI account of unification for a restricted class of terms, *i.e.* terms with no constants and no *hapax variables*. Moreover, we assume that the two terms to unify do not share variables. In Section 7 all these restrictions are removed, namely we show that the general case can be always reduced to the special case discussed in the previous sections.

Now we need to introduce the notions of *occ-substitution* and *occ-unifier*:

▶ **Definition 12** (Occ-unifier).

(i) *An* occ-substitution *is a function $M : O_\Sigma\to O_\Sigma$ defined inductively on the structure of variable occurrences, starting from a substitution for variables $M_{Var} : TVar\to O_\Sigma$, i.e.*

- $M(\alpha) = M_{Var}(\alpha)$
- $M((f_i^j,k)u[\alpha]) = (f_i^j,k)M(u[\alpha])$.

(ii) *An* occ-unifier *of variable occurrences* $u[\alpha], v[\beta]$ *is an occ-substitution* M *such that* $M(u[\alpha]) = M(v[\beta])$.

Notice that occ-unifiers are actually *matchings*, namely two variable occurrences $u[\alpha], v[\beta]$ unify if and only if u is a prefix of v or v is a prefix of u:

▶ **Lemma 13.** *Let* $u[\alpha], v[\beta] \in O_\Sigma$*. The following are equivalent:*

(i) *there exists an occ-unifier of* $u[\alpha]$ *and* $v[\beta]$*;*

(ii) *there exists an occurrence path* w *such that* $u = vw$ *or* $v = uw$*.*

The following definition introduces a special operation on relations on variable occurrences, which will be used in the definition of GoI-unification, *i.e.* the operation of *composition up to occ-substitution*, together with the notion of *inclusion up to substitution*:

▶ **Definition 14.** *Let* $\mathcal{R}, \mathcal{R}', \mathcal{R}'' \in \mathcal{R}(O_\Sigma)$ *be binary relations on variable occurrences.*

(i) *We define* $\mathcal{R}(\sigma)\widehat{;}\mathcal{R}(\tau)$ *as* composition up to occ-substitution*, namely*

$$\langle u''[\gamma], v''[\gamma]\rangle \in \mathcal{R}(\sigma)\widehat{;}\mathcal{R}(\tau) \quad \textit{if and only if}$$

$$\exists\ \langle u[\alpha], v[\alpha]\rangle \in \mathcal{R}(\sigma),\ \langle u'[\beta], v'[\beta]\rangle \in \mathcal{R}(\tau)\ \textit{and}$$
$$\exists M\ \textit{occ-unifier of}\ v[\alpha], u'[\beta]\ \textit{s.t.}\ (M(u''[\gamma]) = M(u[\alpha])\ \wedge\ M(v''[\gamma]) = M(v'[\beta]))\ .$$

(ii) *We define the predicate* inclusion up to substitution*,* $\mathcal{R} \,\widehat{\subseteq}\, \mathcal{R}'$*, which holds if and only if for each* $\langle u[\alpha], v[\alpha]\rangle \in \mathcal{R}$ *there exists a pair* $\langle u'[\beta], v'[\beta]\rangle \in \mathcal{R}'$ *and an occ-substitution* M *such that* $M(u[\alpha]) = M(u'[\beta]) \wedge\ M(v[\alpha]) = M(v'[\beta])$*.*

One can easily check (by case analysis on the occ-substitutions) that Definition 14(i) above determines an associative operation on $\mathcal{R}(O_\Sigma)$:

▶ **Lemma 15.** *Composition up to occ-substitution of relations is an associative operation over* $\mathcal{R}(O_\Sigma)$*.*

The following is the crucial definition of the paper:

▶ **Definition 16** (GoI-unification)**.** *Let* $\sigma, \tau \in T^-_\Sigma$ *be such that* $Var(\sigma) \cap Var(\tau) = \emptyset$*. Then* σ *and* τ GoI-unify*, i.e.* $\mathcal{U}_{GoI}(\sigma, \tau)$*, if and only if*

$$\mathcal{R}(\sigma)\ \widehat{\subseteq}\ \mathcal{R}(\tau)\,\widehat{;}\,(\mathcal{R}(\sigma)\,\widehat{;}\,\mathcal{R}(\tau))^* \quad \textit{and} \quad \mathcal{R}(\tau)\ \widehat{\subseteq}\ \mathcal{R}(\sigma)\,\widehat{;}\,(\mathcal{R}(\tau)\,\widehat{;}\,\mathcal{R}(\sigma))^*\ ,$$

where, for a relation $\mathcal{R}$*,* $\mathcal{R}^*$ *denotes the relation* $Id_{O_\Sigma} \cup \bigcup_{n\geq 1} \mathcal{R}^n$*, where* Id_{O_Σ} *denotes the identity relation on* O_Σ *and* $\mathcal{R}^n$ *the n-ary composition up to substitution of* $\mathcal{R}$*.*
The "flow of control" in checking the two predicates is outlined in the following diagrams

$$\begin{tikzcd} in \arrow[r] & \bullet \arrow[r, bend left, "\mathcal{R}(\tau)"] & \bullet \arrow[l, bend left, "\mathcal{R}(\sigma)"] \arrow[r] & out \end{tikzcd} \qquad \begin{tikzcd} in \arrow[r] & \bullet \arrow[r, bend left, "\mathcal{R}(\sigma)"] & \bullet \arrow[l, bend left, "\mathcal{R}(\tau)"] \arrow[r] & out \end{tikzcd}$$

Notice the similarity of the above definition with Girard's Execution Formula, and hence the connection with game semantics.

The following is the main result of the paper, whose proof will be given in Section 6:

▶ **Theorem 17** (Equivalence)**.** *Let* $\sigma, \tau \in T^-_\Sigma$ *be such that* $Var(\sigma) \cap Var(\tau) = \emptyset$*. Then* σ *and* τ unify *if and only if they* GoI-unify*.*

$$\begin{tikzcd}
\tau : & u_1[\alpha] \arrow[r, bend left] & u_1'[\alpha] \arrow[d] & \dots & u_{n+1}[\alpha_{n+1}] \arrow[r, bend left] & u_{n+1}'[\alpha_{n+1}] \arrow[d] \\
\sigma : & u[\alpha] \arrow[u] & u_2[\alpha_2] \arrow[r, bend right] & \dots \arrow[r, bend right] & u_n'[\alpha_n] \arrow[u] & v[\alpha]
\end{tikzcd}$$

Figure 1 GoI-execution sequence.

The following lemma is very useful in verifying the criterion of Definition 16. It essentially amounts to spelling out that definition, once we observe that the order in which occ-unifications are performed in a composition chain is irrelevant. This in turn follows from Lemma 15.

▶ **Lemma 18.** *Let $\sigma, \tau \in T_\Sigma^-$ be such that $Var(\sigma) \cap Var(\tau) = \emptyset$. Then σ and τ* GoI-unify *($\mathcal{U}_{GoI}(\sigma, \tau)$) if and only if, for any $\langle u[\alpha], v[\alpha] \rangle \in \mathcal{R}(\sigma)$,*

(i) *there exist a sequence* of odd length ≥ 1 $\langle u_1[\alpha_1], u_1'[\alpha_1] \rangle, \dots, \langle u_{n+1}[\alpha_{n+1}], u_{n+1}'[\alpha_{n+1}] \rangle$, *$n \geq 0$, and*

(ii) *occ-substitutions $M_1, \dots, M_n, M_{\langle u[\alpha], v[\alpha] \rangle}$ such that*
- *$\langle u_i[\alpha_i], u_i'[\alpha_i] \rangle \in \mathcal{R}(\tau)$, for i odd;*
- *$\langle u_i[\alpha_i], u_i'[\alpha_i] \rangle \in \mathcal{R}(\sigma)$, for i even;*
- *for all $1 \leq i \leq n$, $(M_1; \dots; M_i)(u_i'[\alpha_i]) = M_i(u_{i+1}[\alpha_{i+1}])$;*
- *$M_{\langle u[\alpha], v[\alpha] \rangle}(u[\alpha]) = (M_1; \dots; M_n)(u_1[\alpha_1])$;*
- *$M_{\langle u[\alpha], v[\alpha] \rangle}(v[\alpha]) = M_n(u_{n+1}'[\alpha_{n+1}])$.*

And vice versa, for any $\langle u[\alpha], v[\alpha] \rangle \in \mathcal{R}(\tau)$ there exists a sequence of pairs of variable occurrences and occ-substitutions satisfying the properties above.
We call GoI-execution sequence *the above sequence of pairs of variable occurrences together with the corresponding occ-substitutions.*

The GoI-execution sequence mimicking the pair $\langle u[\alpha], v[\alpha] \rangle \in \mathcal{R}(\sigma)$ can be visualized with a diagram as in Figure 1, where vertical arrows denote occ-unifications (which are left implicit), while horizontal ones connect pairs in $\mathcal{R}(\sigma)$ or in $\mathcal{R}(\tau)$.

The result in Theorem 17 is far from straightforward, since the two perspectives on unification, namely the one implemented in the algorithm $\mathcal{U}$, introduced in Definition 8, and the one given by the flow of control in dealing with pairs in $\mathcal{R}(\sigma)$ and $\mathcal{R}(\tau)$ in Definition 16 are conceptually different.

In Section 8, we shall briefly outline how term unifiers can be derived from GoI-execution sequences, thereby suggesting the analogue of Proposition 11.

5 GoI-unification at Work

In this section we illustrate positive examples and failures of the unification criterion given by Definition 16. But before doing this we need to introduce some notations.

▶ **Notation 19.** *To ease intuition, when dealing with a single binary constructor f_2^1, terms will be called linear types, moreover we shall use the infix operator $\multimap$ for f_2^1, and we shall denote the paths $(f_2^1, 1)$ and $(f_2^1, 2)$ simply by l and r, respectively.*

▶ **Example 20.** Consider the two linear types $\sigma \equiv \alpha \multimap \alpha$ and $\tau \equiv (\beta \multimap \beta) \multimap (\beta \multimap \beta)$. These two terms clearly unify by taking the variable substitution $\{\alpha \mapsto \beta \multimap \beta\}$. We have $\mathcal{R}(\sigma) = \{\langle l\alpha, r\alpha \rangle\}^+$ and $\mathcal{R}(\tau) = \{\langle ll\beta, lr\beta \rangle, \langle rl\beta, rr\beta \rangle \ \langle lr\beta, rl\beta \rangle\}^+$. For each of the four pairs we provide appropriate GoI-execution sequences, where the occ-substitutions have been fully applied:

$$\begin{tikzcd}
\sigma : & l\alpha \arrow[d] & r\alpha & & l\alpha \arrow[r, bend left] & r\alpha \arrow[d] & r\alpha \arrow[r, bend left] & l\alpha \arrow[d] \\
\tau : & ll\beta \arrow[r, bend right] & rl\beta \arrow[u] & & ll\beta \arrow[u] & rl\beta \arrow[r, bend right] & rr\beta \arrow[u] & lr\beta
\end{tikzcd}$$

Figure 2 Example 20.

(i) $\langle l\alpha, r\alpha\rangle \quad \rightsquigarrow \quad ll\beta \overset{\tau}{\Leftrightarrow} rl\beta$
(ii) $\langle ll\beta, lr\beta\rangle \quad \rightsquigarrow \quad ll\beta \overset{\sigma}{\Leftrightarrow} rl\beta \overset{\tau}{\Leftrightarrow} rr\beta \overset{\sigma}{\Leftrightarrow} lr\beta$
(iii) $\langle rl\beta, rr\beta\rangle \quad \rightsquigarrow \quad rl\beta \overset{\sigma}{\Leftrightarrow} ll\beta \overset{\tau}{\Leftrightarrow} lr\beta \overset{\sigma}{\Leftrightarrow} rr\beta$
(iv) $\langle lr\beta, rl\beta\rangle \quad \rightsquigarrow \quad lr\beta \overset{\sigma}{\Leftrightarrow} rr\beta \overset{\tau}{\Leftrightarrow} ll\beta \overset{\sigma}{\Leftrightarrow} rl\beta$

We compute explicitly all the substitutions M's arising according Lemma 18 in the case (ii), *i.e.* for the pair $\langle ll\beta, lr\beta\rangle$. The sequence of pairs of variable occurrences are $\langle l\alpha, r\alpha\rangle, \langle rl\beta, ll\beta\rangle, \langle l\alpha, r\alpha\rangle$. The substitutions are:

- $M_1 : \alpha \mapsto r\beta$, which is used to match $r\alpha$ in the second component of the first pair and $rl\beta$ in the first component of the second pair;
- $M_2 : \alpha \mapsto l\beta$, which matches the second component of the second pair (after the substitution with M_1), and the first component of the third pair;
- finally we have $M_{\langle ll\beta, lr\beta\rangle} = Id$.

In Figure 2, the left-hand diagram represents the GoI sequence for the pair $\langle l\alpha, r\alpha\rangle$, while the right-hand diagram represents the GoI sequence for the pair $\langle ll\beta, lr\beta\rangle$. In the diagrams of Figure 2, vertical arrows denote occ-unifications (which are left implicit), while horizontal ones connect pairs in $\mathcal{R}(\sigma)$ or in $\mathcal{R}(\tau)$. In the present cases, the GoI-execution sequences can be directly read in the bottom lines of the two diagrams. ⌟

▶ **Example 21.** Consider the terms $\sigma \equiv f(\alpha, \alpha, \alpha)$ and $\tau \equiv f(\beta, g(\gamma, \gamma), \beta)$, where f is a ternary constructor and g is binary. We have that $\mathcal{R}(\sigma) = \{\langle (f,1)\alpha, (f,2)\alpha\rangle, \langle (f,1)\alpha, (f,3)\alpha\rangle, \langle (f,2)\alpha, (f,3)\alpha\rangle\}^+$, while $\mathcal{R}(\tau) = \{\langle (f,1)\beta, (f,3)\beta\rangle, \langle (f,2)(g,1)\gamma, (f,2)(g,2)\gamma\rangle\}^+$. These two terms clearly unify by taking the variable substitution $\{\alpha \mapsto g(\gamma,\gamma),\ \beta \mapsto g(\gamma,\gamma)\}$. We provide appropriate GoI-execution sequences for some of the pairs in the $\mathcal{R}(\ )$'s, the other pairs are dealt with symmetrically:

$\langle (f,1)\alpha, (f,3)\alpha\rangle \rightsquigarrow (f,1)\beta \overset{\tau}{\Leftrightarrow} (f,3)\beta \qquad \langle (f,1)\beta, (f,3)\beta\rangle \rightsquigarrow (f,1)\alpha \overset{\sigma}{\Leftrightarrow} (f,3)\alpha$

$\langle (f,1)\alpha, (f,2)\alpha\rangle \rightsquigarrow (f,1)(g,1)\gamma \overset{\tau}{\Leftrightarrow} (f,3)(g,1)\gamma \overset{\sigma}{\Leftrightarrow} (f,2)(g,1)\gamma \overset{\tau}{\Leftrightarrow} (f,2)(g,2)\gamma \overset{\sigma}{\Leftrightarrow} (f,1)(g,2)\gamma \overset{\tau}{\Leftrightarrow} (f,3)(g,2)\gamma \overset{\sigma}{\Leftrightarrow} (f,2)(g,2)\gamma \overset{\tau}{\Leftrightarrow} (f,2)(g,1)\gamma$

$\langle (f,2)(g,1)\gamma, (f,2)(g,2)\gamma\rangle \rightsquigarrow (f,2)(g,1)\gamma \overset{\sigma}{\Leftrightarrow} (f,3)(g,1)\gamma \overset{\tau}{\Leftrightarrow} (f,1)(g,1)\gamma \overset{\sigma}{\Leftrightarrow} (f,2)(g,1)\gamma \overset{\tau}{\Leftrightarrow} (f,2)(g,2)\gamma \overset{\sigma}{\Leftrightarrow} (f,3)(g,2)\gamma \overset{\tau}{\Leftrightarrow} (f,1)(g,2)\gamma \overset{\sigma}{\Leftrightarrow} (f,2)(g,2)\gamma$. ⌟

▶ **Example 22.** Consider the two linear types $\sigma \equiv \alpha \multimap \alpha$ and $\tau \equiv \gamma \multimap ((\beta \multimap \beta) \multimap \gamma)$. Apparently the two terms do not unify. We have $\mathcal{R}(\sigma) = \{\langle l\alpha, r\alpha\rangle\}^+$ and $\mathcal{R}(\tau) = \{\langle l\gamma, rrr\gamma\rangle, \langle rll\beta, rlr\beta\rangle\}^+$. In order to achieve an instance of the pair $\langle rll\beta, rlr\beta\rangle$ from a GoI-execution sequence both the first and last term must start with an r and be the first and second component respectively of pairs in $\mathcal{R}(\sigma)$. We show that there cannot be any GoI-sequence where both the first and the last terms start with an r. We proceed by contradiction. Assume that such a sequence exists and consider the shortest one. The other components of such pairs must then be occurrence terms starting with an l. These in turn unify with occurrence terms coming from pairs in $\mathcal{R}(\tau)$, so their other component must start with an r. Hence the GoI-execution sequence we started from was not the shortest. Contradiction. ⌟

▶ **Example 23.** Consider the two linear types $\sigma \equiv (\alpha \multimap \alpha) \to \alpha$ and $\tau \equiv (\alpha \multimap \alpha) \multimap \alpha$, where we two binary constructors $\multimap$ and $\to$ appear, and whose occurrence components are denoted respectively by l, r and L, R. We have that $\mathcal{R}(\sigma) = \{\langle Ll\alpha, Lr\alpha\rangle, \langle Ll\alpha, R\alpha\rangle, \langle lr\alpha, r\alpha\rangle\}^+$ while $\mathcal{R}(\tau) = \{\langle ll\alpha, rl\alpha\rangle, \langle ll\alpha, r\alpha\rangle, \langle lr\alpha, r\alpha\rangle\}^+$. One can readily check that no GoI execution sequence can simulate $\langle Ll\alpha, Lr\alpha\rangle$. ⌟

6 Proof of the Main Theorem

For simplicity we shall sketch the proof only for a single binary constructor $(\_ \to \_)$, which we write in infix form. We need some new notation and a number of *invariance* lemmata, whose proofs are essentially straightforward from the definitions.

▶ **Notation 24.** *Let σ be a term where all the variables occur exactly twice. When we want to highlight one or more pairs of occurrences of some of the variables, we write $\sigma[(\alpha_i, \alpha_i)]$ or $\sigma[(\alpha_1, \alpha_1), \ldots, (\alpha_n, \alpha_n)]$. Moreover we write $\sigma[(\tau_1, \rho_1)]$ when we denote the term obtained from σ by replacing the first occurrence of the variable α, which should be clear form the context, in σ with τ_1 and the second occurrence of α in σ with ρ_1, and we write $\sigma[(\tau_1, \rho_1), \ldots (\tau_n, \rho_n))]$ to denote the term obtained from σ by substituting the variables $\alpha_1, \ldots, \alpha_n$. Notice that some pairs in $(\tau_1, \rho_1), \ldots (\tau_n, \rho_n)$ could be equal.*

▶ **Lemma 25.** *Let σ, τ be terms which do not share variables, and let α be a fresh variable. Then*

$$\mathcal{U}_{GoI}(\sigma,\ \tau) \Longleftrightarrow \mathcal{U}_{GoI}(\sigma \to \tau,\ \alpha \to \alpha)\ .$$

The Lemma above holds trivially, if σ and τ do not share variables. Otherwise, the righthand side can be taken as the definition of $\mathcal{U}_{GoI}(\sigma,\ \tau)$ when the two terms share variables (see Section 7.2 below).

▶ **Lemma 26.** *Let σ be a term where all variables occur exactly twice, and let $\sigma[(\tau_1, \rho_1), \ldots, (\tau_n, \rho_n)]$ be such that $Var(\sigma[(\tau_1, \rho_1), \ldots, (\tau_n, \rho_n)]) \cap Var(\sigma) = \emptyset$. If $\tau_i \equiv \tau_{i1} \to \tau_{i2}$ and $\rho_i \equiv \rho_{i1} \to \rho_{i2}$, for some $i \in \{1, \ldots, n\}$, then*

$$\begin{aligned}&\mathcal{U}_{GoI}(\sigma[(\tau_1, \rho_1), \ldots, (\tau_i, \rho_i), \ldots, (\tau_n, \rho_n)],\ \sigma)\\ &\qquad\Leftrightarrow \mathcal{U}_{GoI}(\sigma'[(\tau_1, \rho_1), \ldots, (\tau_{i1}, \rho_{i1}), (\tau_{i2}, \rho_{i2}), \ldots, (\tau_n, \rho_n)],\ \sigma')\end{aligned}$$

where $\sigma'[(\alpha_{i1}, \alpha_{i1}), (\alpha_{i2}, \alpha_{i2})] \equiv \sigma[(\alpha_{i1} \to \alpha_{i2}, \alpha_{i1} \to \alpha_{i2})]$, for α_{i1}, α_{i2} fresh variables.

Notice that the application of Lemma 26 only modifies the second term of the pair, *i.e.* σ, but in a way to preserve the property that its variables occur exactly twice. The proof is again straightforward, since execution sequences on the left hand side are immediately reflected on the right hand side, and vice versa.

▶ **Example 27.** Let $\sigma \equiv \alpha \to \alpha$, $\tau \equiv \delta \to (\beta \to \beta) \to \delta$, and $\rho \equiv \gamma \to \gamma \to \gamma$, where $\to$ associates to the right, as usual. Then $\sigma[(\tau, \rho)] \equiv (\delta \to (\beta \to \beta) \to \delta) \to (\gamma \to \gamma \to \gamma)$. We have:

$$\tau \equiv \tau_1 \to \tau_2,\ \text{where } \tau_1 \equiv \delta \text{ and } \tau_2 \equiv (\beta \to \beta) \to \delta\ ,$$

$$\rho \equiv \rho_1 \to \rho_2,\ \text{where } \rho_1 \equiv \gamma \text{ and } \rho_2 \equiv (\gamma \to \gamma)\ .$$

Applying Lemma 26, we get

$$\mathcal{U}_{GoI}(\sigma[(\tau,\rho)],\ \sigma) \iff \mathcal{U}_{GoI}(\sigma'[(\tau_1,\rho_1),(\tau_2,\rho_2)],\sigma') \ ,$$

where $\sigma' \equiv (\alpha_1 \to \alpha_2) \to (\alpha_1 \to \alpha_2)$.
Now both τ_2 and ρ_2 are arrow types, and so we can apply Lemma 26 again, getting:

$$\mathcal{U}_{GoI}(\sigma'[(\tau_1,\rho_1),(\tau_2,\rho_2)],\ \sigma') \iff \mathcal{U}_{GoI}(\sigma''[(\tau_1,\rho_1),(\tau_{21},\rho_{21}),(\tau_{22},\rho_{22})],\ \sigma'') \ ,$$

where

- $\sigma'' \equiv (\alpha_1 \to (\alpha_{21} \to \alpha_{22})) \to (\alpha_1 \to (\alpha_{21} \to \alpha_{22}))$
- $\tau_{21} \equiv \beta \to \beta$ and $\rho_{21} \equiv \gamma$
- $\tau_{22} \equiv \delta$ and $\rho_{22} \equiv \gamma$.

Notice that the pairs (τ_1,ρ_1) and (τ_{22},ρ_{22}) coincide.
At this point Lemma 26 is not applicable anymore. ⌟

▶ **Lemma 28.** *Let σ be a term where all variables occur exactly twice, and let $\sigma[(\tau_1,\rho_1),\ldots,(\xi,\rho_i),\ldots,$
$(\tau_n,\rho_n)]$ be such that ξ is a variable, and $Var(\sigma[(\tau_1,\rho_1),\ldots,(\xi,\rho_i),\ldots,(\tau_n,\rho_n)]) \cap Var(\sigma) = \emptyset$. Then,*

- *if $\xi \in Var(\rho_i)$ and $\xi \not\equiv \rho_i$, then $\sigma[(\tau_1,\rho_1),\ldots,(\xi,\rho_i),\ldots,(\tau_n,\rho_n)]$ and σ are not GoI-unifiable;*
- *if $\xi \notin Var(\rho_i)$,*

$$\mathcal{U}_{GoI}(\sigma[(\tau_1,\rho_1),\ldots,(\xi,\rho_i),\ldots,(\tau_n,\rho_n)],\ \sigma) \iff \mathcal{U}_{GoI}(\sigma[(\tau_1,\rho_1),\ldots,(\xi,\rho_i),\ldots,(\tau_n,\rho_n)][\rho_i/\xi_i],\ \sigma) \ .$$

And vice versa, i.e. the statement holds for $\sigma[(\tau_1,\rho_1),\ldots,(\tau_i,\xi),\ldots,(\tau_n,\rho_n)]$, where the variable ξ appears as second element in a pair.

The only not immediate part of the proof of the above lemma concerns the case where $\xi \in Var(\rho_i)$ and $\xi \not\equiv \rho_i$. For simplicity, we just consider the case where $\sigma \equiv \alpha \to \alpha$. The argument generalizes quite easily. If $\mathcal{U}_{GoI}(\xi \to \rho_i, \alpha \to \alpha)$, there exists a GoI-execution sequence which simulates the occurrence pair $\langle l[\xi], ru[\xi]\rangle$. But whatever substitution which is mediated by $\langle l[\alpha], r[\alpha]\rangle$, it will always produce occ-subsitutions of the same length in both occurrences, thereby preventing any GoI-execution sequence from reproducing the asymmetry in the original pair through an occ-substitution.

▶ **Example 29.** Continuing from Example 27, we can now apply Lemma 28 above to the last two terms $\sigma''[(\tau_1,\rho_1),(\tau_{21},\rho_{21}),(\tau_{22},\rho_{22})]$ and σ'', by considering the first pair (τ_1,ρ_1), and taking, *e.g.*, $\xi \equiv \delta$. Then we get:

$$\mathcal{U}_{GoI}(\sigma''[(\tau_1,\rho_1),(\tau_{21},\rho_{21}),(\tau_{22},\rho_{22})],\ \sigma'') \iff \mathcal{U}_{GoI}(\sigma''[(\tau_1,\rho_1),(\tau_{21},\rho_{21}),(\tau_{22},\rho_{22})][\gamma/\delta],\ \sigma'')$$

where the latter becomes

$$\mathcal{U}_{GoI}((\gamma \to (\beta \to \beta) \to \gamma) \to (\gamma \to \gamma \to \gamma),\ (\alpha_1 \to (\alpha_{21} \to \alpha_{22})) \to (\alpha_1 \to (\alpha_{21} \to \alpha_{22})) \ .$$

Then, applying again Lemma 28 to the pair (τ_{21},ρ_{21}), *i.e.* $(\beta \to \beta, \gamma)$, and performing the corresponding substitution, we finally get

$$\mathcal{U}_{GoI}(((\beta \to \beta) \to (\beta \to \beta) \to (\beta \to \beta)) \to ((\beta \to \beta) \to (\beta \to \beta) \to (\beta \to \beta)), (\alpha_1 \to (\alpha_{21} \to \alpha_{22})) \to (\alpha_1 \to (\alpha_{21} \to \alpha_{22})) \ .$$

Notice that at this point Lemma 26 above becomes again applicable, since the above coincides with

$$\mathcal{U}_{GoI}(\sigma''[((\beta \to \beta),(\beta \to \beta)),((\beta \to \beta),(\beta \to \beta)),((\beta \to \beta),(\beta \to \beta))],\ \sigma'')\ ,$$

where $\sigma'' \equiv (\alpha_1 \to (\alpha_{21} \to \alpha_{22})) \to (\alpha_1 \to (\alpha_{21} \to \alpha_{22}))$. ⌟

Clearly the above Lemmata 25, 26, 28 hold also for $\mathcal{U}$. Hence we can state:

▶ **Lemma 30.** *Lemmata 25, 26, and 28 hold by replacing $\mathcal{U}_{GoI}$ by $\mathcal{U}$.*

Finally, we have:

▶ **Lemma 31.** *Let σ be a term where all its n variables occur exactly twice. Then*

$$\mathcal{U}_{GoI}(\sigma[(\beta_1,\beta_1),\dots,(\beta_n,\beta_n)],\ \sigma) \Longleftrightarrow \mathcal{U}(\sigma[(\beta_1,\beta_1),\dots,(\beta_n,\beta_n)],\ \sigma)$$

where all the β_i's are fresh, but some of them can be equal.

Now we are ready to prove the main Theorem 17. Namely, we can proceed as follows.

6.1 GoI-unification $\Longrightarrow$ Unification

Assume σ,τ do not share common variables and $\mathcal{U}_{GoI}(\sigma,\tau)$. Then, by Lemma 25, $\mathcal{U}_{GoI}(\sigma \to \tau, \alpha \to \alpha)$, for α fresh variable. Now we repeatedly apply Lemmata 26 or 28 in the ($\Rightarrow$)-direction, until neither is applicable anymore. This procedure is guaranteed to terminate, because the complexity of the pair of terms to GoI-unify decreases at each step, according to the following definition of complexity measure:

▶ **Definition 32.** *Let σ be a term where all its n variables occur exactly twice, and let $\sigma[(\tau_1,\rho_1),\dots,(\tau_n,\rho_n)]$ be such that $Var(\sigma[(\tau_1,\rho_1),\dots,(\tau_n,\rho_n)]) \cap Var(\sigma) = \emptyset$. We define the complexity of the pair of terms $(\sigma[(\tau_1,\rho_1),\dots,(\tau_n,\rho_n)],\sigma)$ as the pair (m_t,m_v), where*

- *m_t is the sum of the complexities of the terms $\tau_1,\dots,\tau_n,\rho_1,\dots,\rho_n$,*
- *m_v is the number of variables appearing in $(\tau_1,\rho_1),\dots,(\tau_n,\rho_n)$.*

Now notice that the application of Lemmata 26 and 28 in the ($\Rightarrow$)-direction decreases the complexity of the pair of terms.

Hence, by repeatedly applying these lemmata, we reach a pair of the shape $(\overline{\sigma}[(\beta_1,\beta_1),\dots,(\beta_n,\beta_n)],\overline{\sigma})$, where $\overline{\sigma}$ has n variables and all the β_i's are fresh (some of them can possibly coincide). Therefore Lemma 31 applies, and we finally get $\mathcal{U}(\overline{\sigma}[(\beta_1,\beta_1),\dots,(\beta_n,\beta_n)],\overline{\sigma})$. At this point Lemmata 26 and 28 can be applied to $\mathcal{U}$ in the reverse direction, until we reach $\mathcal{U}(\sigma \to \tau, \alpha \to \alpha)$ (with α fresh), which in turn is equivalent to $\mathcal{U}(\sigma,\tau)$. This completes the proof.

6.2 Unification $\Rightarrow$ GoI-unification

In order to show the converse, just observe that Lemmata 25, 26, 28 hold also for Martelli-Montanari's unification algorithm $\mathcal{U}$, and apply the steps above, exchanging the role of $\mathcal{U}_{GoI}$ with $\mathcal{U}$, starting from $\mathcal{U}(\sigma,\tau)$.

7 Generalizations

Here we show how to remove all the restrictions on terms that we have considered up to now. This is achieved by providing suitable embeddings of terms including constants or/and hapax variables into the set of terms T_{Σ}^{-}, and by showing how the case of terms with common variables can be reduced to the case of terms which do not share variables.

7.1 Constants and Hapax Variables

Terms with constants can be easily embedded in the set of terms without constants and hapax variables, in such a way that the unification problem does not change. Namely, for any 0-ary constructor f_0^i, we introduce a fresh binary constructor $f_2^{i_k}$ and a fresh variable α_i, and we substitute f_0^i by $f_2^{i_k}(\alpha_i, \alpha_i)$.

Once constants have been eliminated via the simple encoding above, we can address the issue of variables which occur only once, *i.e.* hapax variables. In order to extend Definition 16 to include terms with hapax variables, we can simply give the following definition.

▶ **Definition 33.** *Let σ, τ be terms, possibly including hapax variables. Then σ and τ GoI-unify if there exists a substitution U such that $U(\sigma)$ and $U(\tau)$ do not contain hapax variables and are GoI-unifiable.*

The above definition, however, is highly non-effective. To achieve effectiveness we can go in the opposite direction and embed the two terms in larger terms with no hapax variables as follows:

▶ **Definition 34** (GoI-unification with hapax variables)**.** *Let $\sigma[\alpha_1, \ldots, \alpha_n], \tau[\beta_1, \ldots, \beta_m] \in T_\Sigma$ be terms where $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_m$ are the hapax variables in σ and τ respectively. Pick a fresh $1 + 2n + 2m$ constructor, say $f^k_{1+2n+2m}$, then σ and τ GoI-unify if the terms*

$$f^k_{1+2n+2m}(\sigma[\alpha_1, \ldots, \alpha_n], \alpha_1, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \beta_1, \ldots, \beta_m)$$

and

$$f^k_{1+2n+2m}(\tau[\beta_1, \ldots, \beta_m], \alpha_1, \ldots, \alpha_n, \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m, \beta_1, \ldots, \beta_m)$$

GoI-unify.

The intuition behind the above definition is immediate. We have to duplicate variables in both terms because the set of hapax variables are disjoint and possibly of different cardinality. Of course we can give more involved definitions which reduce the number of extra occurrence pairs to consider, but we do not pursue this issue further. Using Definition 34 we can give the appropriate extensions of Theorem 17, and its consequences, also for terms with hapax variables.

7.2 Unifying Terms with Common Variables

Up to here we have assumed that terms to unify do not share common variables. The reason for this is that, given a term σ, the binary relation $\mathcal{R}(\sigma)$ does not keep track of the name of the variable, which basically plays the role of the place holder for the tail-end of the path. For instance, applying directly Definition 16 to $\mathcal{R}((\alpha \multimap \alpha) \multimap \alpha \multimap \alpha)$ and $\mathcal{R}(\alpha \multimap \alpha)$ would yield success. But clearly the two terms $(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$ and $\alpha \multimap \alpha$ do not unify, if the name of the variable in the two terms has to be taken seriously. Now, we show how we can remove the assumption that the terms to unify do not share variables. We have essentially already solved the problem in Section 6. Namely, if we want to GoI-unify σ and τ, which have variables in common, then we can GoI-unify the new terms $f_2^k(\sigma, \tau)$ and $f_2^k(\alpha, \alpha)$, where α is fresh, and f_2^k is a new binary constructor. Hence we give the following definition:

▶ **Definition 35.** *Let $\sigma, \tau \in T_\Sigma$, let α be a fresh variable, and let f_2^k be a new binary constructor. Then σ and τ GoI-unify if and only if $f_2^k(\sigma, \tau)$ and $f_2^k(\alpha, \alpha)$ GoI-unify.*

8 Conclusion and Final Remarks

In this paper we provide an alternative criterion for checking that two terms unify in terms of Girard's GoI. It can indeed be seen as the GoI model of unification. The new definition amounts to carrying out successful GoI-execution sequences over the terms under consideration. It is inspired by [19], where an explanation, in terms of resolution of principal types, is given of the notion of linear application between purely linear λ-terms, in the line of the game model in [1]. In a sense, this paper elaborates further on the close relation between the two main logic-derived paradigms of computation, namely *cut-elimination* and *resolution*.

The following remarks are in order.

Recovering substitutions from occ-substitutions. Considering all possible GoI-execution sequences, we can derive *unifiers* thereby providing an analogue of Proposition 11. Namely we conjecture the following:

▶ **Conjecture 36.** *Let $\sigma, \tau \in T_\Sigma$ be terms, and suppose $\mathcal{U}_{GoI}(\sigma, \tau)$. Let $\Gamma^{\langle u[\alpha], v[\alpha]\rangle}$ denote a successful GoI-execution sequence for a pair of variable occurrences $\langle u[\alpha], v[\alpha]\rangle$, and let $M^{\Gamma^{\langle u[\alpha], v[\alpha]\rangle}}_{\langle u[\alpha], v[\alpha]\rangle}$ be the overall occ-substitution generated along $\Gamma^{\langle u[\alpha], v[\alpha]\rangle}$, simply denoted by $M^{\Gamma^{\langle u[\alpha], v[\alpha]\rangle}}$ in the sequel. Define the substitution*

$$U_{GoI}(\alpha) = \begin{cases} \mathcal{T}(\{M^{\Gamma^{\langle u[\alpha], v[\alpha]\rangle}}(\alpha) \mid \langle u[\alpha], v[\alpha]\rangle \in \mathcal{R}(\sigma)\}) & \textit{if } \alpha \in \sigma \\ \mathcal{T}(\{M^{\Gamma^{\langle u[\alpha], v[\alpha]\rangle}}(\alpha) \mid \langle u[\alpha], v[\alpha]\rangle \in \mathcal{R}(\tau)\}) & \textit{if } \alpha \in \tau \ , \end{cases}$$

then $U_{GoI}(\sigma) = U_{GoI}(\tau)$.

Moreover, we further conjecture that, if throughout the GoI-execution sequences of Lemma 18 the most general occ-substitutions are always picked, then U_{GoI} is the m.g.u. of σ and τ.

We only provide the intuition behind the above statement. If σ and τ GoI-unify, then for all occurrence pairs $\langle u[\alpha], v[\alpha]\rangle \in \mathcal{R}(\sigma)$ and GoI sequences $\Gamma^{\langle u[\alpha], v[\alpha]\rangle}$ we have, by Lemma 18, that there are occurrences in a substitution instance of τ which match the occurrence $M^{\Gamma^{\langle u[\alpha], v[\alpha]\rangle}}(u[\alpha])$. For the definition in Proposition 36 to be well-defined we need to know that the set $\{M^{\Gamma^{\langle u[\alpha], v[\alpha]\rangle}}(\alpha) \mid \langle u[\alpha], v[\alpha]\rangle \in \mathcal{R}(\sigma)\}$ consists of compatible occurrences, so that Proposition 6 can apply. This is not immediate, but is should ultimately follow from the fact that the substitutions arise from pairs in $\mathcal{R}(\tau)$ which in turn are compatible, using a maximal execution path which passes through all the occurrence pairs of the variables involved. Finally, the fact that one has chosen all possible execution sequences $\Gamma^{\langle u[\alpha], v[\alpha]\rangle}$ for each occurrence pair ensures that, in applying Proposition 6, no extra new variable in Z is necessary.

Comparison between *unification* and *GoI-unification*. *GoI-unification* is more a criterion rather than an algorithm, although it permits to synthesize the unifier, when it exists, as shown above. Usual unification requires a repeated deep substitution of terms for variables, while *GoI unification* only requires pattern matching along GoI execution sequences. Moreover not all possible GoI execution sequences need to be computed to check that unification achieves. If we were to use Definition 16 for checking unification rather than ordinary unification, termination would be more difficult to deal with. Termination is easily proved for standard *unification* by induction on syntax, while termination of *GoI-unification* would require to inspect all the possible finitely many paths along the occurrence terms, or a maximising path which uses all possible occurrence pairs for the variables involved.

Resolution. In [18], only GoI linear application, between purely linear binary types, was discussed in terms of resolution on terms formed using a single binary constructor, namely $\multimap$. Clearly the definitions in this paper can be easily extended to encompass resolution in full generality.

Duplication. We can extend this paper by dealing also with other meta-operations on terms besides substitution, namely *replication* of subterms as arise in Linear Logic. This would amount to generalizing the language of occurrences in the line of [9, 18, 11]. Giving a top-down account of replication is rather complex, in that a unification-duplication algorithm needs to be defined (see [11]). On the other hand, a bottom-up approach comes naturally following the literature on GoI, as Girard has shown, see *e.g.* [13, 1].

Invariants. In [1] only resolutions on copy-cat strategies, *i.e.* partial involutions, were considered, which in our setting amount to terms where each variable occurs exactly twice. On the other hand, in this paper we open up the possibility of using arbitrary terms, *i.e.* non-deterministic strategies. It would be worthwhile to study the λ-models which arise using resolution on such strategies to model β-reduction or, equivalently, the *invariants* of λ-terms that such strategies can express.

References

1 Samson Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441–464, 2005. doi:10.1016/j.tcs.2005.07.002.

2 Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of Interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002. doi:10.1017/S0960129502003730.

3 Samson Abramsky and Marina Lenisa. Linear realizability and full completeness for typed lambda-calculi. *Annals of Pure and Applied Logic*, 134(2):122–168, 2005. doi:10.1016/j.apal.2004.08.003.

4 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The Machinery of Interaction. In *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming*, PPDP '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3414080.3414108.

5 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Multi types and reasonable space. *Proc. ACM Program. Lang.*, 6(ICFP), August 2022. doi:10.1145/3547650.

6 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Reasonable Space for the λ-Calculus, Logarithmically. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533362.

7 Marc Bagnol. *On the Resolution Semiring.* Theses, Aix-Marseille Universite, December 2014. URL: https://theses.hal.science/tel-01215334.

8 Patrick Baillot and Marco Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2):1–31, 2001. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi45-1-2-02.

9 Alberto Ciaffaglione, Pietro Di Gianantonio, Furio Honsell, Marina Lenisa, Ivan Scagnetto, et al. λ!-calculus, Intersection Types, and Involutions. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 131. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2019.

10 Alberto Ciaffaglione, Furio Honsell, Marina Lenisa, Ivan Scagnetto, et al. The involutions-as-principal types/application-as-unification analogy. *EPiC Series in Computing*, 57:254–270, 2018. doi:10.29007/NTWG.

11 Pietro Di Gianantonio and Marina Lenisa. Principal Types as Lambda Nets. In Henning Basold, Jesper Cockx, and Silvia Ghilezan, editors, *27th International Conference on Types for Proofs and Programs (TYPES 2021)*, volume 239 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TYPES.2021.5`.

12 Boris Eng. *An exegesis of transcendental syntax*. Theses, Université Sorbonne Paris Nord, June 2023. URL: `https://hal.science/tel-04179276`.

13 Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, volume 127 of *Studies in Logic and the Foundations of Mathematics*, pages 221–260. Elsevier, 1989. `doi:10.1016/S0049-237X(08)70271-4`.

14 Jean-Yves Girard. Geometry of interaction 2: Deadlock-free algorithms. In Per Martin-Löf and Grigori Mints, editors, *COLOG-88*, pages 76–93, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

15 Jean-Yves Girard. Geometry of Interaction III: accommodating the additives. *London Mathematical Society Lecture Note Series*, pages 329–389, 1995.

16 Jean-Yves Girard. Transcendental syntax I: deterministic case. *Mathematical Structures in Computer Science*, 27(5):827–849, 2017. `doi:10.1017/S0960129515000407`.

17 Furio Honsell. Talk delivered at IFIP W.G. 2.2 Bologna Meeting, 2023.

18 Furio Honsell, Marina Lenisa, and Ivan Scagnetto. Λ-Symsym: An Interactive Tool for Playing with Involutions and Types. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs (TYPES 2020)*, volume 188 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.TYPES.2020.7`.

19 Furio Honsell, Marina Lenisa, and Ivan Scagnetto. Principal Types as Partial Involutions, 2024. `doi:10.48550/arXiv.2402.07230`.

20 Alberto Martelli and Ugo Montanari. An Efficient Unification Algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, April 1982. `doi:10.1145/357162.357169`.

On Approximation Schemes for Stabbing Rectilinear Polygons

Arindam Khan
Indian Institute of Science, Bengaluru, India

Aditya Subramanian
Indian Institute of Science, Bengaluru, India

Tobias Widmann
Technical University of Munich, Germany

Andreas Wiese
Technical University of Munich, Germany

Abstract

We study the problem of stabbing rectilinear polygons, where we are given n rectilinear polygons in the plane that we want to stab, i.e., we want to select horizontal line segments such that for each given rectilinear polygon there is a line segment that intersects two opposite (parallel) edges of it. Our goal is to find a set of line segments of minimum total length such that all polygons are stabbed. For the special case of rectangles, there is an $O(1)$-approximation algorithm and the problem is NP-hard [Chan, van Dijk, Fleszar, Spoerhase, and Wolff, 2018]. Also, the problem admits a QPTAS [Eisenbrand, Gallato, Svensson, and Venzin, 2021] and even a PTAS [Khan, Subramanian, and Wiese, 2022]. However, the approximability for the setting of more general polygons, e.g., L-shapes or T-shapes, is completely open.

In this paper, we give conditions under which the problem admits a $(1+\varepsilon)$-approximation algorithm. We assume that each input polygon is composed of rectangles that are placed on top of each other. We show that if all input polygons satisfy the *hourglass condition*, then the problem admits a quasi-polynomial time approximation scheme. In particular, it is thus unlikely that this case is APX-hard. Furthermore, we show that there exists a PTAS if each input polygon is composed out of rectangles with a bounded range of widths. On the other hand, we prove that the general case of the problem (in which the input polygons may not satisfy these conditions) is APX-hard, already if all input polygons have only eight edges. We remark that all polygons with fewer edges automatically satisfy the hourglass condition. For arbitrary rectilinear polygons we even show a lower bound of $\Omega(\log n)$ for the possible approximation ratio, which implies that the best possible ratio is in $\Theta(\log n)$ since the problem is a special case of Set Cover.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Approximation Algorithms, Stabbing, Rectangles, Rectilinear Polygons, QPTAS, APX-hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.27

Related Version *Full Version*: https://arxiv.org/abs/2402.02412 [34]

Funding *Arindam Khan*: Research partly supported by Pratiksha Trust Young Investigator Award, Google India Research Award, SERB Core Research Grant (CRG/2022/001176) on "Optimization under Intractability and Uncertainty" and Walmart Center for Tech Excellence.
Aditya Subramanian: Research was supported by the Walmart Center for Tech Excellence.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 27; pp. 27:1–27:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The Stabbing problem is a geometric case of the well-studied Set Cover problem. We are given a set of geometric objects in the plane. The goal is to compute a set of horizontal line segments of minimum total length such that each given object R is *stabbed*, i.e., there is a line segment ℓ for which $R \setminus \ell$ consists of two connected components. The problem was introduced by Chan, van Dijk, Fleszar, Spoerhase, and Wolff [9] for the case where each given object is an axis-parallel rectangle. In particular, they argued that this case models a resource allocation problem for frequencies. In this application, the x-axis models time and the y-axis represents a frequency spectrum. Each given rectangle represents a request for a time window $[t_1, t_2]$ and a frequency band $[f_1, f_2]$ that needs to be fulfilled. Each selected segment $[t'_1, t'_2] \times \{f'\}$ corresponds to opening a communication channel f' during a time interval $[t'_1, t'_2]$ which then serves each request whose time window is contained in $[t'_1, t'_2]$ and for which f is a frequency in its corresponding band $[f_1, f_2]$. Also, Das, Fleszar, Kobourov, Spoerhase, Veeramoni, and Wolff [13] showed a connection to the Generalized Minimum Manhattan Network problem.

The first result for the case of rectangles was a polynomial time $O(1)$-approximation due to Chan et al. [9]. Subsequently, Eisenbrand, Gallato, Svensson, and Venzin improved the approximation ratio to 8 and provided a QPTAS, i.e., a $(1+\varepsilon)$-approximation algorithm that runs in quasi-polynomial time [16]. In particular, this implies that the problem is unlikely to be APX-hard. After that, Khan, Subramanian, and Wiese presented a polynomial time approximation scheme (PTAS) for rectangles [35].

A natural question is the Stabbing problem for geometric shapes that are more general than rectangles. We restrict ourselves to rectilinear polygons. Rectilinear polygons can model more general types of requests in the resource allocation problem. Depending on the resource quality, the requested time period and preprocessing times for jobs may be different. This can be modeled as an instance of our problem, where each job is represented by multiple rectangular regions (each of them corresponds to a particular bandwidth interval and a time period during which the job may be processed), and the aim is to select bandwidths and corresponding time intervals such that each job is served. This corresponds to stabbing one of the rectangular regions corresponding to each job. If the rectangular regions are contiguous (which is quite common due to the locality of bandwidth requirements) they correspond to *k-shapes* which motivates studying these objects.

Also, from a theoretical point of view, it is natural to ask which approximation ratios are possible for more general geometric objects. As mentioned above, Stabbing admits a $(1+\varepsilon)$-approximation algorithm when all given objects are rectangles [35]. However, is this also true for slightly more general polygons, e.g., that have the shape of an L or a T, polyominoes, or even for arbitrary rectilinear polygons? If not, under which conditions on the input objects is a $(1+\varepsilon)$-approximation still possible? Also, given that Stabbing is a special case of Set Cover, another natural question is whether it is strictly easier than this problem.

In this paper, we investigate the questions above. We focus on a type of rectilinear polygons that we call *k-shapes*. Intuitively, a k-shape is formed by k rectangles that are stacked on top of each other such that for any two consecutive rectangles, the top edge of the bottom rectangle is contained in the bottom edge of the top rectangle, or vice versa (see Figures 1 and 2). We denote by k-Stabbing the setting of the Stabbing problem in which the input consists of k-shapes.

1.1 Our contribution

In this paper, we give conditions on k-shapes in the input, under which, the k-Stabbing problem admits a $(1+\varepsilon)$-approximation algorithm in (quasi-)polynomial time, which makes it unlikely that it is APX-hard in these cases. We provide two separate conditions for this. Also, we prove that if the input objects may (slightly) violate these conditions, then the problem becomes APX-hard. For arbitrary k-shapes, we prove even that the problem is as difficult as general Set Cover, which yields a lower bound of $\Omega(\log n)$ for the possible approximation ratio.

Our first condition on the input k-shapes is the *hourglass condition*. It requires intuitively that the rectangles of each k-shape in the input are stacked like an hourglass (see Figure 1 and Definitions 1 and 3). Formally, it states that if we consider the rectangle of each k-shape of the smallest width, then the rectangles on top of it are ordered non-decreasingly by width, and an analogous mirrored ordering holds for the rectangles below it. For example, all L-shapes and triominoes fulfill this condition. We prove that this setting admits a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ in quasi-polynomial running time, i.e., in time $n^{(\log n/\varepsilon)^{O(1)}}$. In particular, this makes it unlikely that this case is APX-hard. Our algorithm generalizes the known QPTAS for the case of rectangles [16]. However, it is arguably simpler. For example, it does not need an $O(1)$-approximation algorithm for the problem as a subroutine. Instead, we show that the calls to this subroutine can be replaced by suitable guessing steps and by an $O(\log n)$-approximation algorithm for general Set Cover.

Note that when we say *guess*, we mean that there are only a polynomial number of possible options to choose from. So one can iterate over all possible options and find one of the correct options in polynomial time.

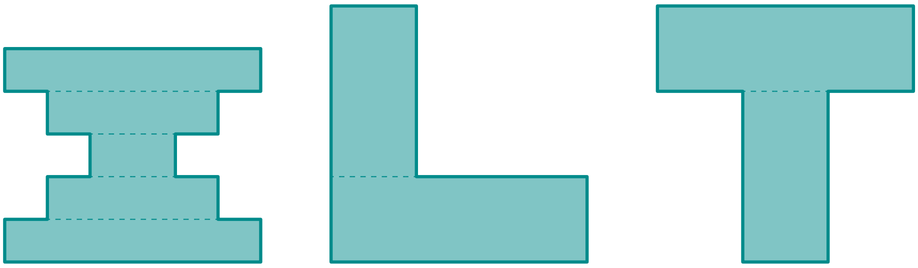


Figure 1 Examples of k-shapes satisfying the hourglass condition.

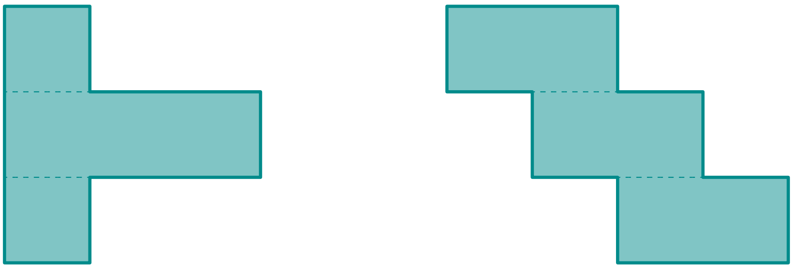


Figure 2 A 3-shape not satisfying the hourglass condition (left), and a stack of rectangles that does not form a k-shape (right).

Our algorithm is based on a hierarchical decomposition of the plane into smaller and smaller rectangular regions. Intuitively, given such a region R, we guess all line segments that are relatively long compared to the width of R. Then, we partition R into smaller rectangular regions inside which we will select only shorter line segments. It can happen that a k-shape K contained in R is composed of at least one wide rectangle (of similar width as the guessed long line segments) and of at least one narrow rectangle (see Figure 3). If the guessed long line segments do not stab K, then it is clear that K needs to be stabbed by a short line segment (that we select in one of the subproblems that we recurse into). Such line segments can stab only the narrow rectangles of K. Therefore, in this case we remove the wide rectangle from K and hence make K smaller. The hourglass condition ensures that after this removal, the remainder of K still consists of only one connected component. We crucially need this property in order to ensure that the subproblems of R we recurse into form independent subproblems. This would not be the case if the remainder of K consisted of two connected components such that each of them lies in a different subproblem.

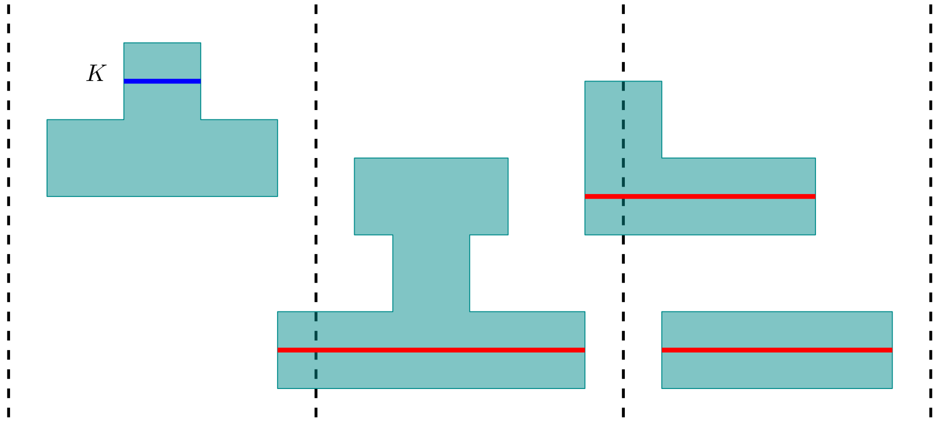


Figure 3 The guessed (red) long segments do not stab K, so it has to be stabbed by a shorter (blue) segment in a future step.

While the hourglass condition is crucial for our algorithm above, it could be that it is not needed in an alternative algorithmic approach that computes a $(1+\varepsilon)$-approximation for, e.g., general k-shapes. However, we prove that this is not the case. We show that our problem is APX-hard, already if the input consists only of 3-shapes that do not satisfy the hourglass condition. On the other hand, note that each 2-shape automatically satisfies the hourglass condition by definition.

In our proof of this APX-hardness result, we construct 3-shapes that are composed out of three rectangles whose widths differ a lot. We prove that the latter is necessary in order to prove that our problem is APX-hard. To this end, we show that it admits a polynomial time $(1+\varepsilon)$-approximation algorithm for any constant $k \in \mathbb{N}$ and $\varepsilon > 0$ if each k-shape is composed out of rectangles whose widths are in a *constant* range. This yields our second condition under which our problem admits a $(1+\varepsilon)$-approximation. In fact, our result can handle some other classes of polygons which may not even be k-shapes, including polyominoes with $O(1)$ number of cells such as trominoes, tetrominoes (shapes that appear in the game *Tetris*), pentominoes, etc. Our algorithm is a generalization of the PTAS for rectangles [35]. One crucial insight is that if the widths of the rectangles of each input k-shape differ by at most a constant factor of $1/\delta$, then we can reduce our problem to the setting of rectangles by losing only a factor of $O(k/\delta)$. To do this, we simply replace each k-shape K by the smallest rectangle that contains K. We use this insight in one step of our algorithm where we need an $O(1)$-approximation algorithm as a black box. More precisely, we again partition the input plane hierarchically into smaller and smaller rectangular regions. In the process, we repeatedly need to compute constant factor approximations for certain sets of k-shapes that intuitively admit a solution whose cost is at most $O(k\delta\varepsilon\mathrm{OPT})$; for those, we use the mentioned algorithm. We stab all other k-shapes with segments whose total cost is at most $(1+\varepsilon)\mathrm{OPT}$, which yields a PTAS.

We round up our results by showing that for general k-shapes and, more generally, even arbitrary rectilinear polygons that are composed of k rectangles each, Stabbing admits a polynomial time $O(k)$-approximation algorithm. A natural question is whether the dependence on k (and the input size) in the approximation ratio can be avoided and there is, e.g., also an $O(1)$-approximation. We show that this is not the case: for arbitrary k, we prove that k-Stabbing is as difficult as arbitrary instances of Set Cover, which yields a lower bound of $\Omega(\log n)$ for our approximation ratio.

1.2 Other related work

As mentioned above, the Stabbing problem is a special case of Set Cover which is NP-hard [23] and which does not admit a $(c \cdot \ln n)$-approximation algorithm for Set Cover for any $c < 1$, assuming that $\mathsf{P} \neq \mathsf{NP}$ [15] (see also [18]). On the other hand, a simple polynomial time greedy algorithm [12] achieves an approximation ratio of $O(\log n)$.

Das, Fleszar, Kobourov, Spoerhase, Veeramoni, and Wolff [13] studied approximation algorithms for the Generalized Minimum Manhattan Network (GMMN) problem, where given a set of n pairs of terminal vertices, the goal is to find a minimum-length rectilinear network such that each pair is connected by a Manhattan path. The currently best known approximation ratio for this problem is $(4+\varepsilon)\log n$, due to Khan, Subramanian, and Wiese [35] by using their PTAS for Stabbing as a subroutine in a variant of the algorithm of Das et al. [13].

Gaur, Ibaraki, and Krishnamurti [24] studied the problem of stabbing rectangles by a minimum number of axis-aligned lines and obtained an LP-based 2-approximation algorithm. Kovaleva and Spieksma [37] studied a weighted generalization of this problem and gave an $O(1)$-approximation algorithm.

Geometric set cover is a related geometric special case of general Set Cover, where the given sets are geometric objects. Brönnimann and Goodrich [5] first gave an $O(d\log(d\cdot \mathrm{OPT}))$-approximation algorithm for unweighted geometric set cover where d is the dual VC dimension of the set system and OPT is the value of the optimal solution. Aronov, Ezra, and Sharir [3] utilized ε-nets to design an $O(\log\log \mathrm{OPT})$-approximation algorithm for the hitting set problem involving axis-parallel rectangles. Varadarajan [46] provided an improved approximation algorithm for weighted geometric set cover for fat triangles or disks, and his techniques were extended by Chan, Grant, Könemann, and Sharpe [7] to any set system with low shallow cell complexity. Subsequently, Chan and Grant [6], and Mustafa, Raman, and Ray [42] have settled the APX-hardness statuses of (almost) all natural variants for this problem. Recently, these problems are studied under online and dynamic setting as well [2, 8, 31].

Maximum Independent Set of Rectangles is another related problem. The problem admits a QPTAS [1], and recently a breakthrough $O(1)$-approximation algorithm was given by Mitchell [41]. Subsequently, a $(2+\varepsilon)$-approximation guarantee [21] was achieved.

Rectangle packing and covering problems such as two-dimensional knapsack [19, 28, 32], two-dimensional bin packing [4, 33], strip packing [27, 30] etc. are well-studied in computational geometry and approximation algorithms. We refer the readers to [11] for a survey on the approximation/online algorithms related to rectangles.

Rectilinear polygons appear naturally in the context of circuit design [38], architectural design [44], geometric information systems [10], computer graphics [45], etc. In computational geometry, often problems (for general polygons) are studied in the rectilinear setting, e.g., the art gallery problem [47], rectilinear convex hull [43], and rectilinear steiner tree [22]. Specially, L-shape polygons are encountered in many geometric problems as they are the simplest nonconvex rectilinear polygons. These L-shapes appear in geometric packing [20, 32], folding [14], VLSI layouts [39], lithography [48], etc. Polyominoes [26, 40] are special type of rectilinear polygons that are formed by joining one or more equal squares edge to edge. They are well-studied in the context of tiling [25], percolation theory and statistical physics [29], polymer chemistry [17], etc. They also appear in many puzzles and board games, including Tetris, Blokus, Rampart, Cathedral, etc.

1.3 Organization of this paper

First, in Section 2 we introduce some basic definitions and notation. Then, in Section 3 we present our QPTAS for k-shapes satisfying the hourglass condition, and in Section 4 we present our PTAS for k-shapes whose rectangles have a bounded ratio of widths. Finally, in Section 5 we present our hardness results.

2 Preliminaries

We start with some basic definitions and notations. We represent a given axis-aligned rectangle R_i as the Cartesian product of two given closed and bounded intervals, i.e., $R_i = [x_i^\ell, x_i^r] \times [y_i^b, y_i^t]$ for given coordinates $x_i^\ell, x_i^r, y_i^b, y_i^t \in \mathbb{N}$, where $x_i^\ell \leq x_i^r$ and $y_i^b \leq y_i^t$. The following notation will be useful: we define

- $b(R_i) := [x_i^\ell, x_i^r] \times \{y_i^b\}$ as the *bottom edge* of R_i,
- $t(R_i) := [x_i^\ell, x_i^r] \times \{y_i^t\}$ as the *top edge* of R_i, and
- $w(R_i) := (x_i^r - x_i^\ell)$ as the *width* of R_i.

A *horizontal line segment* $s \subset \mathbb{R}^2$ is a Cartesian product $s = [x^\ell, x^r] \times \{y\}$ with coordinates $x^\ell, x^r, y \in \mathbb{N}$ and $x^\ell \leq x^r$. We say that s *stabs* the rectangle R_i if and only if $R_i \cap s = [x_i^\ell, x_i^r] \times \{y\}$. Also, we define $|s| := x^r - x^\ell$ to be the *length* or the *cost* of s. We will study the Stabbing problem in the setting where each given object is a k-shape.

▶ **Definition 1** (k-shape). *Let $k \in \mathbb{N}$. A k-shape K is the union of a sequence of at most k axis-aligned rectangles $(R_1, R_2, \dots, R_k)$ such that $t(R_i) \subseteq b(R_{i+1})$ or $t(R_i) \supseteq b(R_{i+1})$ for each $i \in \{1, \dots, k-1\}$.*

We say that a k-shape $K = R_1 \cup \dots \cup R_k$ is *stabbed* by a line segment s, if there exists an index $i \in \{1, \dots, k\}$ such that the rectangle R_i is stabbed by s. This leads to the following formal definition of the Stabbing problem for k-shapes.

▶ **Definition 2.** *Let $k \in \mathbb{N}$. An instance of the k-Stabbing problem is a finite set of k-shapes $\mathcal{K}$, where the objective is to find a set $\mathcal{S}$ of horizontal line segments of minimum total length, such that every k-shape in $\mathcal{K}$ is stabbed by a segment in $\mathcal{S}$.*

In the following section, we shall use the term OPT interchangeably to refer to the optimal solution to the problem, and also to represent its cost, i.e., the total length of segments in the set. Similarly, SOL will be used to represent a solution set and also its cost.

3 Quasi-polynomial-time approximation scheme

In this section, we present our QPTAS for k-Stabbing. The algorithm is an extension of the QPTAS for Stabbing [16] to the more general case of k-shapes; also, we simplify some of its steps.

Let $\varepsilon > 0$ and suppose we are given a set of k-shapes $\mathcal{K}$. In this section, we assume that each given k-shape $K \in \mathcal{K}$ satisfies the hourglass condition (see Figure 1).

▶ **Definition 3.** *A k-shape $K = (R_1, R_2, \dots, R_k)$ satisfies the* hourglass condition *if there is no value $i \in \{2, \dots, k-1\}$ such that both $w(R_{i-1}) < w(R_i)$ and $w(R_{i+1}) < w(R_i)$.*

For each given k-shape K, we define $w_{\max}(K) := \max_{i\in\{1,\dots,k\}} w(R_i)$ and similarly $w_{\min}(K) := \min_{i\in\{1,\dots,k\}} w(R_i)$ which are the widths of the widest and most narrow parts of K, respectively. For sets of k-shapes $\mathcal{K}' \subseteq \mathcal{K}$, we define accordingly $w_{\max}(\mathcal{K}') := \max_{K\in\mathcal{K}'} w_{\max}(K)$, $w_{\min}(\mathcal{K}') := \min_{K\in\mathcal{K}'} w_{\min}(K)$. Moreover, we define $w_{\text{range}}(\mathcal{K}') := \min\{w \mid \exists x \forall K \in \mathcal{K}' : K \subseteq [x, x+w] \times \mathbb{R}\}$ as the width of the most narrow strip that contains all k-shapes in $\mathcal{K}'$. Further, we note here that there are n given k-shapes and each is described by at most $2k$ distinct points. Therefore, the solution to the instance has only $\binom{2kn}{2}$ combinatorially distinct candidate segments, which is a polynomial in n (we shall use the notation that the number of candidate segments is $\text{poly}(n)$).

▶ **Lemma 4.** *By losing a factor of $1+\varepsilon$ in our approximation ratio, we assume that $\frac{\varepsilon}{n} < w_{\min}(\mathcal{K}) \leq w_{\max}(\mathcal{K}) \leq \log n$ and $w_{\text{range}}(\mathcal{K}) \leq n \log n$.*

Let $\mu := \varepsilon / \log^2 n$. We partition the plane into relatively wide vertical strips of width $w_{\max}(\mathcal{K})/\mu$ each. We do this such that, intuitively, almost all input shapes are contained in one of our strips, and the remaining shapes, which are intersected by the vertical grid lines, can be stabbed very cheaply. To construct this partition, we define vertical grid lines with a spacing of $w_{\max}(\mathcal{K})/\mu$ and give them a random horizontal shift (see Figure 4). Then, each shape in $\mathcal{K}$ intersects one of these grid lines only with very small probability. Therefore, we can show that there exists a specific way to perform the shift of our grid lines such that all input shapes intersecting our grid lines can be stabbed with line segments whose cost is at most $\mu \cdot \text{OPT}$.

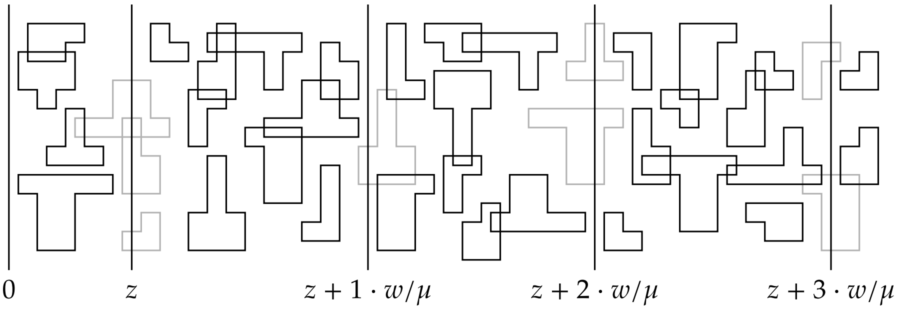


Figure 4 Partitioning the instance into narrow strips.

Formally, we invoke the following lemma with our choice for μ defined above. It guesses a set of line segments that yield our desired partition into narrow strips, i.e., it produces a polynomial number of candidate sets such that one of them has the claimed property. Algorithmically, we recurse on each of these polynomially many options and at the end output the returned solution with the smallest total cost.

▶ **Lemma 5** (Partitioning into narrow strips). *Let $\mu > 0$ such that $\mu/n < w_{\min}(\mathcal{K})$. In polynomial time, we can guess a partition of $\mathcal{K}$ into sets $\mathcal{K}_0, \dots, \mathcal{K}_t$ and one special set $\mathcal{K}_{\text{rest}}$ such that*

(i) $\text{OPT} \geq \sum_{\ell=1}^{t} \text{OPT}(\mathcal{K}_\ell)$,
(ii) $\text{OPT}(\mathcal{K}_{\text{rest}}) \leq 8\mu \cdot \text{OPT}$, *and*
(iii) $w_{\text{range}}(\mathcal{K}_i) \leq w_{\max}(\mathcal{K})/\mu$ *for each* $i \in \{1, \dots, t\}$.

We compute an $O(\log n)$-approximate solution for stabbing $\mathcal{K}_{\text{rest}}$ by reducing our problem to an instance of Set Cover (see full version [34] for details). By our choice of μ, the resulting cost is at most $O(\log n \cdot \mu \cdot \text{OPT}) = O(\text{OPT} \cdot \varepsilon/\log n)$. Hence, this step is simpler than the corresponding step in the previous QPTAS for Stabbing [16]. In that result, an $O(1)$-approximation algorithm for Stabbing was needed, while we can simply call an arbitrary standard $O(\log n)$-approximation algorithm for Set Cover, e.g., the straight-forward greedy algorithm.

Now let $\mathcal{K}_i$ be one of the sets of k-shapes due to Lemma 5. We define $S_i := [a, b] \times \mathbb{R}$ for some values $a, b \in \mathbb{R}$ with $b - a \leq w_{\max}(\mathcal{K})/\mu$ such that each k-shape in $\mathcal{K}_i$ is contained in S_i. We want to partition S_i along horizontal lines into rectangular pieces such that each resulting piece contains line segments from $\text{OPT}(\mathcal{K}_i)$ of total cost at most $O(w_{\max}(\mathcal{K})/\mu^2)$. To this end, we guess whether the segments in $\text{OPT}(\mathcal{K}_i)$ have a total cost of at most $w_{\max}(\mathcal{K})/\mu^2$. If this is not the case, we guess a line segment $s = [a, b] \times \{h\}$ for some value $h \in \mathbb{N}$ according to the following lemma, which intuitively partitions S_i in a balanced way according to the segments in $\text{OPT}(\mathcal{K}_i)$. We call such a segment s a *balanced horizontal cut*. Also, here our algorithm is simpler than the earlier QPTAS for Stabbing [16]. In the latter algorithm, an $O(1)$-approximate algorithm for the problem was used to find the "correct" horizontal cuts algorithmically. Instead, we show that it is sufficient to simply guess them.

▶ **Lemma 6.** *If* $\text{OPT}(\mathcal{K}_i) > w_{\max}(\mathcal{K})/\mu^2$ *then in polynomial time we can guess a value* $h \in \mathbb{N}$ *and a corresponding line segment* $s = [a, b] \times \{h\}$ *such that each connected component* C *of* $S_i \setminus s$ *contains segments from* $\text{OPT}(\mathcal{K}_i)$ *whose total cost is at least* $\text{OPT}(\mathcal{K}_i)/2 - w_{\max}(\mathcal{K})/\mu$.

Proof. Since we use only horizontal segments to stab k-shapes, w.l.o.g. (by stretching along the y direction) we can assume that the at most $2kn$ points describing the instance occupy consecutive integral y-coordinates, starting at $y = 0$. Note that such a stretching step along the y direction will not affect the length of the horizontal segments used to stab the k-shapes.

Consider the segments from $\text{OPT}(\mathcal{K}_i)$. Starting from $y = 0$ and going up, we can start counting the cumulative cost of segments in OPT. Let h be the y-coordinate at which this cumulative cost crosses $\text{OPT}(\mathcal{K}_i)/2$, and $s = [a, b] \times \{h\}$ be the corresponding segment. Since the width of S_i is at most $w_{\max}(\mathcal{K})/\mu$, no segment in $\text{OPT}(\mathcal{K}_i)$ is wider than $w_{\max}(\mathcal{K})/\mu$. From this we can infer that the cost of segments from $\text{OPT}(\mathcal{K}_i)$, below (and similarly, above) the segment s should have been at least $\text{OPT}(\mathcal{K}_i)/2 - w_{\max}(\mathcal{K})/\mu$.

Since there are only a polynomial (i.e., $2kn$) number of possible y-coordinates, we can guess this value h in polynomial time by enumeration. ◀

We add s to our solution and recurse on each connected component C of $S_i \setminus s$ separately. The resulting subproblem is to stab all input shapes that are contained in C. Observe that s stabs all k-shapes contained in S_i that intersect both connected components of $S_i \setminus s$. Given C, we guess again whether $\text{OPT}(C)$, i.e., the optimal solution for all k-shapes contained in C, has a total cost of at most $w_{\max}(\mathcal{K})/\mu^2$, and if not, we guess a corresponding horizontal line segment. Note that we stop after at most $O(\log n)$ recursion levels if all guesses are correct, since $\mathrm{OPT}(S_i) \leq \mathrm{OPT} \leq n \log n$ due to our preprocessing in Lemma 4. We enforce that in any case we stop after $O(\log n)$ recursion levels in order to guarantee a quasi-polynomial bound on the running time later.

▶ **Lemma 7.** *If all guesses for the balanced horizontal cuts are correct, then their total cost is bounded by* $3\mu \cdot \mathrm{OPT}(\mathcal{K}_i)$.

Proof. After our sequence of (correctly guessed) balanced horizontal cuts, let us assume that there are t connected components, with cost at least $w_{\max}(\mathcal{K}_i)/2\mu^2 - w_{\max}(\mathcal{K})/\mu$. This can happen only if there were $t-1$ such cuts applied. If we charge the cost of every cut s to the cost of segments of $\mathrm{OPT}(\mathcal{K}_i)$ within a cell C, we get

$$\frac{|s|}{\mathrm{OPT}(C)} = \frac{w_{\max}(\mathcal{K})/\mu}{w_{\max}(\mathcal{K})/2\mu^2 - w_{\max}(\mathcal{K})/\mu} = \frac{2\mu}{1-2\mu} \leq 3\mu.$$

Where the last inequality follows under the assumption of $\mu \leq \varepsilon < 1/3$. Summing over all such horizontal cuts, we get the total cost to be at most $3\mu \cdot w_{\max}(\mathcal{K}_i)$. ◀

At the end, each resulting subproblem is characterized by a rectangle C of width at most $w_{\max}(\mathcal{K})/\mu$ and for which $\mathrm{OPT}(C) \leq w_{\max}(\mathcal{K})/\mu^2$. We guess all line segments in $\mathrm{OPT}(C)$ whose width is larger than $\varepsilon w_{\max}(\mathcal{K})$. Since $\mathrm{OPT}(C) \leq w_{\max}(\mathcal{K})/\mu^2$ there can be at most $1/\varepsilon\mu^2 = \varepsilon^{-3} \log^4 n$ of them, and for each of them there are only poly(n) options. Hence, we can guess them in time $n^{O(\varepsilon^{-3} \log^4 n)}$. Let $\mathcal{S}_C$ denote the guessed segments.

Our next step crucially differs from the known (Q)PTASs for stabbing rectangles [16, 35]. In particular, it is not necessary when all input objects are rectangles. Inside C, there might be a k-shape K that is not stabbed by any segment in $\mathcal{S}_C$ but for which one of its rectangles R_i satisfies that $w(R_i) > \varepsilon w_{\max}(\mathcal{K})$. Since we have guessed all segments in C of width larger than $\varepsilon w_{\max}(\mathcal{K})$ and did not yet stab K, we know that the optimal solution does not stab K by stabbing R_i (but by stabbing another rectangle that K is composed of). Therefore, we modify K by removing R_i from K. We do this for each rectangle R_i with $w(R_i) > \varepsilon w_{\max}(\mathcal{K})$ that is part of a k-shape K that is contained in C but not yet stabbed. Denote by $\mathcal{K}'(C)$ the resulting set of k-shapes. Importantly, the hourglass property implies that still each k-shape has only one single connected component. This is the reason why we imposed this property.

Observe that for each $K \in \mathcal{K}'(C)$ we have that $w_{\max}(K) \leq \varepsilon \cdot w_{\max}(\mathcal{K})$. Thus, we made progress in the sense that the maximum width of any k-shape reduces by a factor of ε. Also, if all our guesses are correct, then our total cost is small, i.e., $O(\mu \cdot \mathrm{OPT})$. Also, the number of guesses is quasi-polynomially bounded since for each guess there are only $n^{O(\varepsilon^{-3} \log^4 n)}$ many options and our recursion depth is only $O(\log n)$.

▶ **Lemma 8.** *If all our guesses are correct, then the total cost for the selected line segments due to Lemma 5 and Lemma 7 is bounded by* $O(\mu \cdot \mathrm{OPT})$*. Also, the total number of (combinations of) guesses is bounded by* $n^{O(\varepsilon^{-3} \log^4 n)}$.

We continue recursively with each resulting subproblem. Since initially $\frac{\varepsilon}{n} < w_{\min}(\mathcal{K}) \leq w_{\max}(\mathcal{K}) \leq \log n$, we stop after applying the algorithm above for $O(\log(n/\varepsilon))$ levels. Each level incurs in total at most $n^{O(\varepsilon^{-3} \log^4 n)}$ guesses, which yields a total running time of $n^{O(\varepsilon^{-4} \log^5 n)}$. Also, our approximation ratio can easily be bounded by $(1+\mu)^{O(\log n)} = 1 + O(\varepsilon)$.

▶ **Theorem 9.** *There is a QPTAS for the k-Stabbing problem, if all input k-shapes satisfy the hourglass condition.*

4 PTAS if pieces have bounded ratio of widths

In this section, we improve our QPTAS from Section 3 to a PTAS in the special case when k is a constant and when for each given k-shape, for any two of its rectangles R_i, R_j, it holds that $\delta w(R_j) \leq w(R_i) \leq w(R_j)/\delta$ for a given constant $\delta > 0$. Our algorithm generalizes the known PTAS for the case when all input objects are rectangles [35].

Let α be a constant for which the problem admits a polynomial time α-approximation algorithm. We show in the full version [34] that such an algorithm exists. Without loss of generality, we assume that $\alpha, (1/\varepsilon) \in \mathbb{N}$, and we say that an x-coordinate $x \in \mathbb{R}$ is *discrete* if x is an integral multiple of ε^d, where we define $d \in \mathbb{N}$ such that $\varepsilon^3/n < \varepsilon^d \leq \varepsilon^2/n$; note that hence d is unique. Similarly, a y-coordinate is called *discrete* if it is integral. A point is called *discrete* if its x- and y-coordinates are discrete, and similarly a segment or a rectangle is said to be *discrete* if both of its end points, or both of its diagonally opposite corners are discrete.

▶ **Lemma 10.** *Let α be a constant for which k-Stabbing admits an α-approximate algorithm and let $\varepsilon > 0$ with $\varepsilon < 1/3$. In polynomial time we can compute a new instance of k-Stabbing, in which each $K \in \mathcal{K}$ satisfies,*

(i) $\frac{\alpha\varepsilon}{n} < w_{\min}(K) \leq w_{\max}(K) \leq \alpha$,

(ii) *all points defining K are discrete,*

(iii) *K lies within a bounding box of $[0, \alpha n] \times [0, (k+1)n]$,*

and this new instance admits a solution of cost at most $(1 + O(\varepsilon)) \cdot \mathrm{OPT}$ with each segment in the solution being discrete, and having length at most α/ε.

First, we apply Lemma 10 in order to preprocess our instance. In our algorithm, we intuitively embed the recursion of our QPTAS in Section 3 into a polynomial time dynamic program, such that we can afford to forget most of the balanced horizontal cuts from the higher levels, and only need to remember a constant number of the corresponding line segments. The idea is to construct a DP-table that contains one cell for each possible subproblem of a recursive call. Formally, we introduce one DP cell $\mathrm{DP}(R, \mathcal{S})$ for each combination of

- a closed rectangle $R \subseteq [0, \alpha n] \times [0, (k+1)n]$ with discrete coordinates,
- a set $\mathcal{S}$ of at most ε^{-3} discrete horizontal line segments, that all intersect R.

This DP cell encodes the subproblem of stabbing all input k-shapes that are contained in R and that are not already stabbed by the segments in $\mathcal{S}$. Clearly, the DP cell $\mathrm{DP}([0, \alpha n] \times [0, (k+1)n], \emptyset)$ corresponds to our given problem.

Given a DP cell $\mathrm{DP}(R, \mathcal{S})$, we compute its solution as follows. The base case occurs when the line segments in $\mathcal{S}$ already stab all k-shapes that are contained in R. Then we define $\mathrm{DP}(R, \mathcal{S}) := \emptyset$. Another easy case occurs when there is a line segment $\ell \in \mathcal{S}$ that stabs the interior of R, i.e., $R \setminus \ell$ has two connected components R_1 and R_2. Assume that $\mathcal{S}_1$ and $\mathcal{S}_2$ are parts of the line segments from $\mathcal{S}$ that intersect R_1 and R_2, respectively. Then we define $\mathrm{DP}(R, \mathcal{S}) := \mathrm{DP}(R_1, \mathcal{S}_1) \cup \mathrm{DP}(R_2, \mathcal{S}_2) \cup \{\ell\}$. We will refer to this later as the *trivial operation.*

Otherwise, we compute a polynomial number of candidate solutions as follows,

1. *Add operation.* For each set $\mathcal{S}'$ of discrete segments contained in R for which $|\mathcal{S}| + |\mathcal{S}'| \leq 3\varepsilon^{-3}$ holds, we generate the candidate solution $\mathcal{S}' \cup \mathrm{DP}(R, \mathcal{S} \cup \mathcal{S}')$.
2. *Line operation.* Consider each vertical line ℓ that intersects the interior of R. Let $\mathcal{K}_\ell$ denote the set of k-shapes contained in R that intersect with ℓ. For each $K \in \mathcal{K}_\ell$ we construct the smallest axis-parallel rectangle that contains K, let $\mathcal{R}_\ell$ denote the resulting set of rectangles. We apply the PTAS for stabbing rectangles [35] to $\mathcal{R}_\ell$, let $\mathcal{S}_\ell$ denote the computed set of segments. We will show later that the optimal solution for $\mathcal{R}_\ell$ is by

at most a factor $O(k/\delta)$ more expensive that the optimal solution for $\mathcal{K}_\ell$, and that this approximation ratio is good enough for our purposes in this step. Denote by R_1 and R_2 the connected components of $R\backslash\ell$ and by $\mathcal{S}_1$ and $\mathcal{S}_2$ the parts of segments from $\mathcal{S}$ that intersect R_1 and R_2, respectively. We define the candidate solution $\mathcal{S}_\ell \cup \text{DP}(R_1, \mathcal{S}_1) \cup \text{DP}(R_2, \mathcal{S}_2)$.

We store in $\text{DP}(R, \mathcal{S})$ the candidate solution with the smallest cost. Finally, we output the solution stored in the cell $\text{DP}([0, \alpha n] \times [0, 2kn], \emptyset)$.

Analysis

We first note that all DP subproblems and operations are defined on discrete coordinates, and since there are only a polynomial $\frac{\alpha n}{\varepsilon^d} \times 2kn \leq 2\alpha k\varepsilon^{-3}n^3$ number of discrete points, the running time of the dynamic program is also polynomial.

▶ **Lemma 11.** *The running time of the above dynamic program is* $(kn/\varepsilon)^{O(1/\varepsilon^3)}$.

Our proof for bounding our approximation factor is similar to the analysis of the PTAS for rectangles [35] and our QPTAS in Section 3. We describe here its main structure and highlight the key differences.

The solution computed by our DP corresponds to performing a sequence of trivial, add, and *line* operations, and recursing on the respective subproblems. It is sufficient to argue that there exists a sequence of these operations such that

- there exists a DP cell for each arising subproblem; in particular, the number of line segments in each subproblem is bounded by $3\varepsilon^{-3}$, and
- the total cost of the computed solution is bounded by $(1 + O(\varepsilon))\text{OPT}$.

We now describe this sequence. It is based on a hierarchical grid of vertices lines, shifted by a random offset $r \in \{0, \varepsilon^d, 2\varepsilon^d, \dots, \alpha\varepsilon^{-2}\}$ that we will fix later. For each level $j \in \mathbb{N}_0$, we define a grid line $\{r + t \cdot \alpha\varepsilon^{j-2}\} \times \mathbb{R}$ for each $t \in \mathbb{Z}$. Note that for all $j \leq d + 2$, grid lines of level j have discrete x-coordinates. We say that a line segment $\ell \in \text{OPT}$ is of *level* j if the length of ℓ is in $(\alpha\varepsilon^j, \alpha\varepsilon^{j-1}]$. We say that a line segment of some level j is *well-aligned* if its left and right endpoint lies on a grid line of level $j + 3$, and if the y-coordinate of both endpoints is discrete. We can extend each line segment $\ell \in \text{OPT}$ so that it becomes well-aligned, by increasing its length by at most a factor of $1 + O(\varepsilon)$.

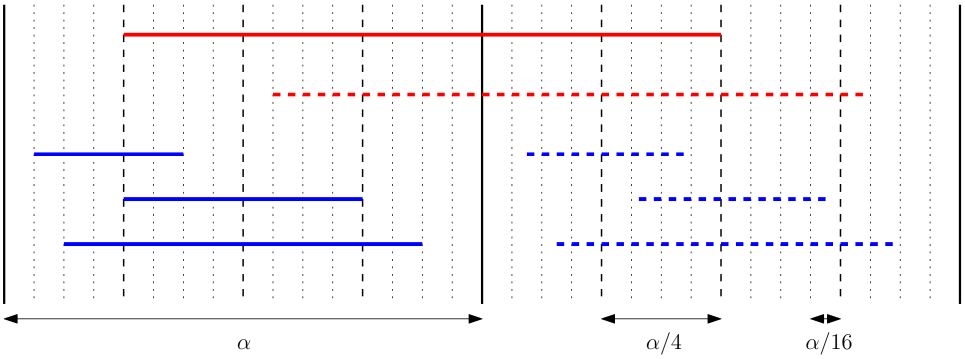


Figure 5 For $\varepsilon = 1/4$, the figure shows vertical grid lines of level $j = 2, 3, 4$ (solid, dashed and dotted lines respectively). Horizontal segments of level $j = 0, 1$ (red and blue respectively) are shown where the solid segments are well-aligned, and the dashed ones are not.

▶ **Lemma 12.** *For any value of our offset, by losing a factor of* $1 + O(\varepsilon)$ *in our approximation ratio, we can assume that each line segment* $\ell \in \text{OPT}$ *is well-aligned.*

Note that each horizontal segment $\ell \in \mathrm{OPT}$ satisfies that $\alpha\varepsilon/n < |\ell| \leq \alpha\varepsilon^{-1}$. By our choice of d we have $\varepsilon^{d-1} \leq \varepsilon/n < \varepsilon^{d-2}$ which implies $\alpha\varepsilon^{d-1} < |\ell| \leq \alpha\varepsilon^{-1}$. Since a segment is of level j if its length is in the range $(\alpha\varepsilon^{j}, \alpha\varepsilon^{j-1}]$, we can conclude that all segments in OPT belong to levels in the range $\{0, \ldots, d-1\}$. From this we can infer that any well-aligned horizontal segment is aligned to a vertical grid line of level at most $d+2$, which as we noted earlier has discrete x-coordinates.

In our sequence of operations, we first perform one *line operation* for each (vertical) grid line of level $j = 0$. This is similar to partitioning the instance into narrow strips as we did it in Lemma 5. However, now each strip has a width of $\alpha\varepsilon^{-2}$ instead of $w_{\max}(\mathcal{K})/\mu$. In our following operations, we add horizontal line segments to partition each vertical strip, similar to Section 3. Formally, we sort the segments from OPT of level $j = 0$ in increasing order of their y-coordinates, and pick every (ε^{-3})-th segment, and do an add operation along the (strip wide) line along it. This leads to a trivial operation immediately after that. Finally, we perform add operations for all line segments of level $j = 0$ in OPT. We call the above operations to be *operations of level* 0.

With the above operations for level $j = 0$ done, in increasing order of level $j = 1, 2, \ldots$ we do *operations of level j* similarly as follows:

- *line operations* on vertical grid lines of level j,
- any valid trivial operations (this step is not done for level 0),
- add, and trivial operations to divide the vertical strips into smaller subproblems,
- and finally the add operations on the segments from OPT of level j,

mimicking the recursive structure from the analysis of the QPTAS.

▶ **Lemma 13.** *The above sequence of operations always leads to valid DP subproblems.*

We wish to bound the cost of the above operations. Suppose that we perform a line operation with a vertical line ℓ and let $\mathcal{K}_\ell$ denote the k-shapes that ℓ intersects. Recall that for each *line operation*, we compute a solution that stabs all k-shapes in $\mathcal{K}_\ell$ (and in fact every rectangle in $\mathcal{R}_\ell$). Note that any horizontal line segment $\ell' \in \mathrm{OPT}$ of some level $j' \geq j$ stabs a k-shape in $\mathcal{K}_\ell$ only if the distance between ℓ and ℓ' is at most $\alpha\varepsilon^{j-1}$. Another key insight is that since the ratio between the widest and the narrowest part of any $K \in \mathcal{K}_\ell$ is $1/\delta$, the solution we compute is also an $O(k/\delta + \varepsilon)$-approximate solution. Using the above facts, we claim that if we choose our offset r uniformly at random from the range $\{0, \varepsilon^d, 2\varepsilon^2 d, \ldots, \alpha\varepsilon^{-2}\}$, then the overall cost of these *line operations* is only $O(\varepsilon) \cdot \mathrm{OPT}$. Further to bound the cost of the add operations, we note that each add operation is either done on a segment in OPT, or is an operation that created a subproblem. We will show that we can charge the latter operations to segments from OPT inside the subproblem thus created, whose total cost is at least ε^{-1} times the width of the subproblem. We refer to the full version [34] for a formal description of our analysis.

▶ **Lemma 14.** *There is a discrete value for the offset $r \in \{0, \varepsilon^d, 2\varepsilon^d, \ldots, \varepsilon^{-2}\}$ such that the described sequence of operations produces a solution of cost at most $(1 + O(\varepsilon))\mathrm{OPT}$.*

▶ **Theorem 15.** *For each constant $k \in \mathbb{N}$ there is a PTAS for the k-STABBING problem when each given k-shape consists of pieces of a constant range of widths.*

▶ Remark 16. In the proof of our result above, we used that the widths of the rectangles of each input k-shape are in a bounded range. Strictly speaking, we used only that the width "spanned" by each input k-shape is at most a constant factor larger than the width of the narrowest rectangle of the k-shape. Hence, the result will also hold for other polygons, like polyominoes with $O(1)$ number of cells, that are not k-shapes but that do satisfy the latter property.

5 General case

In this section, we study the general case of stabbing rectilinear polygons. Please refer to the full version [34] for the missing proofs, and details of the results in this section.

5.1 APX-hardness

In contrast to the cases studied in Sections 3 and 4, we show that the general case of the stabbing problem does *not* admit a $(1+\varepsilon)$-approximation algorithm, even for only slightly more general types of instances. Formally, we prove that stabbing is APX-hard, already if each input polygon is a 3-shape.

▶ **Theorem 17.** *The stabbing problem for 3-shapes is* APX*-hard.*

On the other hand, any 2-shape satisfies the hourglass property; hence, stabbing is unlikely to be APX-hard for this class of objects since we have a QPTAS for this case.

▶ **Proposition 18.** *Each 2-shape satisfies the hourglass property.*

In the remainder of this subsection, we prove Theorem 17. We give an L-reduction from the vertex cover problem to 3-Stabbing. Note that it is NP-hard to approximate vertex cover with a strictly better approximation factor than $\sqrt{2}$ [36]. We will obtain the same lower bound for stabbing.

Consider a given instance $G=(V,E)$ of vertex cover. Remember that in vertex cover, we are required to select a subset $S \subseteq V$ of smallest size such that for each $e \in E$ one of its end points is in S. We construct an instance of k-Stabbing corresponding to G as follows. Assume that $V=\{v_1,\dots,v_n\}$. For each $v_i \in V$ construct a 1×1 square s_i, such that they are all arranged in a column separated by 1 unit distance each (see Figure 6). Formally, for each $v_i \in V$ the top-left corner of the square s_i has the coordinates $(0,2i-1)$. Note that the squares $s_1,\dots,s_n$ do *not* belong to our input shapes, but they only help us to construct the latter. For each edge $\{v_i,v_j\}\in E$ we define a 3-shape $r_{i,j}$ as the union of the three rectangles s_i, $[0,n+1]\times[2i-1,2j-2]$ and s_j (see Figure 6).

Note that none of the resulting shapes satisfies the hourglass property, and also for neither of them the widths of its three rectangles are in a constant range. The width of the widest rectangle of each constructed 3-shape is greater than n, but there is always a feasible solution with cost n that simply stabs the square s_i for each vertex $v_i \in V$. Thus, in any given solution to the stabbing instance, we can assume w.l.o.g. that no 3-shape is stabbed across its widest rectangle.

▶ **Lemma 19.** *For each $\gamma \in \mathbb{N}$, the given instance of vertex cover has a solution of size γ if and only if the corresponding k-Stabbing instance has a solution of cost γ.*

Proof. We first show that if there is a vertex cover of size γ then there is a solution of cost γ to our instance of k-Stabbing. Given a solution $S=\{v_1,v_2,\dots,v_\gamma\}$ to the vertex cover instance, construct a solution to the stabbing instance as follows: for each $v_i \in S$, stab the corresponding s_i along its top edge by a segment of length one. Clearly the cost of this set of segments is γ. Now we notice that every k-shape $r_{i,j}$ corresponds to an edge $e(v_i,v_j)$ in the graph. Since this edge has been covered by one of its adjacent vertices $v_i \in S$, $r_{i,j}$ is also stabbed by the segment that stabs s_i. We know that every edge of the graph is covered by some vertex in S, and hence every k-shape in the instance is also stabbed in the solution we constructed.

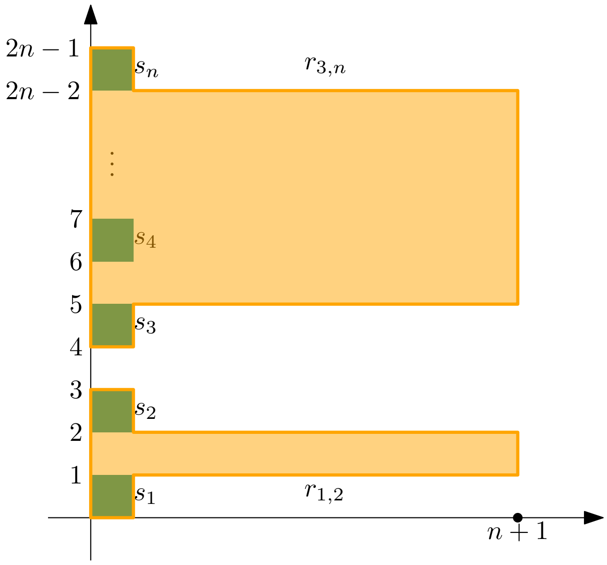


Figure 6 Construction of k-Stabbing instance in our reduction from vertex cover.

Next, we argue that a solution of cost γ to our instance of k-Stabbing yields a solution to vertex cover of size at most γ. Consider any solution to the stabbing instance of cost γ. We can assume that there are no segments of length greater than one in this solution, since any segment of length at least $n+1$, can be broken down into at most n segments of length 1 stabbing the same set of k-shapes, but along their bordering squares; and segments of length in the range $(1, n+1)$ can stab only one k-shape, and hence be shortened to length one. Further, segments in any solution can also not be of length less than one, since such a segment cannot stab any k-shape. Hence we conclude that all segments in the solution are of length one, and by extension that they stab any k-shape along one of its bordering squares.

Now we construct a vertex cover solution by picking the vertices v_i, that correspond to any square s_i that has been stabbed by the given (or modified as mentioned above) k-Stabbing solution. Note that every k-shape is stabbed by the given solution, and hence a vertex adjacent to every edge in the vertex cover instance has been picked by us. This shows that the selected set is in fact a valid vertex set, and is of size at most γ. ◀

This yields the proof of Theorem 17.

5.2 Set Cover hardness

We further show that k-Stabbing for arbitrary k-shapes cannot be approximated with a ratio of $o(\log n)$, unless $\mathsf{P} = \mathsf{NP}$. In fact, we show that the problem is as hard as general instances of Set Cover.

▶ **Theorem 20.** *The k-Stabbing problem does not admit an $o(\log n)$-approximation algorithm, unless* $\mathsf{P} = \mathsf{NP}$.

The proof of the above theorem is a generalization of the proof of Theorem 17, and its details can be found in the full version [34].

5.3 Approximation algorithm

We show that there is a polynomial time $O(k)$-approximation algorithm for k-Stabbing. Given an instance $\mathcal{K}$ of k-Stabbing with $n := |\mathcal{K}|$, we first show that we can restrict ourselves to a polynomial number of line segments which we construct using the following lemma.

▶ **Lemma 21.** *In polynomial time, we can construct a set $\mathcal{C}$ of line segments with the following properties:*

- *$\mathcal{C}$ contains $O((kn)^3)$ segments,*
- *$\mathcal{C}$ contains no redundant segments, where a segment is* redundant *if it stabs exactly the same k-shapes as another segment, or no k-shapes at all, and*
- *$\mathcal{K}$ admits an optimal solution using only the segments from $\mathcal{C}$.*

Using $\mathcal{C}$, we define a linear program that corresponds to $\mathcal{K}$.

$$\begin{aligned} \min \sum_{s\in\mathcal{C}} |s| \cdot z_s & & \\ \text{s.t.} \sum_{s\in\mathcal{C}:\ s \text{ stabs } K} z_s \geq 1 & \qquad \forall K \in \mathcal{K} & \quad (1) \\ z_s \geq 0 & \qquad \forall s \in \mathcal{C}. & \end{aligned}$$

If each k-shape $K \in \mathcal{K}$ is a rectangle, then it was shown by Chan et al. [9] that this LP has a constant integrality gap. We prove that for arbitrary k-shapes it has an integrality gap of $O(k)$, and we give a corresponding polynomial time rounding algorithm, in which we use the result by Chan et al. [9] as a black-box.

▶ **Theorem 22** ([9]). *If each k-shape $K \in \mathcal{K}$ is a rectangle, then there is a constant α such that for any solution z to LP (1), in polynomial time we can compute an integral solution to (1) whose cost is at most $\alpha \sum_{s\in\mathcal{F}} |s|\dot{z}_s$.*

Using Theorem 22, we construct now an $(\alpha \cdot k)$-approximation algorithm for arbitrary k-shapes. Suppose we are given an optimal solution z^* to the LP (1). We define a new solution $\tilde{z}$ by setting $\tilde{z}_s := k \cdot z^*_s$ for each segment $s \in \mathcal{F}$. Each k-shape $K \in \mathcal{K}$ is composed out of at most k rectangles. Thus, for each k-shape $K \in \mathcal{K}$ there is one of these rectangles R for which $\sum_{s\in\mathcal{F}:\ s \text{ stabs } R} z^*_s \geq 1/k$ and, therefore, $\sum_{s\in\mathcal{F}:\ s \text{ stabs } R} \tilde{z}_s \geq 1$. Let $\mathcal{R}$ denote the set of all these rectangles for all k-shapes in $\mathcal{K}$. We apply Theorem 22 on $\tilde{z}_s$ and $\mathcal{R}$ which yields a set of segments $\tilde{\mathcal{S}}$ whose cost is at most $\alpha \cdot \sum_{s\in\mathcal{F}} |s| \cdot \tilde{z}_s = \alpha k \cdot \sum_{s\in\mathcal{F}} |s| \cdot z^*_s \leq \alpha k \cdot \mathrm{OPT}$. Since $\tilde{\mathcal{S}}$ stabs $\mathcal{R}$, it also stabs $\mathcal{K}$. Hence, $\tilde{\mathcal{S}}$ yields an $O(k)$-approximation to our problem.

▶ **Theorem 23.** *There is a polynomial time $O(k)$-approximation algorithm for k-Stabbing.*

We remark that our algorithm extends also to the setting in which each given shape consists of at most k rectangles that are not necessarily connected, but such that still at least one of them needs to be stabbed.

References

1 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. ACM*, 66(4):29:1–29:40, 2019. doi:10.1145/3326122.

2 Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. *ACM Trans. Algorithms*, 18(4):40:1–40:37, 2022. doi:10.1145/3551639.

3 Boris Aronov, Esther Ezra, and Micha Sharir. Small-size ε-nets for axis-parallel rectangles and boxes. *SIAM J. Comput.*, 39(7):3248–3282, 2010. doi:10.1137/090762968.

4 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014. doi:10.1137/1.9781611973402.2.

5 Hervé Brönnimann and Michael T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discret. Comput. Geom.*, 14(4):463–479, 1995. doi:10.1007/BF02570718.

6 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Comput. Geom.*, 47(2):112–124, 2014. doi:10.1016/J.COMGEO.2012.04.001.

7 Timothy M. Chan, Elyot Grant, Jochen Könemann, and Malcolm Sharpe. Weighted capacitated, priority, and geometric set cover via improved quasi-uniform sampling. In *SODA*, pages 1576–1585, 2012. doi:10.1137/1.9781611973099.125.

8 Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. In *SODA*, pages 3496–3528, 2022. doi:10.1137/1.9781611977073.139.

9 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 *ISAAC*, pages 61:1–61:13, 2018. doi:10.4230/LIPICS.ISAAC.2018.61.

10 Kang-Tsung Chang. *Introduction to geographic information systems (4. ed.)*. McGraw-Hill, 2008.

11 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017. doi:10.1016/J.COSREV.2016.12.001.

12 Vasek Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979. doi:10.1287/MOOR.4.3.233.

13 Aparna Das, Krzysztof Fleszar, Stephen G. Kobourov, Joachim Spoerhase, Sankar Veeramoni, and Alexander Wolff. Approximating the generalized minimum manhattan network problem. *Algorithmica*, 80(4):1170–1190, 2018. doi:10.1007/S00453-017-0298-0.

14 Emily Dinan, Alice Nadeau, and Isaac Odegard. Folding concave polygons into convex polyhedra: The L-shape. *Rose-Hulman Undergraduate Mathematics Journal*, 16(1):13, 2015.

15 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633, 2014. doi:10.1145/2591796.2591884.

16 Friedrich Eisenbrand, Martina Gallato, Ola Svensson, and Moritz Venzin. A QPTAS for stabbing rectangles. *arXiv*, 2021. arXiv:2107.06571.

17 Adeel Farooq, Mustafa Habib, Abid Mahboob, Waqas Nazeer, and Shin Min Kang. Zagreb polynomials and redefined zagreb indices of dendrimers and polyomino chains. *Open Chemistry*, 17(1):1374–1381, 2019. doi:10.1515/chem-2019-0144.

18 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

19 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. *ACM Trans. Algorithms*, 17(4):33:1–33:67, 2021. doi:10.1145/3473713.

20 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple L-shapes, spirals, and more. In *SoCG*, pages 39:1–39:17, 2021. doi:10.4230/LIPICS.SOCG.2021.39.

21 Waldo Gálvez, Arindam Khan, Mathieu Mari, Tobias Mömke, Madhusudhan Reddy Pittu, and Andreas Wiese. A $(2+\epsilon)$-approximation algorithm for maximum independent set of rectangles. *arXiv*, 2021. arXiv:2106.00623.

22 M. R. Garey and David S. Johnson. The rectilinear steiner tree problem is NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977. doi:10.1137/0132071.

23 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979. doi:10.5555/574848.

24 Daya Ram Gaur, Toshihide Ibaraki, and Ramesh Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *J. Algorithms*, 43(1):138–152, 2002. doi:10.1006/JAGM.2002.1221.

25 Solomon W. Golomb. Tiling with polyominoes. *Journal of Combinatorial Theory*, 1(2):280–296, 1966. doi:10.1016/S0021-9800(66)80033-9.

26 Solomon W. Golomb. *Polyominoes: puzzles, patterns, problems, and packings*, volume 111. Princeton University Press, 1996. doi:10.2307/j.ctv10vm1sc.

27 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A $(5/3 + \epsilon)$-approximation for strip packing. *Comput. Geom.*, 47(2):248–267, 2014. doi:10.1016/J.COMGEO.2013.08.008.

28 Klaus Jansen, Arindam Khan, Marvin Lira, and K. V. N. Sreenivas. A PTAS for packing hypercubes into a knapsack. In *ICALP*, pages 78:1–78:20, 2022. doi:10.4230/LIPICS.ICALP.2022.78.

29 Iwan Jensen and Anthony J. Guttmann. Statistics of lattice animals (polyominoes) and polygons. *Journal of Physics A: Mathematical and General*, 33(29):L257, 2000. doi:10.1088/0305-4470/33/29/102.

30 Arindam Khan, Aditya Lonkar, Arnab Maiti, Amatya Sharma, and Andreas Wiese. Tight approximation algorithms for two-dimensional guillotine strip packing. In *ICALP*, pages 80:1–80:20, 2022. doi:10.4230/LIPICS.ICALP.2022.80.

31 Arindam Khan, Aditya Lonkar, Saladi Rahul, Aditya Subramanian, and Andreas Wiese. Online and dynamic algorithms for geometric set cover and hitting set. In *SoCG*, pages 46:1–46:17, 2023. doi:10.4230/LIPICS.SOCG.2023.46.

32 Arindam Khan, Arnab Maiti, Amatya Sharma, and Andreas Wiese. On guillotine separable packings for the two-dimensional geometric knapsack problem. In *SoCG*, pages 48:1–48:17, 2021. doi:10.4230/LIPICS.SOCG.2021.48.

33 Arindam Khan and Eklavya Sharma. Tight approximation algorithms for geometric bin packing with skewed items. *Algorithmica*, 85(9):2735–2778, 2023. doi:10.1007/S00453-023-01116-0.

34 Arindam Khan, Aditya Subramanian, Tobias Widmann, and Andreas Wiese. On approximation schemes for stabbing rectilinear polygons. *arXiv*, 2024. doi:10.48550/arXiv.2402.02412.

35 Arindam Khan, Aditya Subramanian, and Andreas Wiese. A PTAS for the horizontal rectangle stabbing problem. In *IPCO*, pages 361–374, 2022. doi:10.1007/978-3-031-06901-7_27.

36 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in grassmann graph have near-perfect expansion. In *FOCS*, pages 592–601, 2018. doi:10.1109/FOCS.2018.00062.

37 Sofia Kovaleva and Frits C. R. Spieksma. Approximation algorithms for rectangle stabbing and interval stabbing problems. *SIAM J. Discret. Math.*, 20(3):748–768, 2006. doi:10.1137/S089548010444273X.

38 Jens Lienig and Juergen Scheible. *Fundamentals of layout design for electronic circuits*. Springer, 2020. doi:10.1007/978-3-030-39284-0.

39 Mario Alberto López and Dinesh P. Mehta. Efficient decomposition of polygons into L-shapes with application to VLSI layouts. *ACM Trans. Design Autom. Electr. Syst.*, 1(3):371–395, 1996. doi:10.1145/234860.234865.

40 George Martin. *Polyominoes: A guide to puzzles and problems in tiling*. Cambridge University Press, 1991. doi:10.1080/00029890.1993.11990425.

41 Joseph S. B. Mitchell. Approximating maximum independent set for rectangles in the plane. In *FOCS*, pages 339–350, 2021. doi:10.1109/FOCS52979.2021.00042.

42 Nabil H. Mustafa, Rajiv Raman, and Saurabh Ray. Settling the apx-hardness status for geometric set cover. In *FOCS*, pages 541–550, 2014. doi:10.1109/FOCS.2014.64.

43 Thomas Ottmann, Eljas Soisalon-Soininen, and Derick Wood. On the definition and computation of rectilinear convex hulls. *Information Sciences*, 33(3):157–171, 1984. doi:10.1016/0020-0255(84)90025-2.

44 Helmut Pottmann, Andreas Asperl, Michael Hofer, Axel Kilian, and Daril Bentley. *Architectural geometry*, volume 724. Bentley Institute Press Exton, 2007. doi:10.1016/j.cag.2014.11.002.

45 Peter Shirley, Michael Ashikhmin, and Steve Marschner. *Fundamentals of computer graphics.* AK Peters/CRC Press, 2009. doi:10.5555/1628957.

46 Kasturi R. Varadarajan. Weighted geometric set cover via quasi-uniform sampling. In *STOC*, pages 641–648, 2010. doi:10.1145/1806689.1806777.

47 Chris Worman and J. Mark Keil. Polygon decomposition and the orthogonal art gallery problem. *Int. J. Comput. Geom. Appl.*, 17(02):105–138, 2007. doi:10.1142/S0218195907002264.

48 Bei Yu, Jhih-Rong Gao, and David Z. Pan. L-shape based layout fracturing for e-beam lithography. In *ASP-DAC*, pages 249–254, 2013. doi:10.1109/ASPDAC.2013.6509604.

Maximizing Phylogenetic Diversity Under Ecological Constraints: A Parameterized Complexity Study

Christian Komusiewicz
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Jannik Schestag
Institute of Computer Science, Friedrich Schiller University Jena, Germany

Abstract

In the NP-hard Optimizing Phylogenetic Diversity with Dependencies (PDD) problem, the input consists of a phylogenetic tree $\mathcal{T}$ over a set of taxa X, a food-web that describes the prey-predator relationships in X, and integers k and D. The task is to find a set S of k species that is viable in the food-web such that the subtree of $\mathcal{T}$ obtained by retaining only the vertices of S has total edge weight at least D. Herein, viable means that for every predator taxon of S, the set S contains at least one prey taxon.

We provide the first systematic analysis of PDD and its special case with star trees, s-PDD, from a parameterized complexity perspective. For solution-size related parameters, we show that PDD is fixed-parameter tractable (FPT) with respect to D and with respect to k plus the height of the phylogenetic tree. Moreover, we consider structural parameterizations of the food-web. For example, we show an FPT-algorithm for the parameter that measures the vertex deletion distance to graphs where every connected component is a complete graph. Finally, we show that s-PDD admits an FPT-algorithm for the treewidth of the food-web. This disproves, unless P = NP, a conjecture of Faller et al. [Annals of Combinatorics, 2011] who conjectured that s-PDD is NP-hard even when the food-web is a tree.

2012 ACM Subject Classification Applied computing → Computational biology; Theory of computation → Fixed parameter tractability

Keywords and phrases phylogenetic diversity, food-webs, structural parameterization, color-coding, dynamic programming

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.28

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2405.17314

1 Introduction

Human activity has greatly accelerated the rate at which biological species go extinct. The conservation of biological diversity is thus one of mankind's most urgent tasks. The inherently limited amount of resources that one may devote to this task, however, necessitates decisions on which conservation strategies to pursue. To support such decisions, one needs to incorporate quantitative information on the possible impact and the success likelihood of conservation strategies. In this context, one task is to compute an optimal conservation strategy in light of this information.

To find a conservation strategy with the best positive impact, one would ideally aim to maximize the functional diversity of the surviving taxa (species). However, measuring this diversity is hard or impossible in many scenarios [18]. As a result, maximizing phylogenetic diversity has become the standard, albeit imperfect, surrogate for maximizing functional diversity [11, 13, 18]. Informally, phylogenetic diversity measures the evolutionary distance of a set of taxa. In its most simple form, this measurement is based on an edge-weighted

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 28; pp. 28:1–28:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

phylogenetic tree $\mathcal{T}$ of the whole set of taxa X, and the phylogenetic diversity of a subset of taxa S is the sum of the weights of the edges of the subtree of $\mathcal{T}$ obtained by retaining only the taxa of S. Assuming equal protection costs for all taxa, the task is to find a set S of at most k taxa that achieves maximal phylogenetic diversity. This problem, called Maximize Phylogenetic Diversity [9], can be solved very efficiently by a greedy algorithm [9, 19, 23, 27].

Computing an optimal conservation strategy becomes much more difficult, however, when the success likelihood of a strategy is included in the model. One way to achieve this is to add concrete survival probabilities for protected taxa, leading in its most general form to the NP-hard Generalized Noah's Ark Problem [12, 16]. This problem formulation, however, still has a central drawback: It ignores that the survival of some taxa may also depend on the survival of other taxa. This aspect was first considered by Moulton et al. [21] in the Optimizing PD with Dependencies (PDD) problem. Here, the input additionally contains a directed acyclic graph $\mathcal{F}$ with vertex set X where an arc uv is present if the existence of taxa u provides all the necessary foundations for the existence of taxon v. In other words, $\mathcal{F}$ models ecological dependencies between taxa. Now, a taxon v may survive only if (i) it does not depend on other taxa at all, that is, it has no incoming arcs, or (ii) at least one taxon u survives such that $\mathcal{F}$ contains the arc uv. The most-wide spread interpretation of such ecological dependency networks are food-webs where the arc uv means that taxon v feeds on taxon u.[1] A subset of taxa X where every vertex fulfills (i) or (ii) is called *viable*. The task in PDD is to select a *viable* set of k taxa that achieves a maximal phylogenetic diversity. In this work, we study PDD from an algorithmic point of view.

Moulton et al. [21] showed that PDD can be solved by the greedy algorithm if the objective of maximizing phylogenetic diversity agrees with the viability constraint in a precise technical sense. Later, PDD was conjectured to be NP-hard [26]. This conjecture was confirmed by Faller et al. [10], who showed that PDD is NP-hard even if the food-web $\mathcal{F}$ is a tree. Further, Faller et al. [10] considered s-PDD, the special case where the phylogenetic tree is restricted to be a star, and showed that s-PDD is NP-hard even for food-webs which have a bipartite graph as underlying graph. Polynomial-time algorithms were provided for very restricted special cases, for example for PDD when the food-web is a *directed* tree [10]. Finally, for food-webs with constant depth, PDD was shown to admit a constant-factor approximation algorithm [8].

Our Contribution. As PDD is NP-hard even on very restricted instances [10], we turn to parameterized complexity in order to overcome this intractability. In particular, we aim to identify problem-specific parameters κ such that PDD can be solved in $f(\kappa) \cdot |I|^{\mathcal{O}(1)}$ time (these are called FPT-algorithms) or to show that such algorithms are unlikely, by showing W[1]-hardness for the parameter κ. Here, we consider the most natural parameters related to the solution, such as the solution size k and the threshold of diversity D, and parameters that describe the structure of the input food-web $\mathcal{F}$. We formally consider the decision problem, where we ask for the existence of a viable solution with diversity at least D, but our algorithms actually solve the optimization problem as well.

Our most important results are the following; Table 1 gives an overview. In Theorem 3.4, we prove that PDD is FPT when parameterized with the solution size k plus the height of the phylogenetic tree $\mathcal{T}$. This also implies that PDD is FPT with respect to D, the diversity threshold. We also consider the dual parameter $\overline{D}$, that is, the amount of diversity that is lost from $\mathcal{T}$ and show that PDD is W[1]-hard with respect to $\overline{D}$.

[1] We remark that previous works [21, 10] consider a reversed interpretation of the arcs. We define the order such that a source of the network also corresponds to a source of the ecosystem.

Table 1 Parameterized complexity results for PDD and s-PDD. Here, D.t. τ stands for Distance to τ – the number of vertices that need to be removed to obtain a graph from graph class τ.

Parameter	s-PDD		PDD	
Budget k	FPT	Thm. 3.2	XP (FPT is *open*)	Obs. 3.1
Diversity D	FPT	Thm. 4.1	FPT	Thm. 4.1
Species-loss $\overline{k}$	W[1]-hard, XP	Prop. 4.2, Obs. 3.1	W[1]-hard, XP	Prop. 4.2, Obs. 3.1
Diversity-loss $\overline{D}$	W[1]-hard, XP	Prop. 4.2, Obs. 3.1	W[1]-hard, XP	Prop. 4.2, Obs. 3.1
D.t. Cluster	FPT	Thm. 5.1	NP-h for 0	Thm. 5.3
D.t. Co-Cluster	FPT	Thm. 5.5	FPT	Thm. 5.5
Treewidth $\text{tw}_\mathcal{F}$	FPT	Thm. 5.6	NP-h for $\text{tw}_\mathcal{F} = 1$	[10]

We then consider the structure of the food-web. In particular, we consider the special case that each connected component of the food-web $\mathcal{F}$ is a complete digraph. As we will show, this case is structurally equivalent to the case that each connected component of $\mathcal{F}$ is a star with one source vertex. Thus, this case describes a particularly simple dependency structure, where taxa are either completely independent or have a common source. We show that PDD is NP-hard in this special case while s-PDD has an FPT-algorithm when parameterized by the vertex deletion distance to this special case. Our results thus yield structured classes of food-webs where the complexity of s-PDD and PDD strongly differ. Finally, we show that s-PDD is FPT with respect to the treewidth of the food-web and therefore can be solved in polynomial time if the food-web is a tree (Theorem 5.6). Our result disproves a conjecture of Faller et al. [10, Conjecture 4.2] that s-PDD is NP-hard even when the food-web is a tree (unless P = NP). Again, this result shows that s-PDD can be substantially easier than PDD on some structured classes of food-webs.

Structure of the Paper. In Section 2, we formally define Optimizing PD with Dependencies, give an overview of previous results and our contribution, and prove some simple initial results. In Section 3, we study s-PDD and PDD with respect to k, the solution size. In Section 4, we show that PDD is FPT with respect to the desired diversity but W[1]-hard for the acceptable loss of diversity. In Section 5, we consider parameterization by structural parameters of the food-web. Finally, in Section 6, we discuss future research ideas. The proofs of theorems, lemmas, and observations marked with ($\star$) are deferred to a full version of this work.

2 Preliminaries

2.1 Definitions

For a positive integer a, by $[a]$ we denote the set $\{1, 2, \dots, a\}$, and by $[a]_0$ the set $\{0\} \cup [a]$. We generalize functions $f : A \to B$, where B is a family of sets, to handle subsets $A' \subseteq A$ of the domain by defining $f(A') := \bigcup_{a \in A'} f(a)$.

For any graph G, we write $V(G)$ and $E(G)$, respectively, to denote the set of vertices and edges of G. We write $\{u, v\}$ for an undirected edge between u and v. For a directed edge from u to v, we write uv or (u, v) to increase readability. For a vertex set $V' \subseteq V(G)$, we let $G[V'] := (V', \{e \in E(G) \mid \text{both endpoints of } e \text{ are in } V'\})$ denote the subgraph of G induced by V'. Moreover, with $G - V' := G[V \setminus V']$ we denote the graph obtained from G by removing V' and its incident edges.

Phylogenetic Trees and Phylogenetic Diversity. A tree $T = (V, E)$ is a directed graph in which the *root* is the only vertex with an in-degree of zero, each other vertex has an in-degree of one. The root is denoted with ρ. The *leaves* of a tree are the vertices which have an out-degree of zero. We refer to the non-leaf vertices of a tree as *internal vertices*. A tree is a star if the root is the only internal vertex and all other vertices are leaves. For a given set X, a *phylogenetic X-tree* $\mathcal{T} = (V, E, \omega)$ is a tree $T = (V, E)$ with an *edge-weight* function $\omega : E \to \mathbb{N}_{>0}$ and a bijective labeling of the leaves with elements from X where all non-leaves in $\mathcal{T}$ have out-degree at least two. We write $\max_\omega$ to denote the biggest edge weight in $\mathcal{T}$. The set X is a set of *taxa* (species). Because of the bijective labeling, we interchangeably use the words taxon and leaf. In biological applications, the set X is a set of taxa, the internal vertices of $\mathcal{T}$ correspond to biological ancestors of these taxa and $\omega(e)$ describes the phylogenetic distance between the endpoints of e, as these endpoints correspond to distinct (possibly extinct) taxa, we may assume this distance is greater than zero.

For a directed edge $uv \in E$, we say u is the *parent* of v and v is a *child* of u. If there is a directed path from u to v in $\mathcal{T}$ (including when $u = v$), we say that u is an *ancestor* of v and v is a *descendant* of u. The sets of ancestors and descendants of v are denoted by $\operatorname{anc}(v)$ and $\operatorname{desc}(v)$, respectively. The set of descendants of v which are in X are *offspring* $\operatorname{off}(v)$ of a vertex v. For an edge $e = uv \in E$, we denote $\operatorname{off}(e) = \operatorname{off}(v)$.

For a tree $T = (V, E)$ and a vertex set $V' \subseteq V$, the *spanning tree of V'* is denoted by $T\langle V' \rangle$. The *subtree of T rooted at v* is $T\langle \{v\} \cup \operatorname{off}(v) \rangle$ and denoted by T_v, for some vertex $v \in V$. Given a set of taxa $A \subseteq X$, let $E_{\mathcal{T}}(A)$ denote the set of edges $e \in E$ with $\operatorname{off}(e) \cap A \neq \emptyset$. The *phylogenetic diversity* $PD_{\mathcal{T}}(A)$ of A is defined by

$$PD_{\mathcal{T}}(A) := \sum_{e \in E_{\mathcal{T}}(A)} \omega(e). \tag{1}$$

In other words, the phylogenetic diversity $PD_{\mathcal{T}}(A)$ of a set A of taxa is the sum of the weights of edges which have offspring in A.

Food-Webs. For a set of taxa X, a *food-web $\mathcal{F} = (X, E)$ on X* is a directed acyclic graph. If xy is an edge of E then x is *prey* of y and y is a *predator* of x. The set of prey and predators of x is denoted with $N_<(x)$ and $N_>(x)$, respectively. A taxon x with an empty set of prey is a *source* and $\operatorname{sources}(\mathcal{F})$ denotes the set of sources in the food-web $\mathcal{F}$. For a taxon $x \in X$ we define $X_{\leq x}$ to be the set of taxa X which can reach x in $\mathcal{F}$. Analogously, $X_{\geq x}$ is the set of taxa which x can reach in $\mathcal{F}$.

For a given food-web $\mathcal{F}$ and a set $Z \subseteq X$ of taxa, a set of taxa $A \subseteq Z$ is *Z-viable* if $\operatorname{sources}(\mathcal{F}[A]) \subseteq \operatorname{sources}(\mathcal{F}[Z])$. A set of taxa $A \subseteq X$ is *viable* if A is X-viable. In other words, a set $A \subseteq Z$ is Z-viable or viable if each source in $\mathcal{F}[A]$ is also a source in $\mathcal{F}[Z]$ or in $\mathcal{F}$, respectively.

Problem Definitions and Parameterizations. Our main problem is defined as follows.

Optimizing PD with Dependencies (PDD)
Input: A phylogenetic X-tree $\mathcal{T}$, a food-web $\mathcal{F}$ on X, and integers k and D.
Question: Is there a viable set $S \subseteq X$ such that $|S| \leq k$ and $PD_{\mathcal{T}}(S) \geq D$?

Additionally in Optimizing PD in Vertex-Weighted Food-Webs (s-PDD) we consider the special case of PDD in which the phylogenetic X-tree $\mathcal{T}$ is a star. Throughout the paper, we adopt the convention that n is the number of taxa $|X|$ and we let m denote the number of edges in the food-web $|E(\mathcal{F})|$. Observe that $\mathcal{T}$ has $\mathcal{O}(n)$ edges.

For an instance $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ of PDD, we define $\overline{D} := PD_{\mathcal{T}}(X) - D = \sum_{e \in E} \omega(e) - D$. Informally, $\overline{D}$ is the acceptable loss of diversity: If we save a set of taxa $A \subseteq X$ with $PD_{\mathcal{T}}(A) \geq D$, then the total amount of diversity we lose from $\mathcal{T}$ is at most $\overline{D}$. Similarly, we define $\overline{k} := |X| - k$. That is, $\overline{k}$ is the minimum number of species that need to become extinct.

Parameterized Complexity. Throughout this paper, we consider a number of parameterizations of PDD and s-PDD. For a detailed introduction to parameterized complexity refer to the standard monographs [5, 7]; we only give a brief overview here.

A parameterization of a problem Π associates with each input instance $\mathcal{I}$ of Π the value of a specific *parameter* κ. A parameterized problem Π is *fixed-parameter tractable* (FPT) with respect to some parameter κ if there exists an algorithm solving every instance $(\mathcal{I}, \kappa)$ of Π in time $f(\kappa) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$. A parameterized problem Π is *slice-wise polynomial* (XP) with respect to some parameter κ if there exists an algorithm solving every instance $(\mathcal{I}, \kappa)$ of Π in time $|\mathcal{I}|^{f(\kappa)}$. Here, in both cases, f is some computable function only depending on κ. Parameterized problems that are W[1]-*hard* are believed not to be FPT. We use the $\mathcal{O}^*$-notation which omits factors polynomial in the input size.

Color Coding. For an in-depth treatment of color coding, we refer the reader to [5, Chapter 5] and [1]. Here, we give some definitions which we use throughout the paper.

For integers n and k, an (n, k)*-perfect hash family* $\mathcal{H}$ is a family of functions $f : [n] \to [k]$ such that for every subset Z of $[n]$ of size k, some $f \in \mathcal{H}$ exists which is injective when restricted to Z. For any integers $n, k \geq 1$ an (n, k)-perfect hash family which contains $e^k k^{\mathcal{O}(\log k)} \cdot \log n$ functions can be constructed in time $e^k k^{\mathcal{O}(\log k)} \cdot n \log n$ [22, 5].

2.2 Preliminary Observations

We start with some observations and reduction rules which we use throughout the paper.

▶ **Observation 2.1.** *Let $\mathcal{F}$ be a food-web. A set $A \subseteq X$ is viable if and only if there are edges $E_A \subseteq E(\mathcal{F})$ such that every connected component in the graph (A, E_A) is a tree with root in* $\text{sources}(\mathcal{F})$*.*

Proof. If A is viable, then $\text{sources}(\mathcal{F}[A])$ is a subset of $\text{sources}(\mathcal{F})$. It follows that for each taxon $x \in A$, either x is a source in $\mathcal{F}$ or A contains a prey y of x. Conversely, if such a graph (A, E_A) exists then explicitly the sources of $\mathcal{F}[A]$ are a subset of $\text{sources}(\mathcal{F})$. ◀

▶ **Observation 2.2.** *Let $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ be a* yes*-instance of* PDD*. Then $k > |X|$ or a viable set $S \subseteq X$ with $PD_{\mathcal{T}}(S) \geq D$ exists which has size exactly k.*

Proof. Let S be a solution for $\mathcal{I}$. If S has a size of k, nothing remains to be shown. Otherwise, observe that $S \cup \{x\}$ is viable and $PD_{\mathcal{T}}(S \cup \{x\}) \geq PD_{\mathcal{T}}(S)$ for each taxon $x \in (N_{>}(S) \cup \text{sources}(\mathcal{F})) \setminus S$. Because $(N_{>}(S) \cup \text{sources}(\mathcal{F})) \setminus S$ is non-empty unless $S = X$, we conclude that $S \cup \{x\}$ is a solution and iteratively, there is a solution of size k. ◀

▶ **Observation 2.3.** *Let $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ be an instance of* PDD*. In $\mathcal{O}(|\mathcal{I}|^2)$ time one can compute an equivalent instance $\mathcal{I}' := (\mathcal{T}', \mathcal{F}', k', D')$ of* PDD *with only one source in $\mathcal{F}'$, $k' := k + 1$ and $D' \in \mathcal{O}(D)$.*

Proof. Let $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ be an instance of PDD. Add a new taxon $\star$ to $\mathcal{F}$ and add edges from $\star$ to each taxon x of sources($\mathcal{F}$) to obtain $\mathcal{F}'$. To obtain $\mathcal{T}'$, add $\star$ as a child to the root ρ of $\mathcal{T}$ and set $\omega'(\rho\star) = D + 1$ and $\omega'(e) = \omega(e)$ for each $e \in E(\mathcal{T})$. Finally, set $k' := k + 1$ and $D' := 2 \cdot D + 1$ and let $\mathcal{I}' = (\mathcal{T}', \mathcal{F}', k + 1, 2 \cdot D + 1)$. All steps can be performed in $\mathcal{O}(|\mathcal{I}|^2)$ time.

The equivalence of $\mathcal{I}$ and $\mathcal{I}'$ follows from the observation that $S \subseteq X$ is a solution for $\mathcal{I}$ if and only if $S \cup \{\star\}$ is a solution for $\mathcal{I}'$. ◀

▶ **Reduction Rule 1.** *Let $R \subseteq X$ be the set of taxa which have a distance of at least k to every source. Then, set $\mathcal{F}' := \mathcal{F} - R$ and $\mathcal{T}' := \mathcal{T} - R$.*

▶ **Lemma 2.4.** *Reduction Rule 1 is correct and can be applied exhaustively in $\mathcal{O}(n+m)$ time.*

Proof. By definition, each viable set of taxa which has a size of k is disjoint from R. Therefore, the set R is disjoint from every solution. The set R can be found in $\mathcal{O}(n + m)$ time by breadth-first search. This is also the total running time for the rule, since one application of the rule is exhaustive. ◀

After Reduction Rule 1 has been applied exhaustively, for any taxon $x \in X$ there is a viable set S_x of size at most k with $x \in S_x$. If some edge e has weight at least D, then for each taxon x which is an offspring of e, the set S_x is viable, has size at most k, and $PD_{\mathcal{T}}(S_x) \geq PD_{\mathcal{T}}(\{x\}) \geq D$. So, S_x is a solution. This implies the correctness of the following rule.

▶ **Reduction Rule 2.** *Apply Reduction Rule 1 exhaustively. If* $\max_\omega \geq D$ *return* `yes`*.*

We can also remove some edges which are not important for guaranteeing viability.

▶ **Reduction Rule 3.** *Given an instance $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ of* PDD *with $vw \in E(\mathcal{F})$. If v is not a source and $uw \in E(\mathcal{F})$ for each $u \in N_<(v)$, then remove vw from $E(\mathcal{F})$.*

▶ **Lemma 2.5.** *Reduction Rule 3 is correct and can be applied exhaustively in $\mathcal{O}(n^3)$ time.*

Proof. First, observe that if $\mathcal{I}'$ is a `yes`-instance, then so is $\mathcal{I}$ because every set that is viable in $\mathcal{I}'$ is viable in $\mathcal{I}$. Conversely, let $\mathcal{I}$ be a `yes`-instance of PDD with solution S. If $v \notin S$, then S is also a solution for instance $\mathcal{I}'$. If $v \in S$ then because S is viable in $\mathcal{F}$, some vertex u of $N_<(v)$ is in S. Consequently, S is also viable in $\mathcal{F} - vw$, as w still could be fed by u (if $w \in S$).

The running time can be seen as follows. For each pair of taxa v and w, we can check $N_<(v) \subseteq N_<(w)$ in $\mathcal{O}(n)$ time. Consequently, an exhaustive application of Reduction Rule 3 takes $\mathcal{O}(n^3)$ time. ◀

3 Parameterization by the Solution Size

In this section, we consider parameterization by the size of the solution k. First, we observe that PDD is XP when parameterized by k and $\overline{k}$. In Section 3.1 we show that s-PDD is FPT with respect to k. We generalize this result in Section 3.2 by showing that PDD is FPT when parameterized by $k + \text{height}_{\mathcal{T}}$. Recall that $\overline{k} := n - k$.

▶ **Observation 3.1.** PDD *can be solved in $\mathcal{O}(n^{k+2})$ and $\mathcal{O}(n^{\overline{k}+2})$ time.*

Proof. One may use the following brute-force algorithm: Iterate over the sets S of X of size k. Return `yes` if there is a viable set S with $PD_{\mathcal{T}}(S) \geq D$. Return `no` if there is no such set.

The correctness of the algorithm follows from Observation 2.2. Checking whether a set S is viable and has diversity of at least D can be done $\mathcal{O}(n^2)$ time. The running time bound now follows because there are $\binom{n}{k} = \binom{n}{n-k} = \binom{n}{\overline{k}}$ subsets of X of size k. ◀

3.1 s-PDD

We show that s-PDD is FPT when parameterized by the size of the solution k.

▶ **Theorem 3.2** ($\star$). *s-PDD can be solved in* $\mathcal{O}(2^{3.03k+o(k)} \cdot nm \cdot \log n)$ *time.*

The idea behind the algorithm is to color the taxa and require that a solution should contain at most one taxon of each color. Formally, we consider the following auxiliary problem. In k-Colored Optimizing PD in Vertex-Weighted Food-Webs (k-c-s-PDD), alongside the usual input $(\mathcal{T}, \mathcal{F}, k, D)$ of s-PDD, we are given a coloring $c : X \to [k]$ which assigns each taxon a *color* $c(x) \in [k]$. We ask whether there is a viable set $S \subseteq X$ of taxa such that $PD_{\mathcal{T}}(S) \geq D$, and $c(S)$ is *colorful*. A set $c(S)$ is colorful if c is injective on S. Observe that each colorful set S satisfies $|S| \leq k$. We first show how to solve k-c-s-PDD via dynamic programming. Then, applying the color-coding toolbox allows us to extend this result to the uncolored version.

▶ **Lemma 3.3.** *k-c-s-PDD can be solved in* $\mathcal{O}(3^k \cdot n \cdot m)$ *time.*

Proof.

Table definition. Let $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D, c)$ be an instance of k-c-s-PDD and by Observation 2.3 we assume that $\star \in X$ is the only source in $\mathcal{F}$.

Given $x \in X$, a set of colors $C \subseteq [k]$, and a set of taxa $X' \subseteq X$, we say that a set $S \subseteq X' \subseteq X$ is (C, X')-*feasible* if (i) $c(S) = C$, (ii) $c(S)$ is colorful, and (iii) S is X'-viable. We define a dynamic programming algorithm with tables DP and DP′. For $x \in X$, $C \subseteq [k]$ we want entry DP$[x, C]$ to store the maximum $PD_{\mathcal{T}}(S)$ of $(C, X_{\geq x})$-feasible sets S. Recall $X_{\geq x}$ is the set of taxa which x can reach in $\mathcal{F}$. If no $(C, X_{\geq x})$-feasible set $S \subseteq X'$ exists, we want DP$[x, C]$ to store $-\infty$. In other words, in DP$[x, C]$ we store the biggest phylogenetic diversity of a set S which is $X_{\geq x}$-viable and c bijectively maps S to C.

For any taxon x, let $y_1, \dots, y_q$ be an arbitrary but fixed order of $N_>(x)$. In the auxiliary table DP′, we want entry DP′$[x, p, C]$ for $p \in [q]$, and $C \subseteq [k]$ to store the maximum $PD_{\mathcal{T}}(S)$ of (C, X')-feasible sets $S \subseteq X'$, where $X' = \{x\} \cup X_{\geq y_1} \cup \dots \cup X_{\geq y_p}$. If no (C, X')-feasible set $S \subseteq X'$ exists, we want DP′$[x, p, C]$ to store $-\infty$.

Algorithm. As a base case for each $x \in X$ and $p \in [|N_>(x)|]$ let DP$[x, \emptyset]$ and DP$[x, p, \emptyset]$ store 0 and let DP$[x, C]$ store $-\infty$ if C is non-empty and $c(x) \notin C$. For each $x \in X$ with $N_>(x) = \emptyset$, we store $\omega(\rho x)$ in DP$[x, \{c(x)\}]$. Recall that ρx is an edge because $\mathcal{T}$ is a star.

Fix a taxon $x \in X$. For every $Z \subseteq C \setminus \{c(x)\}$, we set DP′$[x, 1, \{c(x)\} \cup Z] :=$ DP$[y_1, Z]$. To compute further values, once DP′$[x, q, Z]$ for each $q \in [p]$, and every $Z \subseteq C$ is computed, for $Z \subseteq C \setminus \{c(x)\}$ we use the recurrence

$$\mathrm{DP}'[x, p+1, \{c(x)\} \cup Z] := \max_{Z' \subseteq Z} \mathrm{DP}'[x, p, \{c(x)\} \cup Z \setminus Z'] + \mathrm{DP}[y_{p+1}, Z']. \tag{2}$$

Finally, we set DP$[x, C] :=$ DP′$[x, q, C]$ for every $C \subseteq [k]$.

We return `yes` if DP$[\star, C]$ stores at least d for some $C \subseteq [k]$. Otherwise, we return `no`.

Correctness. The base cases are correct. The tables are computed first for taxa further away from the source and with increasing size of C. Assume that for a fixed taxon x with predators $y_1, \dots, y_q$ and a fixed $p \in [q]$, the entries DP$[x', Z]$ and DP′$[x, p', Z]$ for each $x' \in N_>(x)$, for each $p' \in [p]$, and every $Z \subseteq [k]$ store the desired value. Fix a set $C \subseteq [k]$ with $c(x) \in C$. We show that if DP′$[x, p+1, C]$ stores d then there is a (C, X')-feasible set

$S \subseteq X' \cup X_{\geq y_{p+1}}$ for $X' := \{x\} \cup X_{\geq y_1} \cup \cdots \cup X_{\geq y_p}$ with $PD_{\mathcal{T}}(S) = d$. Afterward, we show that if $S \subseteq X' \cup X_{\geq y_{p+1}}$ with $PD_{\mathcal{T}}(S) = d$ is a (C, X')-feasible set then $\mathrm{DP}'[x, p+1, C]$ stores at least d.

If $\mathrm{DP}'[x, p+1, C] = d > 0$ then by Recurrence (2), there is a set $Z \subseteq C \setminus \{c(x)\}$ such that $\mathrm{DP}'[x, p, C \setminus Z] = d_x$ and $\mathrm{DP}[y_{p+1}, Z] = d_y$ with $d = d_x + d_y$. Therefore, there is a $(C \setminus Z, X')$-feasible set $S_x \subseteq X'$ with $PD_{\mathcal{T}}(S_x) = d_x$ and a $(Z, X_{\geq y_{p+1}})$-feasible set $S_y \subseteq X_{\geq y_{p+1}}$ with $PD_{\mathcal{T}}(S_y) = d_y$. Define $S := S_x \cup S_y$ and observe that $PD_{\mathcal{T}}(S) = d$. It remains to show that S is a $(C, X' \cup X_{\geq y_{p+1}})$-feasible set. First, observe that because $C \setminus Z$ and Z are disjoint, we conclude that $c(S)$ is colorful. Then, $c(S) = c(S_x) \cup c(S_y) = C \setminus Z \cup Z = C$ where the first equation holds because $c(S)$ is colorful. The taxa x and y_{p+1} are the only sources in $\mathcal{F}[X_{\geq x}]$ and $\mathcal{F}[X_{\geq y_{p+1}}]$, respectively. Therefore, x is in S_x and y_{p+1} is in S_y unless S_y is empty. If $S_y = \emptyset$ then $S = S_x$ and S is $X' \cup X_{\geq y_{p+1}}$-viable because S is X'-viable. Otherwise, if S_y is non-empty then because S_y is $X_{\geq y_{p+1}}$-viable, we conclude $\mathrm{sources}(\mathcal{F}[S_y]) = \{y_{p+1}\}$. As $x \in S$ and $y_{p+1} \in N_{>}(x)$ we conclude $\mathrm{sources}(\mathcal{F}[S]) = \{x\}$ and so S is $X' \cup X_{\geq y_{p+1}}$-viable. Therefore, S is a $(C, X' \cup X_{\geq y_{p+1}})$-feasible set.

Conversely, let $S \subseteq X' \cup X_{\geq y_{p+1}}$ be a non-empty $(C, X' \cup X_{\geq y_{p+1}})$-feasible set with $PD_{\mathcal{T}}(S) = d$. Observe that X' and $X_{\geq y_{p+1}}$ are not necessarily disjoint. We define S_y to be the set of taxa of $X_{\geq y_{p+1}}$ which are connected to y_{p+1} in $\mathcal{F}[X_{\geq y_{p+1}}]$. Further, define $Z := c(S_y)$ and define $S_x := S \setminus S_y$. As $c(S)$ is colorful especially $c(S_x)$ and $c(S_y)$ are colorful. Thus, S_y is a $(Z, X_{\geq y_{p+1}})$-feasible time. Further, $c(S_x) = C \setminus c(S_y) = C \setminus Z$. As $\mathrm{sources}(\mathcal{F}[S]) = \mathrm{sources}(\mathcal{F}[X' \cup X_{\geq y_{p+1}}]) = \{x\}$, we conclude $x \in S$. Because $\mathcal{F}$ is a DAG, x is not in $X_{\geq y_{p+1}}$ and so x is in S_x. Each vertex of S which can reach y_{p+1} in $\mathcal{F}[S]$ is in $F_{\geq y_{p+1}}$ and subsequently in S_y. Consequently, because S is $X' \cup X_{\geq y_{p+1}}$-viable we conclude $\mathrm{sources}(\mathcal{F}[S_x]) = \{x\}$. Thus, S_x is $(C \setminus Z, X')$-feasible. So, $\mathrm{DP}[y_{p+1}, Z] = PD_{\mathcal{T}}(S_x)$ and $\mathrm{DP}'[x, p, C \setminus Z] = PD_{\mathcal{T}}(S_y)$. Hence, $\mathrm{DP}'[x, p+1, C]$ stores at least $PD_{\mathcal{T}}(S)$.

Running time. The base cases can be checked in $\mathcal{O}(k)$ time. As each $c \in [k]$ in Recurrence (2) can either be in Z', in $\{c(x)\} \cup Z \setminus Z'$ or in $[k] \setminus (\{c(x)\} \cup Z)$, all entries of the tables can be computed in $\mathcal{O}(3^k \cdot n \cdot m)$ time. ◀

We defer the details of the proof of Theorem 3.2 to the long version since this is essentially standard application of color coding using a perfect hash family as defined in Section 2.1.

3.2 PDD on Trees with Bounded Height

Next, we generalize the result of the previous subsection and we show that PDD is FPT when parameterized with the size of the solution k plus $\mathrm{height}_{\mathcal{T}}$, the height of the phylogenetic tree. The algorithm uses color-coding, data reduction rules, and the enumeration of trees.

▶ **Theorem 3.4.** PDD *can be solved in* $\mathcal{O}^*(K^K \cdot 2^{3.028K + o(K)})$ *time where* $K := k \cdot \mathrm{height}_{\mathcal{T}}$.

To show Theorem 3.4, we use a subroutine for solving the following problem. We define a *pattern-tree* $\mathcal{T}_P = (V_P, E_P, c_P)$ to be a tree (V_P, E_P) with a vertex-coloring $c_P : V_P \to [k \cdot \mathrm{height}_{\mathcal{T}}]$. Recall that $\mathcal{T}\langle Y \rangle$ is the spanning tree of the vertices in Y. In Optimizing PD with Pattern-Dependencies (PDD-pattern), we are given alongside the usual input $(\mathcal{T}, \mathcal{F}, k, D)$ of PDD a pattern-tree $\mathcal{T}_P = (V_P, E_P, c_P)$, and a vertex-coloring $c : V(\mathcal{T}) \to [k \cdot \mathrm{height}_{\mathcal{T}}]$. We ask whether there is a viable set $S \subseteq X$ of taxa such that S has a size of at most k, $c(\mathcal{T}\langle S \cup \{\rho\}\rangle)$ is colorful, and $\mathcal{T}\langle S \cup \{\rho\}\rangle$ and $\mathcal{T}_P$ are *color-equal*. That is, there is an edge uv of $\mathcal{T}\langle S \cup \{\rho\}\rangle$ with $c(u) = c_u$ and $c(v) = c_v$ if and only if there is an edge $u'v'$ of $\mathcal{T}_P$ with $c(u') = c_u$ and $c(v') = c_v$. Informally, given a pattern-tree we want that it matches the colors of the spanning tree induced by the root and the solution.

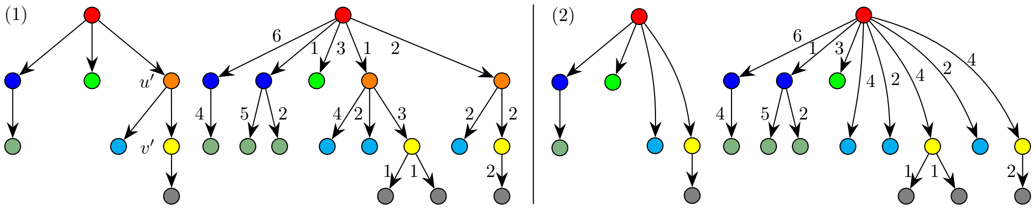


Figure 1 An example for Reduction Rule 6. (1) An instance of PDD-PATTERN (2) The instance after an application of Reduction Rule 6 to the marked vertices. In both instances, the pattern-tree is on the left and the phylogenetic tree is on the right.

Next we present reduction rules with which we can reduce the phylogenetic tree in an instance of PDD-PATTERN to be a star which subsequently can be solved with Theorem 3.2. Afterward, we show how to apply this knowledge to compute a solution for PDD.

▶ **Reduction Rule 4.** *Let uv be an edge of $\mathcal{T}$. If there is no edge $u'v' \in E_P$ with $c_P(u') = c(u)$ and $c_P(v') = c(v)$, then set $\mathcal{T}' := \mathcal{T} - \operatorname{desc}(v)$ and $\mathcal{F}' := \mathcal{F} - \operatorname{off}(v)$.*

▶ **Lemma 3.5.** *Reduction Rule 4 is correct and can be applied exhaustively in $\mathcal{O}(n^3)$ time.*

Proof. Assume $S \subseteq X$ is a solution of the instance of PDD-PATTERN. As there is no edge $u'v' \in E_P$ with $c_P(u') = c(u)$ and $c_P(v') = c(v)$ we conclude that $S \cap \operatorname{desc}(v) = \emptyset$ and so the reduction rule is safe.

The running time can be seen as follows. To check whether Reduction Rule 4 can be applied, we need to iterate over both $E(\mathcal{T})$ and E_P. Therefore, a single application can be executed in $\mathcal{O}(n^2)$ time. In each application of Reduction Rule 4 we remove at least one vertex so that an exhaustive application can be computed in $\mathcal{O}(n^3)$ time. ◀

▶ **Reduction Rule 5.** *Let $u'v'$ be an edge of $\mathcal{T}_P$. For each vertex $u \in V(\mathcal{T})$ with $c(u) = c_P(u')$ such that u has no child v with $c(v) = c_P(v')$, set $\mathcal{T}' := \mathcal{T} - \operatorname{desc}(v)$ and $\mathcal{F}' := \mathcal{F} - \operatorname{off}(v)$.*

▶ **Lemma 3.6.** *Reduction Rule 5 is correct and can be applied exhaustively in $\mathcal{O}(n^3)$ time.*

Proof. Let S be a solution for the instance of PDD-PATTERN. The spanning tree $\mathcal{T}\langle S \cup \{\rho\}\rangle$ contains exactly one vertex w with color $c(u)$. As $c(w) = c_P(u')$ we conclude that w has a child w' and $c(w') = c(v')$. Consequently, $w \neq u$ and $S \cap \operatorname{desc}(u) = \emptyset$.

As in Reduction Rule 4, we may apply the rule by iterating over the edges of $\mathcal{T}$ and $\mathcal{T}_P$. Each application either removes at least one vertex or concludes that the reduction rule is applied exhaustively. ◀

▶ **Reduction Rule 6.** *Apply Reduction Rules 4 and 5 exhaustively. Let ρ be the root of $\mathcal{T}$ and let ρ_P be the root of $\mathcal{T}_P$. Let v' be a grand-child of ρ_P and let u' be the parent of v'. Then, do the following.*

1. *For each vertex u of $\mathcal{T}$ with $c(u) = c_P(u')$ add edges ρv to $\mathcal{T}$ for every child v of u.*
2. *Set the weight of ρv to be $\omega(uv)$ if $c(v) \neq c_P(v')$ or $\omega(uv) + \omega(\rho u)$ if $c(v) = c_P(v')$.*
3. *Add edges $\rho_P w'$ to $\mathcal{T}_P$ for every child w' of u'.*
4. *Set $\mathcal{T}_P' := \mathcal{T}_P - u'$ and $\mathcal{T}' := \mathcal{T} - u$.*

Figure 1 depicts an application of Reduction Rule 6.

▶ **Lemma 3.7.** *Reduction Rule 6 is correct and can be applied exhaustively in $\mathcal{O}(n^3 m)$ time.*

Proof. Assume that $\mathcal{I}$ is a yes-instance of PDD-pattern with solution S. Because $\mathcal{T}\langle S \cup \{\rho\}\rangle$ and $\mathcal{T}_P$ are color-equal also $\mathcal{T}'\langle S \cup \{\rho\}\rangle$ and $\mathcal{T}'_P$ are color-equal. Let u^* and w_1 be the unique vertices in $\mathcal{T}\langle S \cup \{\rho\}\rangle$ with $c(u^*) = c_P(u')$ and $c(w_1) = c_P(v')$. Let $w_2, \dots, w_\ell$ be the other children of u^*. As $PD_{\mathcal{T}'}(S)$ is the sum of the weights of the edges of $\mathcal{T}'\langle S \cup \{\rho\}\rangle$ we conclude $PD_{\mathcal{T}'}(S) = PD_{\mathcal{T}}(S) - (\omega(\rho u^*) + \sum_{i=1}^{\ell} \omega(u^* w_i)) + \sum_{i=1}^{\ell} \omega'(\rho w_i)$. Since $\omega'(\rho w_1) = \omega(\rho u^*) + \omega(u^* w_1)$ and $\omega'(\rho w_i) = \omega(u^* w_i)$ for $i \in [\ell] \setminus \{1\}$, we conclude that $PD_{\mathcal{T}'}(S) = PD_{\mathcal{T}}(S) \geq D$. Therefore, S is a solution for $\mathcal{I}'$. The converse direction of the equivalence can be shown analogously.

It remains to bound the running time. For a given grand-child v' of ρ_P, one performs $\mathcal{O}(n)$ color-checks and adds $\mathcal{O}(n)$ edges. As the reduction rule can be applied at most $|\mathcal{T}_P| \in \mathcal{O}(n)$ times, an exhaustive application takes $\mathcal{O}(n^2)$ time. So, the predominant factor in the running time is the exhaustive application of the other reduction rules. ◀

With these reduction rules, we can reduce the phylogenetic tree of a given instance of PDD-pattern to only be a star and then solve PDD-pattern by applying Theorem 3.2.

▶ **Lemma 3.8.** PDD-pattern *can be solved in* $\mathcal{O}(3^k \cdot n \cdot m + n^3)$ *time.*

Proof. Let $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D, \mathcal{T}_P = (V_P, E_P, c_P), c)$ be a given instance of PDD-pattern. We use the following algorithm. If there is a vertex $v \in V_P$ and $c_P(v) \notin c(V(\mathcal{T}))$ then return no. If $c(\rho) \neq c_P(\rho_P)$ where ρ and ρ_P are the roots of $\mathcal{T}$ and $\mathcal{T}_P$ respectively, return no. Otherwise, apply Reduction Rule 6 exhaustively. Then, both $\mathcal{T}_P$ and $\mathcal{T}$ are stars. Return yes if and only if $(\mathcal{T}', \mathcal{F}', k, D, c)$ is a yes-instance of k-c-s-PDD.

For the correctness, first observe that if $\mathcal{T}_P$ contains a vertex v with $c_P(v) \notin c(V(\mathcal{T}))$, or if $c(\rho) \neq c_P(\rho_P)$, then $\mathcal{I}$ is a no-instance. For the remaining cases, the correctness follows from Lemma 3.7 and Lemma 3.3.

The running time can be seen as follows. By Reduction Rule 6 can be applied exhaustively in $\mathcal{O}(n^2 \cdot (n+m))$ time. By Lemma 3.3, the overall running time thus is $\mathcal{O}(3^k \cdot n \cdot m + n^3)$. ◀

To prove Theorem 3.4 we reduce from PDD to PDD-pattern and apply Lemma 3.8. For this, we use the fact that there are n^{n-2} labeled directed trees with n vertices [25] which can be enumerated in $\mathcal{O}(n^{n-2})$ time [2]. To solve instance $\mathcal{I}$ of PDD, we will check each of these trees as a pattern-tree for a given coloring of the phylogenetic tree. These colorings will be defined with a perfect hash family as defined in Section 2.1. Recall that $K = k \cdot \text{height}_{\mathcal{T}}$.

Proof of Theorem 3.4.

Algorithm. Let $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ be an instance of PDD. Let the vertices of $\mathcal{T}$ be $v_1, \dots, v_{|V(\mathcal{T})|}$. Iterate over $i \in [\min\{K, |V(\mathcal{T})|\}]$. Compute a $(|V(\mathcal{T})|, i)$-perfect hash family $\mathcal{H}_i$. Compute the set $\mathcal{P}_i$ of labeled directed trees with i vertices.

For every $\mathcal{T}_P = (V_P, E_P, c_P) \in \mathcal{P}_i$, proceed as follows. Assume that the labels of $\mathcal{T}_P$ are in $[i]$. For every $f \in \mathcal{H}_i$, first construct the coloring c_f such that $c_f(v_j) = f(j)$ for each $v_j \in V(\mathcal{T})$. Then, solve instance $\mathcal{I}_{\mathcal{T}_P,f} := (\mathcal{T}, \mathcal{F}, k, D, \mathcal{T}_P, c_f)$ of PDD-pattern using Lemma 3.8. Return yes if and only if $\mathcal{I}_{\mathcal{T}_P,f}$ is a yes-instance for some $f \in \mathcal{H}_i$ and some $\mathcal{T}_P \in \mathcal{P}_i$.

Correctness. Any solution of an instance $\mathcal{I}_{\mathcal{T}_P,f}$ of PDD-pattern clearly is a solution for $\mathcal{I}$.

Conversely, we show that if S is a solution for $\mathcal{I}$, then there are $\mathcal{T}_P$ and f such that $\mathcal{I}_{\mathcal{T}_P,f}$ is a yes-instance of PDD-pattern. So let S be a viable set of taxa with $|S| \leq k$ and $PD_{\mathcal{T}}(S) \geq D$. Let $V^* \subseteq V(\mathcal{T})$ be the set of vertices v that have offspring in S. It follows $|V^*| \leq \text{height}_{\mathcal{T}} \cdot |S| \leq K$. Then, there is a hash function $f \in \mathcal{H}_{V^*}$ mapping V^* bijectively to $[|V^*|]$. Consequently, $\mathcal{P}_{|V^*|}$ contains a tree $\mathcal{T}_P$ which is isomorphic to $\mathcal{T}[V^*]$ with labels c_f. Hence, $\mathcal{I}_{\mathcal{T}_P,f}$ is a yes-instance of PDD-pattern.

Running Time. For a fixed $i \in [K]$, the set $\mathcal{H}_i$ contains $e^i i^{\mathcal{O}(\log i)} \cdot \log n$ hash functions and the set $\mathcal{P}_i$ contains $\mathcal{O}(i^{i-2})$ labeled trees. Both sets can be computed in $\mathcal{O}(i^{i-2} \cdot n \log n)$ time.

Each instance $\mathcal{I}_{\mathcal{T}_P, f}$ of PDD-pattern is constructed in $\mathcal{O}(n)$ time and can be solved in $\mathcal{O}(3^k \cdot n^3)$ time. Thus, the overall running time is $\mathcal{O}(K \cdot e^K K^{K-2+\mathcal{O}(\log K)} \cdot 3^k \cdot n^3 \log n)$, which summarizes to $\mathcal{O}(K^K \cdot 2^{1.443K+1.585k+o(K)} \cdot n^3 \log n)$. ◀

4 Parameterization by Desired Diversity and Accepted Diversity Loss

In this section, we first consider parameterization with the diversity threshold D. For this parameter, we present an FPT algorithm for PDD. Afterward, we show that s-PDD is intractable with respect to $\overline{D}$, the acceptable loss of phylogenetic diversity. As the edge-weights are integers, we conclude that we can return yes if $k \geq D$ or if the height of the phylogenetic tree $\mathcal{T}$ is at least D, after Reduction Rule 1 has been applied exhaustively. Otherwise, $k + \text{height}_{\mathcal{T}} \in \mathcal{O}(D)$ and thus the FPT algorithm for $k + \text{height}_{\mathcal{T}}$ (Theorem 3.4) directly gives an FPT algorithm for PDD in that case.

Here, we present another algorithm with a faster running time. To obtain this algorithm, we subdivide edges of the phylogenetic tree according to their edge weights. We then use color coding on the vertices of the subdivided tree. Let us remark that this technique is closely related to an algorithm of Jones and Schestag [15] for another hard problem related to diversity maximization.

▶ **Theorem 4.1** ($\star$). PDD *can be solved in* $\mathcal{O}(2^{3.03(2D+k)+o(D)} \cdot nm + n^2)$ *time.*

In some instances, the diversity threshold D may be very large. Then, however, the acceptable loss of diversity $\overline{D} = PD_{\mathcal{T}}(X) - D$ could be small. Encouraged by this observation, recently, several problems in maximizing phylogenetic diversity have been studied with respect to the acceptable diversity loss [14, 15]. In this section, we show that, unfortunately, s-PDD is already $W[1]$-hard with respect to $\overline{D}$ even if the edge weights are at most two.

To show this, we reduce from Red-Blue Non-Blocker. Here, the input is an undirected bipartite graph G with vertex bipartition $V = V_r \cup V_b$ and an integer k. The question is whether there is a set $S \subseteq V_r$ of size at least k such that the neighborhood of $V_r \setminus S$ is V_b. Red-Blue Non-Blocker is W[1]-hard when parameterized by the solution size k [6].

▶ **Proposition 4.2.** s-PDD *is* $W[1]$*-hard with respect to* $\overline{D}$*, even if* $\max_\omega = 2$*.*

Proof.

Reduction. Let $\mathcal{I} := (G = (V = V_r \cup V_b, E), k)$ be an instance of Red-Blue Non-Blocker. We construct an instance $\mathcal{I}' = (\mathcal{T}, \mathcal{F}, k', D)$ of s-PDD as follows. Let $\mathcal{T}$ be a star with root $\rho \notin V$ and leaves V. In $\mathcal{T}$, an edge $e = \rho u$ has weight 1 if $u \in V_r$ and otherwise $\omega(e) = 2$, if $u \in V_b$. Define a food-web $\mathcal{F}$ with vertices V and for each edge $\{u, v\} \in E$, and every tuple of vertices $u \in V_b$, $v \in V_r$, add an edge uv to $\mathcal{F}$. Finally, set $k' := |V| - k$ and $D := 2 \cdot |V_b| + |V_r| - k$, or equivalently $\overline{k} = \overline{D} = k$.

Correctness. The reduction can be computed in polynomial time. We show that if $\mathcal{I}$ is a yes-instance of Red-Blue Non-Blocker then $\mathcal{I}'$ is a yes-instance of PDD. Afterward, we show the converse.

Assume that $\mathcal{I}$ is a yes-instance of Red-Blue Non-Blocker. Therefore, there is a set $S \subseteq V_r$ of size at least k such that $N_G(V_r \setminus S) = V_b$. (We assume $|S| = k$ as $N_G(V_r \setminus S) = V_b$ still holds if we shrink S.) We define $S' := V \setminus S$ and show that S' is a solution for $\mathcal{I}'$. The size of S' is $|V \setminus S| = |V| - |S| = k'$. Further, $PD_{\mathcal{T}}(S) = 2 \cdot |V_b| + |V_r \setminus S| = 2 \cdot |V_b| + |V_r| - k = D$. By definition, the vertices in V_r are sources. Further, because S is a solution for $\mathcal{I}$, each vertex of V_b has a neighbor in $V_r \setminus S$. So, S' is viable and $\mathcal{I}'$ is a yes-instance of s-PDD.

Conversely, let $S' \subseteq V$ be a solution for instance $\mathcal{I}'$ of s-PDD. Without loss of generality, S' contains r vertices from V_r and b vertices of V_b. Consequently, $|V| - k \geq |S'| = b + r$ and $2 \cdot |V_b| + |V_r| - k = D \leq PD_{\mathcal{T}}(S') = 2b + r$. We conclude $r \leq |V| - k - b$ and so $2b \geq 2 \cdot |V_b| + |V_r| - k - r \geq 2 \cdot |V_b| + |V_r| - k - (|V| - k - b) = |V_b| + b$. Therefore, $b = |V_b|$ and $V_b \subseteq S'$. Further, $r = |V_r| - k$. We define $S := V_r \setminus S'$ and conclude $|S| = |V_r| - r = k$. Because S' is viable, each vertex in V_b has a neighbor in $S' \setminus V_b$. Therefore, S is a solution for the yes-instance $\mathcal{I}$ of Red-Blue Non-Blocker. ◀

5 Structural Parameters of the Food-Web

Next, we study how the structure of the food-web affects the complexity of s-PDD and PDD. First, we consider parameterization with respect to the distance of the food-web to a cluster graph, denoted cvd. We show that PDD is NP-hard even if the food-web is a cluster graph but s-PDD is FPT when parameterized by cvd. Afterward, we show that PDD is FPT with respect to the distance to co-cluster and s-PDD is FPT with respect to the treewidth of the food-web, denoted by $\text{tw}_{\mathcal{F}}$.

5.1 Distance to Cluster Graphs

In this subsection, we consider the special case that given an instance of PDD or s-PDD, we need to remove few vertices from the undirected underlying graph of the food-web $\mathcal{F}$ to obtain a cluster graph. Here, a graph is a cluster graph if every connected component is a clique. We show that s-PDD is easy on graphs that are close to being a cluster graph. More precisely, in Theorem 5.1, we show that s-PDD is FPT with respect to the distance to cluster. Herein, for a graph $G = (V, E)$ the *distance to cluster* $\text{cvd}(G)$ is the smallest number d such that there exists a set $Y \subseteq V$ of size at most d such that $G - Y$ is a cluster graph. The FPT-algorithm shows in particular that s-PDD is tractable even on some very dense classes of food-webs. Afterward, we show that PDD is NP-hard on cluster graphs.

The FPT-algorithm exploits the following fact: If $\mathcal{F}$ is acyclic and its underlying graph is a cluster graph, then every clique in $\mathcal{F}$ has exactly one vertex $v_0 \in V(C)$ such that $v_0 \in N_<(v)$ for each $v \in V(C) \setminus \{v_0\}$. After applying Reduction Rule 3 exhaustively to a cluster graph, each connected component of the food-web is thus an out-star.

▶ **Theorem 5.1** ($\star$). s-PDD *can be solved in $\mathcal{O}(6^d \cdot n^2 \cdot m \cdot k^2)$ time, when we are given a set $Y \subseteq X$ of size d such that $\mathcal{F} - Y$ is a cluster graph.*

To prove Theorem 5.1, we first show how to solve the case where we want to save all taxa in Y.

▶ **Lemma 5.2.** *Given an instance $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ of* s-PDD *and a set $Y \subseteq X$ of size d such that $\mathcal{F} - Y$ is a cluster graph, we can compute whether there is a viable set $S \cup Y$ with $|S \cup Y| \leq k$ and $PD_{\mathcal{T}}(S \cup Y) \geq D$ in $\mathcal{O}(3^d \cdot n \cdot k^2)$ time.*

Proof. We provide a dynamic programming algorithm. Let $C_1, \dots, C_c$ be the connected components of $\mathcal{F} - Y$ and let $x_1^{(i)}, \dots, x_{|C_i|}^{(i)}$ be an order of C_i such that $(x_{j_1}^{(i)}, x_{j_2}^{(i)}) \in E(\mathcal{F})$ for $j_1 < j_2$.

Table definition. A set $S \subseteq X \setminus Y$ of taxa is (ℓ, Z)-*feasible*, if $|S| \leq \ell$ and $S \cup Z$ is viable. The dynamic programming algorithm has tables DP and DP_i for each $i \in [c]$. The entry $\mathrm{DP}[i, \ell, Z]$ for $i \in [c]$, $\ell \in [k]_0$, and $Z \subseteq Y$ stores the largest phylogenetic diversity $PD_{\mathcal{T}}(S)$ of an (ℓ, Z)-feasible set $S \subseteq C_1 \cup \dots \cup C_i$ and $-\infty$ if no such set S exists.

The table entries $\mathrm{DP}_i[j, b, \ell, Z]$ additionally have a dimension b with $b \in \{0, 1\}$. For $b = 0$, an entry $\mathrm{DP}_i[j, b, \ell, Z]$ with $b \in \{0, 1\}$ stores the largest phylogenetic diversity $PD_{\mathcal{T}}(S)$ of an (ℓ, Z)-feasible set $S \subseteq \{x_1^{(i)}, \dots, x_j^{(i)}\}$. For $b = 1$, additionally some vertex $v_{j'}^{(i)}$ with $j' < j$ needs to be contained in S.

Algorithm. Iterate over the edges of $\mathcal{F}$. For each edge $uv \in E(\mathcal{F})$ with $u, v \in Y$, remove all edges incoming at v, including uv, from $E(\mathcal{F})$. After this removal, v is a new source.

We initialize the base cases of DP_i by setting $\mathrm{DP}_i[j, 0, 0, Z] := 0$ for each $i \in [c]$, each $j \in [|C_i|]$, and every $Z \subseteq \mathrm{sources}(\mathcal{F})$. Moreover, $\mathrm{DP}_i[1, b, \ell, Z] := \omega(\rho v_1^{(i)})$ if $\ell \geq 1$ and $Z \subseteq N_>(v_1^{(i)}) \cup \mathrm{sources}(\mathcal{F})$; and $\mathrm{DP}_i[1, b, \ell, Z] := -\infty$, otherwise.

To compute further values for $j \in [|C_i| - 1]$, $b \in \{0, 1\}$, and $\ell \in [k]$ we use the recurrences

$$\mathrm{DP}_i[j+1, b, \ell, Z] = \max\{\mathrm{DP}_i[j, b, \ell, Z], \mathrm{DP}_i[j, b', \ell - 1, Z \setminus N_>(v_{j+1}^{(i)})] + \omega(\rho v_{j+1}^{(i)})\}, \quad (3)$$

where $b' = 0$ if there is an edge from a vertex in Y to $x_{j+1}^{(i)}$ and otherwise $b' = 1$.

Finally, we set $\mathrm{DP}[1, \ell, Z] := \mathrm{DP}_1[|C_1|, 0, \ell, Z]$ and compute further values with

$$\mathrm{DP}[i+1, \ell, Z] = \max_{Z' \subseteq Z, \ell' \in [\ell]_0} \mathrm{DP}[i, \ell', Z'] + \mathrm{DP}_{i+1}[|C_{i+1}|, 0, \ell - \ell', Z \setminus Z']. \quad (4)$$

There is a viable set $S \cup Y$ with $|S \cup Y| \leq k$ and $PD_{\mathcal{T}}(S \cup Y) \geq D$ if and only if $\mathrm{DP}[c, k - |Y|, Z] \geq D - PD_{\mathcal{T}}(Y)$.

Correctness. Assume that DP stores the intended values. Then, if $\mathrm{DP}[c, k - |Y|, Z] \geq D - PD_{\mathcal{T}}(Y)$, there is an (ℓ, Z)-feasible set $S \subseteq X \setminus Y$. First, this implies that $S \cup Y$ is viable. Moreover, since S has size at most $k - |Y|$, we obtain $|S \cup Y| \leq k$. Finally, because $\mathcal{T}$ is a star and S and Y are disjoint, $PD_{\mathcal{T}}(S) \geq D - PD_{\mathcal{T}}(Y)$ implies $PD_{\mathcal{T}}(S \cup Y) \geq D$. The converse direction can be shown analogously.

It remains to show that DP and DP_i store the right values. The base cases are correct. Towards the correctness of Recurrence (3), as an induction hypothesis, assume that $\mathrm{DP}_i[j, b, \ell, Z]$ stores the desired value for a fixed $j \in [|C_i| - 1]$, each $i \in [c]$, $b \in \{0, 1\}$, $\ell \in [k]_0$ and every $Z \subseteq Y$. Let $\mathrm{DP}_i[j + 1, b, \ell, Z]$ store d. We show that there is an (ℓ, Z)-feasible set $S \subseteq \{x_1^{(i)}, \dots, x_{j+1}^{(i)}\}$. By Recurrence (3), $\mathrm{DP}_i[j, b, \ell, Z]$ stores d or $\mathrm{DP}_i[j, 1, \ell - 1, Z \setminus N_>(v_{j+1}^{(i)})]$ stores $d - \omega(\rho v_{j+1}^{(i)})$. If $\mathrm{DP}_i[j, b, \ell, Z]$ stores d then there is an (ℓ, Z)-feasible set $S \subseteq \{x_1^{(i)}, \dots, x_j^{(i)}\} \subseteq \{x_1^{(i)}, \dots, x_{j+1}^{(i)}\}$. If $\mathrm{DP}_i[j, 1, \ell - 1, Z \setminus N_>(v_{j+1}^{(i)})]$ stores $d - \omega(\rho v_{j+1}^{(i)})$ then there is an $(\ell - 1, Z \setminus N_>(v_{j+1}^{(i)}))$-feasible set $S \subseteq \{x_1^{(i)}, \dots, x_j^{(i)}\}$ containing $x_1^{(i)}$ or $x_{j'}^{(i)} \in N_>(Y)$. Consequently, also $S \cup \{x_{j+1}^{(i)}\}$ is (ℓ, Z)-feasible.

Now, let $S \subseteq \{x_1^{(i)}, \dots, x_{j+1}^{(i)}\}$ be an (ℓ, Z)-feasible set. We show that $\mathrm{DP}_i[j + 1, b, \ell, Z]$ stores at least $PD_{\mathcal{T}}(S)$. If $S \subseteq \{x_1^{(i)}, \dots, x_j^{(i)}\}$ then we know from the induction hypothesis that $\mathrm{DP}_i[j, b, \ell, Z]$ stores $PD_{\mathcal{T}}(S)$ and then also $\mathrm{DP}_i[j+1, b, \ell, Z]$ stores $PD_{\mathcal{T}}(S)$. If $x_{j+1}^{(i)} \in S$, then S contains $x_1^{(i)}$ or some $x_{j'}^{(i)} \in N_>(Y)$. Define $S' := S \setminus \{x_{j+1}^{(i)}\}$. Then, $|S'| = \ell - 1$ and $S' \cup (Z \setminus N_>(x_{j+1}^{(i)}))$ is viable because S is (ℓ, Z)-feasible. Consequently, $\mathrm{DP}_i[j, 1, \ell - 1, Z \setminus N_>(x_{j+1}^{(i)})] \geq PD_{\mathcal{T}}(S') = PD_{\mathcal{T}}(S) - \omega(\rho x_{j+1}^{(i)})$. Therefore, $\mathrm{DP}_i[j + 1, b, \ell, Z] \geq PD_{\mathcal{T}}(S)$.

Now, we focus on the correctness of Recurrence (4). Let $\mathrm{DP}[i+1, \ell, Z]$ store d. We show that there is an (ℓ, Z)-feasible set $S \subseteq C_1 \cup \cdots \cup C_{i+1}$ with $PD_{\mathcal{T}}(S) = d$. Because $\mathrm{DP}[i{+}1, \ell, Z]$ stores d, by Recurrence (4), there are $Z' \subseteq Z$ and $\ell' \in [\ell]_0$ such that $\mathrm{DP}[i, \ell', Z'] = d_1$, $\mathrm{DP}_{i+1}[|C_{i+1}|, 0, \ell - \ell', Z \setminus Z'] = d_2$ and $d_1 + d_2 = d$. By the induction hypothesis, there is an (ℓ', Z')-feasible set $S_1 \subseteq C_1 \cup \cdots \cup C_i$ and an $(\ell - \ell', Z \setminus Z')$-feasible set $S_2 \subseteq C_{i+1}$ such that $PD_{\mathcal{T}}(S_1) = d_1$ and $PD_{\mathcal{T}}(S_2) = d_2$. Then, $S := S_1 \cup S_2$ holds $|S| \leq |S_1| + |S_2| \leq \ell' + (\ell - \ell') = \ell$. Further, because Y has no outgoing edges $Z' \subseteq N_{>}(S_1) \cup \mathrm{sources}(\mathcal{F})$ and $Z \setminus Z' \subseteq N_{>}(S_2) \cup \mathrm{sources}(\mathcal{F})$. Therefore, $Z \subseteq N_{>}(S) \cup \mathrm{sources}(\mathcal{F})$ and $S \cup Z$ is viable. We conclude that S is the desired set.

Let there be an (ℓ, Z)-feasible set $S \subseteq C_1 \cup \cdots \cup C_{i+1}$ with $PD_{\mathcal{T}}(S) = d$. We show that $\mathrm{DP}[i+1, \ell, Z]$ stores at least d. Define $S_1 := S \cap (C_1 \cup \cdots \cup C_i)$ and $Z' := N_{>}(S_1) \cap Z$. We conclude that $S_1 \cap Z'$ is viable. Then, S_1 is (ℓ', Z')-feasible, where $\ell' := |S_1|$. Define $S_2 := S \cap C_{i+1} = S \setminus S_1$. Because $S \cup Z$ is viable and Z does not have outgoing edges, we know that $Z \subseteq N_{>}(S) \cup \mathrm{sources}(\mathcal{F})$. So, $Z \setminus Z' \subseteq N_{>}(S_2) \cup \mathrm{sources}(\mathcal{F})$ and because $|S_2| = |S| - |S_1| = \ell - \ell'$ we conclude that S_2 is $(\ell - \ell', Z \setminus Z')$-feasible. Consequently, $\mathrm{DP}[i, \ell', Z'] \geq PD_{\mathcal{T}}(S_1)$ and $\mathrm{DP}_{i+1}[|C_{i+1}|, \ell{-}\ell', Z{\setminus}Z'] \geq PD_{\mathcal{T}}(S_2)$. Hence, $\mathrm{DP}[i{+}1, \ell, Z] \geq PD_{\mathcal{T}}(S_1) + PD_{\mathcal{T}}(S_2) = PD_{\mathcal{T}}(S)$ because $\mathcal{T}$ is a star.

Running time. The tables DP and DP_i for $i \in [c]$ have $\mathcal{O}(2^d \cdot n \cdot k)$ entries in total. Whether one of the base cases applies can be checked in linear time. We can compute the set $Z \setminus N_{>}(x)$ for any given $Z \subseteq Y$ and $x \in X$ in $\mathcal{O}(d^2)$ time. Therefore, the $\mathcal{O}(2^d \cdot n \cdot k)$ times we need to apply Recurrence (3) consume $\mathcal{O}(2^d d^2 \cdot n \cdot k)$ time in total. In Recurrence (4), each $x \in Y$ can be in Z', in $Z \setminus Z'$ or in $Y \setminus Z$ so that we can compute all the table entries of DP in $\mathcal{O}(3^d \cdot n \cdot k^2)$ which is also the overall running time. ◀

Now Theorem 5.1 can be shown by reducing the general case to the special case of Lemma 5.2 as follows: Iterate over the $\mathcal{O}(2^d)$ subsets of Y. For each subset $Z \subseteq Y$, compute whether there is a solution S for $\mathcal{I}$ with $S \cap Y = Z$; we defer the details of this branching to the long version.

Next, we show that, in contrast to s-PDD, PDD is NP-hard even when the food-web is restricted to be a cluster graph. We obtain this hardness by a reduction from Vertex Cover on cubic graphs. Here, we are given an undirected graph $G = (V, E)$ in which every vertex has degree *exactly* three and an integer k and ask whether a set $C \subseteq V$ of size at most k exists such that $u \in C$ or $v \in C$ for each $\{u, v\} \in E$. The set C is called a *vertex cover*. Vertex Cover remains NP-hard on cubic graphs [20].

▶ **Theorem 5.3.** PDD *is* NP*-hard even if the food-web is a cluster graph.*

Proof.

Reduction. Let (G, k) be an instance of Vertex Cover, where $G = (V, E)$ is cubic. We define an instance $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k', D)$ of PDD as follows. Let $\mathcal{T}$ have a root ρ. For each vertex $v \in V$, we add a child v of ρ. For each edge $e = \{u, v\} \in E$, we add a child e of ρ and two children $[u, e]$ and $[v, e]$ of e. Let N be a big integer. We set the weight of ρe to $N - 1$ for each edge e in E. All other edges of $\mathcal{T}$ have a weight of 1. Additionally, for each edge $e = \{u, v\} \in E$ we add edges $(u, [u, e])$ and $(v, [v, e])$ to $\mathcal{F}$. Finally, we set $k' := |E| + k$ and $D := N \cdot |E| + k$.

Correctness. The instance $\mathcal{I}$ of PDD is constructed in polynomial time. The sources of $\mathcal{F}$ are V. Let e_1, e_2, and e_3 be the edges incident with $v \in V(G)$. Each connected component in $\mathcal{F}$ contains four vertices, v, and $[v, e_i]$ for $i \in \{1, 2, 3\}$.

We show that (G, k) is a yes-instance of Vertex Cover if and only if $\mathcal{I}$ is a yes-instance of PDD. Let $C \subseteq V$ be a vertex cover of G of size at most k. If necessary, add vertices to C until $|C| = k$. For each edge $e \in E$, let v_e be an endpoint of e that is contained in C. Note that v_e exists since C is a vertex cover. We show that $S := C \cup \{[v_e, e] \mid e \in E\}$ is a solution for $\mathcal{I}$: The size of S is $|C| + |E| = k + |E|$. By definition, for each taxon $[v_e, e]$ we have $v_e \in C \subseteq S$, so S is viable. Further, as S contains a taxon $[v_e, e]$ for each edge $e \in E$, we conclude that $PD_{\mathcal{T}}(S) \geq N \cdot |E| + PD_{\mathcal{T}}(C) = N \cdot |E| + k = D$. Therefore, S is a solution.

Let S be a solution of instance $\mathcal{I}$ of PDD. Define $C := S \cap V(G)$ and define $S' := S \setminus C$. Because $PD_{\mathcal{T}}(S) \geq D$, we conclude that for each $e \in E$ at least one taxon $[u, e]$ with $u \in e$ is contained in S'. Thus, $|S'| \geq |E|$ and $|C| \leq k$. Because S is viable we conclude that $u \in C$ for each $[u, e] \in S'$. Hence, C is a vertex cover of size at most k of G. ◀

5.2 Distance to Co-cluster Graphs

In this section, we show that PDD is FPT with respect to the distance to co-cluster of the food-web. A graph is a co-cluster graph if its complement graph is a cluster graph. Herein, the complement graph is the graph obtained by replacing edges with non-edges and vice versa. In other words, a graph is a co-cluster graph if its vertex set can be partitioned into independent sets such that each pair of vertices from different independent sets is adjacent.

We define an auxiliary problem Hitting Set with Tree-Profits in which we are given a universe $\mathcal{U}$, a family of sets $\mathcal{W}$ over $\mathcal{U}$, a $\mathcal{U}$-tree $\mathcal{T}$, and integers k and D. We ask whether there is a set $S \subseteq \mathcal{U}$ of size at most k such that $PD_{\mathcal{T}}(S) \geq D$ and $S \cap W \neq \emptyset$ for each $W \in \mathcal{W}$. Solutions to this problem can be found with a dynamic programming algorithm over the tree, similar to the idea in [24]. The proof is therefore deferred to the long version.

▶ **Lemma 5.4** ($\star$). Hitting Set with Tree-Profits *can be solved in* $\mathcal{O}(3^{|\mathcal{W}|} \cdot n)$ *time.*

In the following we reduce from PDD to Hitting Set with Tree-Profits. Herein, we select a subset of the modulator Y to survive. Additionally, we select the first taxon x_i which survives in $X \setminus Y$. Because $\mathcal{F} - Y$ is a co-cluster graph, x_i is in a specific independent set $I \subseteq X$ and any taxon $X \setminus (I \cup Y)$ feed on x_i. Then, by selecting taxon $x_j \in X \setminus (I \cup Y)$, any other taxon in $X \setminus Y$ has some prey. Subsequently, a solution is found by Lemma 5.4.

▶ **Theorem 5.5.** PDD *can be solved in* $\mathcal{O}(6^d \cdot n^3)$ *time, when we are given a set* $Y \subseteq X$ *of size* d *such that* $\mathcal{F} - Y$ *is a co-cluster graph.*

Proof.

Algorithm. Given an instance $\mathcal{I} = (\mathcal{T}, \mathcal{F}, k, D)$ of PDD. Let $x_1, \dots, x_n$ be a topological ordering of X which is induced by $\mathcal{F}$. Iterate over the subsets Z of Y. Let P_Z be the sources of $\mathcal{F}$ in $X \setminus Y$ and let Q_Z be $N_>(Z) \setminus Y$, the taxa in $X \setminus Y$ which are being fed by Z. Further, define $R_Z := P_Z \cup Q_Z \subseteq X \setminus Y$. Iterate over the vertices $x_i \in R_Z$. Let x_i be from the independent set I of the co-cluster graph $\mathcal{F} - Y$. Iterate over the vertices $x_j \in X \setminus (Y \cup I)$.

For each set Z, and taxa x_i, x_j, with Lemma 5.4 we compute the optimal solution for the case that Z is the set of taxa of Y that survive while all taxa of $Y \setminus Z$ go extinct, x_i is the first taxon in $X \setminus Y$, and x_j the first taxon in $X \setminus (Y \cup I)$ to survive. (The special cases that only taxa from $I \cup Y$ or only from Y survive are omitted here.)

We define an instance $\mathcal{I}_{Z,i,j}$ of Hitting Set with Tree-Profits as follows. Let the universe $\mathcal{U}_{i,j}$ be the union of $\{x_{i+1}, \dots, x_{j-1}\} \cap I$ and $\{x_{j+1}, \dots, x_n\} \setminus Y$. For each taxon $x \in Z$ compute $N_<(x)$. If $x \notin \text{sources}(\mathcal{F})$ and $N_<(x) \cap (Z \cup \{x_i, x_j\}) = \emptyset$, then add $N_<(x) \setminus Y$ to the family of sets $\mathcal{W}_{Z,i,j}$. Contract edges $e \in E(\mathcal{T})$ with $\text{off}(e) \cap (Z \cup \{x_i, x_j\}) \neq \emptyset$ to obtain $\mathcal{T}_{Z,i,j}$. Finally, we define $k' := k - |Z| - 2$ and $D' := D - PD_{\mathcal{T}}(Z \cup \{x_i, x_j\})$.

Solve $\mathcal{I}_{Z,i,j}$. If $\mathcal{I}_{Z,i,j}$ is a `yes`-instance then return `yes`. Otherwise, continue with the iteration. If $\mathcal{I}_{Z,i,j}$ is a `no`-instance for every $Z \subseteq Y$, and each $i, j \in [n]$, then return `no`.

Correctness. We show that the algorithm returns `yes` if and only if $\mathcal{I}$ is a `yes`-instance.

First, assume the algorithm returns `yes`. Then, there is a set $Z \subseteq Y$, and there are taxa $x_i \in X \setminus Y$ and $x_j \in X \setminus (Y \cup V(I))$ such that $\mathcal{I}_{Z,i,j}$ is a `yes`-instance of Hitting Set with Tree-Profits. Here, I is the independent set such that $x_i \in V(I)$. Consequently, there is a set $S \subseteq \mathcal{U}_{i,j}$ of size at most $k-|Z|-2$ such that $PD_{\mathcal{T}_{Z,i,j}}(S) \geq D' = D - PD_{\mathcal{T}}(Z \cup \{x_i, x_j\})$ and $S \cap W \neq \emptyset$ for each $W \in \mathcal{W}_{Z,i,j}$. We show that $S^* := S \cup Z \cup \{x_i, x_j\}$ is a solution for instance $\mathcal{I}$ of PDD. Clearly, $|S^*| = |S| + |Z| + 2 \leq k$ and $PD_{\mathcal{T}}(S^*) = PD_{\mathcal{T}_{Z,i,j}}(S) + PD_{\mathcal{T}}(Z \cup \{x_i, x_j\}) \geq D$ as $\mathcal{T}_{Z,i,j}$ is the $Z \cup \{x_i, x_j\}$-contraction of $\mathcal{T}$. Further, by definition $x_i \in (\text{sources}(\mathcal{F}) \cup N_>(Z)) \setminus Y$. Because $\mathcal{F} - Y$ is a co-cluster graph and x_j is not in I, the independent set in which x_i is, we conclude that $x_j \in N_>(x_i)$. As $S \cap W \neq \emptyset$ for each $W \in \mathcal{W}_{Z,i,j}$, each taxon $x \in Z$ has a prey in $Z \cup \{x_i, x_j\}$ or in S so that $N_<(x) \cap S^* \neq \emptyset$. Therefore, S^* is viable and indeed a solution for $\mathcal{I}$.

Assume now that S is a solution for instance $\mathcal{I}$ of PDD. We define $Z := S \cap Y$ and let x_i and x_j be the taxa in $S \setminus Y$, respectively $S \setminus (Y \cup I)$, with the smallest index. As before, I is the independent set of x_i. We show that instance $\mathcal{I}_{Z,i,j}$ of Hitting Set with Tree-Profits has solution $S^* := S \setminus (Z \cup \{x_i, x_j\})$. Clearly, $|S^*| = |S| - |Z| - 2 \leq k'$ and by the definition of $\mathcal{T}_{Z,i,j}$ we also conclude $PD_{\mathcal{T}_{Z,i,j}}(S^*) \geq D'$. Let $M \in \mathcal{W}_{Z,i,j}$. By definition, there is a taxon $z \in Z$ with $M = N_<(z) \setminus Y$, and $z \notin \text{sources}(\mathcal{F})$, and $N_<(z) \cap (Z \cup \{x_i, x_j\}) = \emptyset$. Consequently, as S is viable, there is a taxon $x \in S \cap N_<(z)$ so that $S \cap M \neq \emptyset$. Hence, S^* is a solution of instance $\mathcal{I}_{Z,i,j}$ of Hitting Set with Tree-Profits.

Running time. For a given $Z \subseteq Y$, we can compute the topological order $x_1, \ldots, x_n$ and the set R_Y in $\mathcal{O}(n^2)$ time. The iterations over x_i and x_j take $\mathcal{O}(n^2)$ time. Observe, $|\mathcal{W}_{Z,i,j}| \leq |Z|$. By Lemma 5.4 checking whether $\mathcal{I}_{Z,i,j}$ is a `yes`-instance takes $\mathcal{O}(3^d n)$ time each. The overall running time is $\mathcal{O}(6^d \cdot n^3)$ time. ◀

5.3 Treewidth

Faller et al. [10] conjectured that s-PDD remains NP-hard even when the underlying graph of the food-web is a tree. We disprove this conjecture by showing that s-PDD can be solved in polynomial time on food-webs which are trees (assuming P≠NP). We even show a stronger result: s-PDD is FPT with respect to the treewidth of the food-web.

▶ **Theorem 5.6** (⋆). s-PDD *can be solved in* $\mathcal{O}(9^{\text{tw}_{\mathcal{F}}} \cdot nk)$ *time.*

To show Theorem 5.6, we define a dynamic programming algorithm over a tree-decomposition of $\mathcal{F}$. In each bag, we divide the taxa into three sets indicating that they a) are supposed to go extinct, b) will be saved but still need prey, c) or will be saved without restrictions. The algorithm is similar to the standard treewidth algorithm for Dominating Set [5].

6 Discussion

Several interesting questions remain open after our examination of PDD and s-PDD. Arguably the most relevant one is whether PDD is FPT with respect to k, the size of the solution. Also, it remains open whether PDD can be solved in polynomial time if each connected component in the food-web contains at most two vertices.

Clearly, further structural parameterizations can be considered. We only considered structural parameters which consider the underlying graph. But parameters which also consider the orientation of edges, such as the largest anti-chain, could give a better view on the structure of the food-web than parameters which only consider the underlying graph.

Liebermann et al. [17] introduced and analyzed weighted food-webs. Such a weighted model may provide a more realistic view of a species' effect on and interaction with other species [4]. Maximizing phylogenetic diversity with respect to a weighted food-web in which one potentially needs to save several prey per predator would be an interesting generalization for our work and has the special case in which one needs to save all prey for each predator.

Recent works consider the maximization of phylogenetic diversity in phylogenetic networks [29, 3, 14, 28] which may provide a more realistic evolutionary model of the considered species. It would be interesting to study these problems also under ecological constraints. Do the resulting problems become much harder than PDD? Finally, it has been reported that maximizing phylogenetic diversity is only marginally better than selecting a random set of species when it comes to maximizing the functional diversity of the surviving species [18]. The situation could be different, however, when ecological constraints are incorporated. Here, investigating the following two questions seems fruitful: First, do randomly selected viable species sets have a higher functional diversity than randomly selected species? Second, do viable sets with maximal phylogenetic diversity have a higher functional diversity than randomly selected viable sets?

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Terry Beyer and Sandra Mitchell Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9(4):706–712, 1980. doi:10.1137/0209055.

3 Magnus Bordewich, Charles Semple, and Kristina Wicke. On the complexity of optimising variants of phylogenetic diversity on phylogenetic networks. *Theoretical Computer Science*, 917:66–80, 2022. doi:10.1016/J.TCS.2022.03.012.

4 Alyssa R. Cirtwill, Giulio Valentino Dalla Riva, Marilia P. Gaiarsa, Malyon D. Bimler, E. Fernando Cagua, Camille Coux, and D. Matthias Dehling. A review of species role concepts in food webs. *Food Webs*, 16:e00093, 2018.

5 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.

6 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science*, 141(1-2):109–131, 1995. doi:10.1016/0304-3975(94)00097-3.

7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

8 Wolfgang Dvorák, Monika Henzinger, and David P. Williamson. Maximizing a submodular function with viability constraints. *Algorithmica*, 77(1):152–172, 2017. doi:10.1007/S00453-015-0066-Y.

9 Daniel P. Faith. Conservation evaluation and Phylogenetic Diversity. *Biological Conservation*, 61(1):1–10, 1992.

10 Beáta Faller, Charles Semple, and Dominic Welsh. Optimizing Phylogenetic Diversity with Ecological Constraints. *Annals of Combinatorics*, 15(2):255–266, 2011.

11 Pille Gerhold, James F Cahill Jr, Marten Winter, Igor V Bartish, and Andreas Prinzing. Phylogenetic patterns are not proxies of community assembly mechanisms (they are far better). *Functional Ecology*, 29(5):600–614, 2015.

12 Klaas Hartmann and Mike Steel. Maximizing phylogenetic diversity in biodiversity conservation: Greedy solutions to the Noah's Ark problem. *Systematic Biology*, 55(4):644–651, 2006.

13 Nick JB Isaac, Samuel T Turvey, Ben Collen, Carly Waterman, and Jonathan EM Baillie. Mammals on the edge: conservation priorities based on threat and phylogeny. *PloS one*, 2(3):e296, 2007.

14 Mark Jones and Jannik Schestag. How can we maximize phylogenetic diversity? Parameterized approaches for networks. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *LIPIcs*, pages 30:1–30:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.IPEC.2023.30.

15 Mark Jones and Jannik Schestag. Maximizing Phylogenetic Diversity under Time Pressure: Planning with Extinctions Ahead. *arXiv preprint*, 2024. doi:10.48550/arXiv.2403.14217.

16 Christian Komusiewicz and Jannik Schestag. A Multivariate Complexity Analysis of the Generalized Noah's Ark Problem. In *Proceedings of the 19th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW '23)*, volume 13 of *AIRO*, pages 109–121. Springer, 2023.

17 Erez Lieberman, Christoph Hauert, and Martin A. Nowak. Evolutionary dynamics on graphs. *Nature*, 433(7023):312–316, 2005.

18 Florent Mazel, Matthew W Pennell, Marc W Cadotte, Sandra Diaz, Giulio Valentino Dalla Riva, Richard Grenyer, Fabien Leprieur, Arne O Mooers, David Mouillot, Caroline M Tucker, et al. Prioritizing phylogenetic diversity captures functional diversity unreliably. *Nature Communications*, 9(1):2888, 2018.

19 Bui Quang Minh, Steffen Klaere, and Arndt von Haeseler. Phylogenetic Diversity within Seconds. *Systematic Biology*, 55(5):769–773, October 2006.

20 Bojan Mohar. Face covers and the genus problem for apex graphs. *Journal of Combinatorial Theory, Series B*, 82(1):102–117, 2001. doi:10.1006/JCTB.2000.2026.

21 Vincent Moulton, Charles Semple, and Mike Steel. Optimizing phylogenetic diversity under constraints. *Journal of Theoretical Biology*, 246(1):186–194, 2007.

22 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS '95)*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.

23 Fabio Pardi and Nick Goldman. Species Choice for Comparative Genomics: Being Greedy Works. *PLoS Genetics*, 1, 2005.

24 Fabio Pardi and Nick Goldman. Resource-aware taxon selection for maximizing phylogenetic diversity. *Systematic Biology*, 56(3):431–444, 2007.

25 Peter W Shor. A new proof of Cayley's formula for counting labeled trees. *Journal of Combinatorial Theory, Series A*, 71(1):154–158, 1995.

26 Andreas Spillner, Binh T. Nguyen, and Vincent Moulton. Computing phylogenetic diversity for split systems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(2):235–244, 2008. doi:10.1109/TCBB.2007.70260.

27 Mike Steel. Phylogenetic Diversity and the greedy algorithm. *Systematic Biology*, 54(4):527–529, 2005.

28 Leo van Iersel, Mark Jones, Jannik Schestag, Celine Scornavacca, and Mathias Weller. Maximizing network phylogenetic diversity. *arXiv preprint*, 2024. doi:10.48550/arXiv.2405.01091.

29 Kristina Wicke and Mareike Fischer. Phylogenetic Diversity and biodiversity indices on Phylogenetic Networks. *Mathematical Biosciences*, 298:80–90, 2018.

Matchings in Low-Arboricity Graphs in the Dynamic Graph Stream Model

Christian Konrad
University of Bristol, UK

Andrew McGregor
University of Massachusetts Amherst, MA, USA

Rik Sengupta
IBM Research, Cambridge, MA, USA
University of Massachusetts Amherst, MA, USA

Cuong Than
University of Massachusetts Amherst, MA, USA

Abstract

We consider the problem of estimating the size of a maximum matching in low-arboricity graphs in the dynamic graph stream model. In this setting, an algorithm with limited memory makes multiple passes over a stream of edge insertions and deletions, resulting in a low-arboricity graph. Let n be the number of vertices of the input graph, and α be its arboricity. We give the following results.

1. As our main result, we give a three-pass streaming algorithm that produces an $(\alpha+2)(1+\epsilon)$-approximation and uses space $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{1/2} \cdot \log n)$. This result should be contrasted with the $\Omega(\alpha^{-5/2} \cdot n^{1/2})$ space lower bound established by [Assadi et al., SODA'17] for one-pass algorithms, showing that, for graphs of constant arboricity, the one-pass space lower bound can be achieved in three passes (up to poly-logarithmic factors). Furthermore, we obtain a two-pass algorithm that uses space $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{3/5} \cdot \log n)$.
2. We also give a $(1+\epsilon)$-approximation multi-pass algorithm, where the space used is parameterized by an upper bound on the size of a largest matching. For example, using $O(\log\log n)$ passes, the space required is $O(\epsilon^{-1} \cdot \alpha^2 \cdot k \cdot \log n)$, where k denotes an upper bound on the size of a largest matching.

Finally, we define a notion of arboricity in the context of matrices. This is a natural measure of the sparsity of a matrix that is more nuanced than simply bounding the total number of nonzero entries, but less restrictive than bounding the number of nonzero entries in each row and column. For such matrices, we exploit our results on estimating matching size to present upper bounds for the problem of rank estimation in the dynamic data stream model.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Data Streams, Graph Matching, Graph Arboricity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.29

Funding *Christian Konrad*: Supported by EPSRC New Investigator Award EP/V010611/1.
Andrew McGregor: Funded in part by NSF Award CCF 1934846.
Rik Sengupta: Funded in part by NSF Award CCF 1934846.
Cuong Than: Funded by NSF CAREER Award CCF 2237288, NSF Award CCF 2121952, and a Google Research Scholar Award.

Acknowledgements We wish to thank Cameron Musco, Hung Le, Rajarshi Bhattacharjee, and David Woodruff for a lot of preliminary discussions about this work.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 29; pp. 29:1–29:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Streaming algorithms for graph problems have been studied for more than 25 years [25]. In this setting, an algorithm performs one or more passes over the input graph and produces a solution using as little space as possible.

Much of the development of the literature on graph streams has been driven by the study of the Maximum Matching problem (see, e.g., [2–6, 10–12, 17–19, 22–24, 28–34, 36, 41] for a non-comprehensive list of recent and early works). This problem was first addressed by Feigenbaum et al. [21], and a large number of algorithms and impossibility results that cover various aspects of the problem are known today, such as one-pass/multi-pass algorithms, adversarial/random order streams, insertion-only/insertion-deletion streams, and dense/sparse input graphs.

In this paper, we consider the size estimating variant of the problem, which we denote by Matching Size Estimation (MSE). This is in contrast to the much better understood objective of outputting the actual edges of a large matching. We address MSE in *dynamic* or *insertion-deletion* streams, i.e., streams consisting of a sequence of edge insertions and deletions. Our focus lies on sparse graphs as parameterized by the *arboricity* of the input graph. The arboricity of a graph is the smallest[1] integer α such that the edges of the graph can be partitioned into α forests. Nash-Williams [39] showed that an equivalent definition of the arboricity is $\alpha = \max_{S \subseteq V} |e(S)|/(|S|-1)$ where $e(S)$ is the number of edges in the induced subgraph $G[S]$.

Chitnis et al. [14] were the first to study MSE in graphs of bounded arboricity in the dynamic graph stream setting. They showed that there is a one-pass $\tilde{O}(\alpha \cdot n^{4/5})$-space algorithm[2] with approximation factor $O(\alpha)$. Cormode et al. [15] subsequently gave an $\tilde{O}(\alpha^2)$-approximation algorithm using space $\tilde{O}(\alpha^{10/3} \cdot n^{2/3})$, albeit under the restriction that the length of the input stream is $O(\alpha \cdot n)$. On the lower bound side, Assadi et al. [7] showed that computing an α-approximation in a single pass requires space $\Omega(n^{1/2} \cdot \alpha^{-5/2})$. The problem is thus wide open, even in the one-pass setting, and even for constant-arboricity graphs.

1.1 Our Results

In this paper, we give the first multi-pass algorithms for MSE in the dynamic graph stream setting, for graphs of arboricity α. We assume throughout that n, m, and α are known in advance. All our algorithms succeed with high probability, i.e., they output the correct matching size with probability $1 - 1/\text{poly}(n)$. We observe at this juncture that none of our algorithms require the assumption that the input stream length is bounded. We reiterate that the one-pass algorithm by Cormode et al. [15], which uses space $O(\alpha^{10/3} \cdot n^{2/3})$, requires this assumption.

Our main result is a three-pass $O(\alpha)$-approximation algorithm that uses roughly $\sqrt{n}$ space[3].

▶ **Theorem 1.** *There exists a three-pass algorithm using $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{1/2} \cdot \log^3 n)$ space that returns an $(\alpha + 2)(1 + \epsilon)$-approximation for* MSE *with high probability.*

[1] We shall abuse terminology slightly and say the arboricity of a graph G is α as long as the smallest integer is at most α.

[2] We write $\tilde{O}(.)$ to mean $O(.)$ with poly-log dependencies on n suppressed.

[3] Henceforth, we say specify the space use of the algorithms in terms of the number of words of memory where a word may store $O(\log n)$ bits.

This result should be contrasted with the "$\sqrt{n}$-barrier" result established by Assadi et al. [7], who showed that one-pass α-approximation algorithms for MSE in dynamic graph streams on graphs of arboricity α require space $\Omega(\sqrt{n}/\alpha^{2.5})$. While our algorithm uses three passes, and, consequently, the lower bound from [7] does not apply in this setting, we nevertheless show that the $\sqrt{n}$-barrier can be achieved at the expense of just two additional passes. Interestingly, we note that no multi-pass $O(\alpha)$-approximation dynamic streaming algorithms are known for MSE that break the $\sqrt{n}$-barrier, even if significantly more passes are allowed.

Next, we also give a new two-pass algorithm that requires less space than the best one-pass algorithms known (e.g., [15]).

▶ **Theorem 2.** *There exists a two-pass algorithm using $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{3/5} \cdot \log^3 n)$ space that returns an $(\alpha+2)(1+\epsilon)$-approximation for* MSE *with high probability.*

We also show a $(1+\epsilon)$-approximation multi-pass algorithm, in the case when the maximum matching size is bounded by a given parameter k.

▶ **Theorem 3.** *If the maximum matching size is upper bounded by k, there exists a $O(\epsilon^{-1} \cdot \alpha^2 \cdot n^{1/(2^p-1)} \cdot k^{1-1/(2^p-1)} \cdot \log n)$ space, p-pass dynamic graph streaming algorithm that returns a $(1+\epsilon)$-approximation for* MSE *with high probability. In particular, there exists a $O(\log\log n)$-pass algorithm that uses space $O(\epsilon^{-1} \cdot \alpha^2 \cdot k \cdot \log n)$.*

This result is similar in spirit to a result by Chitnis et al. [14], who showed that, in general graphs, a matching of size k (if there is one) can be computed in the one-pass dynamic setting using space $\tilde{O}(k^2)$; this result can also be obtained from the algorithm given by Assadi et al. [8]. Phrased differently, given an upper bound k on the size of a largest matching, we can compute one using space $\tilde{O}(k^2)$.

Lastly, as a more conceptual contribution, we introduce the notion of *low-arboricity* matrices. We say that a matrix A has arboricity α if every $t \times t$ submatrix of A has at most $\alpha \cdot t$ nonzero entries. This generalizes many natural subclasses of sparse matrices (including the adjacency matrices of low-arboricity graphs). We show that, given such a matrix A, we can associate a bipartite graph G_A of arboricity at most α with A, such that the rank of A is within an α-factor of $\mu(G_A)$, where $\mu(G_A)$ denotes the size of a largest matching in G_A. Hence, using any β-approximation algorithm for estimating the size of a maximum matching in graphs of arboricity α (where β might depend on α), we obtain an $(\alpha \cdot \beta)$-approximation to its rank. In particular, our two-pass and three-pass $O(\alpha)$-approximation algorithms immediately yield $O(\alpha^2)$-approximation algorithms for rank approximation in matrices of arboricity α.

1.2 Our Techniques

We will first discuss the ideas behind our $(1+\epsilon)$-approximation algorithm and our two-pass algorithm. Our three-pass algorithm, which constitutes our main result, combines ideas from these two algorithms.

$(1+\epsilon)$-approximation Algorithm. Our $(1+\epsilon)$-approximation algorithm works by iteratively identifying all "high" degree vertices with Count-Sketch. In the ith pass (for $i \leq p-1$) we identify a set of vertices V_i in the induced subgraph $G[V - V_1 - \ldots - V_{i-1}]$ using estimates of their degrees. Removing the high-degree vertices discovered in previous passes and exploiting properties of the degree sequence of low-arboricity graphs enable us to get increasingly accurate estimates of the degrees in the remaining graph. In the final pass, we collect all edges in $G[V - V_1 - \ldots - V_{p-1}]$, along with a few edges incident to every vertex in $V_1 \cup \ldots \cup V_{p-1}$. We are then able to argue that, by carefully setting the parameters of the algorithm and appealing to a sparsification result by Solomon [40], this approach can be used to obtain a $(1 + \epsilon)$-approximation. The space used by the algorithm (in terms of the number of edges) decreases at a rate which is doubly-exponential in p. This space/passes trade-off is somewhat unusual (but not unprecedented, e.g., [1, 9]); it is more typical in the data streams literature to see a singly-exponential or even just a polynomial trade-off.

2-pass Algorithm. Our 2-pass algorithm uses a result about the fractional matching due to McGregor and Vorotnikova [37]. They showed that it is possible to set a weight for each edge that just depends on the degrees of the endpoints of the edge, such that the total weight of all edges yields an $(\alpha + 2)$-approximation to the size of the maximum matching. This yields a simple 2-pass algorithm: we sample edges uniformly (first pass), compute their weights (second pass), and return an estimator based on these weights. This approach works well when the matching is large. By combining this approach with ideas from the previous result, we get our final algorithm, which uses small space regardless of the size of the matching.

3-pass Algorithm. Our 3-pass algorithm combines ideas from the other two algorithms. In the first pass, we identify all vertices with degree roughly $\sqrt{n}$ or higher. Let E_H be the edges that share an endpoint with these vertices, and let E_L be the remaining edges. The approach is to sparsify E_H to produce E'_H in such a way that $\mu(E'_H \cup E_L) \geq \mu(E_H \cup E_L)(1 - \epsilon)$. We then use uniform sampling to construct a multiset E'_L of edges, and use this, along with the weights of these edges, to estimate $\mu(E'_H \cup E_L)$ via the fractional matching approach.

1.3 Further Related Work: Matching Size Estimation in Graph Streams

Esfandiari et al. [20] were the first to consider the MSE problem in low-arboricity graphs in the streaming model. They focused on the insertion-only setting, where edges can only be inserted but not deleted, and gave a one-pass $O(\alpha)$-approximation algorithm that uses $\tilde{O}(\alpha \cdot n^{2/3})$ space. This result was significantly improved by Cormode et al. [15], who gave an algorithm with the same approximation guarantee, that uses only $O(\alpha \cdot \log^2 n)$ space. This was further improved, in terms of both the approximation factor and space, by McGregor and Vorotnikova [38].

At the heart of many of the algorithms for approximating the matching size in low-arboricity graphs lie structural lemmas that relate the maximum matching size to a function of the degree sequence of the graph in question. McGregor and Vorotnikova [37] gave such a characterization which gave rise to improved approximation guarantees over those established by Esfandiari et al. See also [26] for a different structural result. In particular, they obtained a $(5 + \epsilon)$-approximation for planar graphs, improving over a $(24 + \epsilon)$-approximation guarantee established in [20].

Assadi et al. [7] gave various upper and lower bounds for approximating the matching size in general graphs. They showed that there is a one-pass $\tilde{O}(n^2/\alpha^4)$-space algorithm that approximates the size of a matching within a factor of α in the dynamic graph stream setting. They also gave a lower bound, showing that space $n^{2-O(\epsilon)}$ is needed to obtain a $(1 + \epsilon)$-approximation factor.

1.4 Outline

In Section 2, we present known results, including a result due to McGregor and Vorotnikova [37] that links the matching size in low-arboricity graphs to the degree sequence of the graph, as well as a matching-size preserving sparsification result by Solomon [40]. We will need both of these in this paper. Subsequently, in Section 3, we give all our algorithmic results. In Section 4, we introduce the notion of low-arboricity matrices and show how their ranks can be approximated using algorithms for approximating the matching size in low-arboricity graphs. Finally, we conclude in Section 5 with some open problems.

2 Preliminaries

Notation and Definitions. Given a graph $G = (V, E)$, for each vertex $u \in V$, we denote by $\deg(u)$ the degree of u. The size of the maximum matching of G is denoted by $\mu(G)$. We also use $\mu(E')$ to denote the maximum matching among any set E' of edges. Throughout this paper, we let n and m denote the sizes of the sets V and E respectively, and α denote the arboricity of G. It is well-known that $m \leq \alpha \cdot n$.

Algorithmic Primitives. We will use the following results throughout the remaining sections, to simplify our proofs.

▶ **Theorem 4** (Algorithmic Primitives [13, 16, 27]). *There exist single-pass dynamic graph stream algorithms for:*

1. *Uniformly sampling edges: The algorithm uses $O(\log^2 n)$ space. This is a special case of ℓ_0-sampling [27].*
2. *Estimating degrees: We can compute estimates $\widetilde{\deg}$ such that with probability at least $1 - 1/n$:*
   a. *For all $u \in V$: $\deg(u) \leq \widetilde{\deg}(u) \leq \deg(u) + \|\mathbf{d}_{w\text{-tail}}\|_1/w$*
   b. *For all $u \in V$: $|\widetilde{\deg}(u) - \deg(u)| \leq \|\mathbf{d}_{w\text{-tail}}\|_2/\sqrt{w}$*

   where $\mathbf{d}_{w\text{-tail}}$ is the vector of degrees with the largest w entries replaced by 0. The first guarantee is achieved by CountMin sketch [16] and the second by CountSketch [13]. Both algorithms use $O(w \log n)$ space.

Structural Results. We will make repeated use of the following structural results for low-arboricity graphs. The first theorem effectively shows that there is a fractional matching where (a) the weight of each edge is just a function of the degrees of the endpoints of that edge, and (b) the total weight of the fractional matching is still a good approximation to the maximum-cardinality matching.

▶ **Theorem 5** (McGregor and Vorotnikova [37]). *Given a graph $G = (V, E)$ with arboricity at least α, let:*

$$w(G) = \sum_{(u,v)\in E} w_{u,v}, \text{ where } w_{u,v} = \min\left(\frac{1}{\deg(u)}, \frac{1}{\deg(v)}, \frac{1}{\alpha+1}\right) .$$

Then, we have:

$$\mu(G) \leq (\alpha+1)w(G) \leq \mu(G)\cdot\gamma, \text{ where } \gamma = \begin{cases} \alpha+2 & \text{if } \alpha \text{ is odd} \\ \frac{(\alpha+3)(\alpha+1)}{\alpha+2} & \text{if } \alpha \text{ is even} \\ \alpha+1 & \text{if } G \text{ is bipartite.} \end{cases}$$

Algorithm 1 A p-pass algorithm for approximating matching size.

```
1. $G_1 \leftarrow G$ and $s = m^{1/(2^p-1)} \cdot (2k)^{1-1/(2^p-1)}$, where $k$ is an upper bound on $\mu(G)$.
2. For $i = 1$ to $p-1$:
   a. Pass $i$: Use CountSketch to compute an estimate $\widetilde{\deg}(u)$ of the degree of each vertex $u \in G_i$.
      Let $V_i = \{u \in G_i : \widetilde{\deg}(u) \geq 0.75 \cdot \tau_{i+1}\}$, where $\tau_2 = \sqrt{m/s}$ and for $i \geq 2$, we have
      $\tau_{i+1} = m^{1/2^i} \cdot (2k/s)^{1-1/2^{i-1}}$. Also let:
          $G_{i+1} := G[V - V_1 - V_2 - \ldots - V_i]$
3. Pass $p$: Store all edges in $G_p$, and call this set $E_L$. For each $u \in V_1 \cup \ldots \cup V_{p-1}$, store $\Theta(\alpha/\epsilon)$
   incident edges to $u$. Call these edges $E'_H$.
4. Output: $\mu(E_L \cap E'_H)$.
```

Note that, in all cases, $\gamma \leq \alpha + 2$. Furthermore, if h is the number of vertices of degree at least $\alpha + 2$, and s is the number of edges whose endpoints have degree strictly less than $\alpha + 2$, then $\mu(G) \leq h + s \leq (\alpha + 2) \cdot \mu(G)$.

The next theorem demonstrates that, in a low-arboricity graph, it is possible to remove most of the edges incident to high degree vertices without significantly reducing the size of the maximum-cardinality matching.

▶ **Theorem 6** (Solomon [40]). *Fix a graph G with arboricity α, and a positive number $\epsilon > 0$. For each vertex $u \in V(G)$, mark $\Theta(\alpha/\epsilon)$ arbitrary edges incident to u. Let G' be the graph containing edges marked by both ends. Then, $\mu(G') \leq \mu(G) \leq (1 + \epsilon) \cdot \mu(G')$.*

3 Graph Results

3.1 $(1 + \epsilon)$-Approximation

Consider Algorithm 1. The main idea behind the algorithm is to exploit the fact that, if G has low arboricity and $\mu(G)$ is "small", then G has only a "small" number of "high"-degree vertices. If we can remove the high-degree vertices, then the remaining graph has significantly fewer edges, by the following lemma.

▶ **Lemma 7.** *Any graph G with maximum degree Δ has at most $(2\Delta - 1) \cdot \mu(G) < 2\Delta \cdot \mu(G)$ edges.*

Proof. This follows because the endpoints of a maximal matching is a vertex cover. ◀

If we can decrease the total number of edges, we can estimate the degrees in the remaining graph with greater accuracy. The next lemma quantifies this, where the proof exploits properties of the degree sequence for low-arboricity graphs. Repeating this process for $p - 1$ passes allows us to iteratively "peel off" high-degree vertices, until we are left with a graph that is sufficiently sparse such that it can be stored explicitly.

▶ **Lemma 8.** *Let $s \geq (\alpha + 2) \cdot \mu(G)$. Using $O(\alpha \cdot s \cdot \log n)$ space in the dynamic graph streaming model, given a graph with m edges, with high probability we can identify a subset of vertices that includes all vertices of degree $\geq \sqrt{m/s}$, and no vertex of degree $< 0.5\sqrt{m/s}$.*

Proof. We use CountSketch to find all vertices of degree more than $\sqrt{m/s}$. Let $d_1 \geq d_2 \geq d_3 \geq \ldots \geq d_n$ be the degree sequence of G. Observe that since $s \geq (\alpha+2) \cdot \mu(G)$, Theorem 5 implies s is greater than the total number of vertices with degree at least $\alpha + 2$. Thus, all degrees d_i with $i > s$ are at most $\alpha + 1$.

Hence, the ℓ_2^2 of the tail $\mathbf{t} = (d_{s+1}, d_{s+2}, \ldots, d_n)$ is at most $\|\mathbf{t}\|_\infty \cdot \|\mathbf{t}\|_1 \leq m(\alpha+1)$. Using CountSketch with space $O(\alpha \cdot s \cdot \log n)$, we can compute the degree of each vertex with additive error:

$$\frac{\ell_2((d_{s+1}, d_{s+2}, \ldots, d_n))}{\sqrt{\alpha \cdot s}} = O\left(\sqrt{m/s}\right) .$$

With the proper choice of the suppressed constant, we obtain an additive error of $0.25\sqrt{m/s}$. For each vertex u, let $\widetilde{\deg}(u)$ be the degree of u estimated by CountSketch. Let $V' = \{u : \widetilde{\deg}(u) \geq 0.75\sqrt{m/s}\}$. For any $u \in V'$, we have $\deg(u) \geq 0.75\sqrt{m/s} - 0.25\sqrt{m/s} = 0.5\sqrt{m/s}$. Furthermore, for every vertex u such that $\deg(u) \geq \sqrt{m/s}$, we have $\widetilde{\deg}(u) \geq \sqrt{m/s} - 0.25\sqrt{m/s} = 0.75\sqrt{m/s}$, implying that $u \in V'$. ◀

This leads to the following theorem.

▶ **Theorem 3.** *If the maximum matching size is upper bounded by k, there exists a $O(\epsilon^{-1} \cdot \alpha^2 \cdot n^{1/(2^p-1)} \cdot k^{1-1/(2^p-1)} \cdot \log n)$ space, p-pass dynamic graph streaming algorithm that returns a $(1+\epsilon)$-approximation for* MSE *with high probability. In particular, there exists a $O(\log\log n)$-pass algorithm that uses space $O(\epsilon^{-1} \cdot \alpha^2 \cdot k \cdot \log n)$.*

Proof. Consider the multi-pass algorithm where in pass i, we find a subset V_i of vertices in $G_i = G[V - V_1 - V_2 - \ldots - V_{i-1}]$ that includes all vertices of degree at least $\Delta_{i+1} = \sqrt{m_i/s}$, where m_i is the number of edges in G_i (see Algorithm 1). By Lemma 7, we have $m_i \leq 2\Delta_i \cdot \mu(G_i) \leq 2\Delta_i \cdot k$, since the maximum degree of G_i is less than Δ_i. Hence:

$$\begin{aligned}\Delta_{i+1} = \sqrt{m_i/s} &\leq \Delta_i^{1/2}(2k/s)^{1/2}\\ &\leq \Delta_{i-1}^{1/4}(2k/s)^{1/2+1/4}\\ &\leq \ldots\\ &\leq \Delta_2^{1/2^{i-1}}(2k/s)^{1-1/2^{i-1}}\\ &\leq m^{1/2^i}(2k)^{1-1/2^{i-1}}/s^{1-1/2^i} ,\end{aligned}$$

since $\Delta_2 \leq \sqrt{m/s}$. In particular, if $s = m^{1/(2^p-1)}(2k)^{1-1/(2^p-1)}$, then $\Delta_p \leq s/(2k)$.

Hence, G_p has at most $2(s/(2k)) \cdot k = s$ edges, which can be stored in memory. We collect $O(\alpha/\epsilon)$ edges incident to all vertices in $V_1 \cup \ldots \cup V_{p-1}$, and all edges between the remaining vertices. Let G_S be the new graph. The total number of edges in G_S is at most the number of edges in G_p, plus $(\alpha/\epsilon) \cdot \sum_{i=1}^{p-1} |V_i|$, which is at most $\alpha(\alpha+2) \cdot \mu(G)/\epsilon$, since by Theorem 5, the total number of vertices with degree higher than $\alpha + 2$ is at most $(\alpha + 2) \cdot \mu(G)$.

We then have that $\mu(G_S) \leq \mu(G) \leq (1+\epsilon) \cdot \mu(G_S)$. It follows from Theorem 6 that G_S is a supergraph of some graph G' satisfying $\mu(G') \geq \mu(G)/(1+\epsilon)$. Thus, $\mu(G_S) \geq \mu(G)/(1+\epsilon)$. On the other hand, since G_S is a subgraph of G, we obtain $\mu(G_S) \leq \mu(G)$, as desired. ◀

3.2 $O(\alpha)$-Approximation, Two Passes, and $\tilde{O}(m^{3/5})$ Space

Consider Algorithm 2. The analysis to establish the following theorem is a relatively straightforward application of the Chernoff bound.

Algorithm 2 A 2-pass algorithm for approximating matching size.

```
1. First Pass: Sample $t := 3m\epsilon^{-2}(\alpha+1)\log(2n)/k$ edges with replacement. Let $E'$ be the (multi-)set of edges sampled.
2. Second Pass: For each $(u,v) \in E'$, compute $w_{u,v} = \min\left(\frac{1}{\deg(u)}, \frac{1}{\deg(v)}, \frac{1}{\alpha+1}\right)$.
3. Output: $W = (m/t) \cdot X$, where $X = \sum_{e \in E'} w_e$.
```

▶ **Theorem 9.** *If $\mu(G) \geq k$, there exists a $O(\epsilon^{-2} \cdot \alpha \cdot mk^{-1} \cdot \log^3 n)$ space, 2-pass dynamic graph streaming algorithm that returns a $(\alpha+2)(1+\epsilon)$-approximation of $\mu(G)$ with probability at least $1 - 1/n$.*

Proof. Note that $\mathbb{E}[X] = t \cdot w(G)/m$. Since each edge sampled is drawn independently and each $0 \leq w_{u,v} \leq 1$, we can apply the Chernoff bound to conclude:

$$\Pr\left[|W - w(G)| \geq \epsilon \cdot w(G)\right] = \Pr\left[\left|X - \frac{w(G) \cdot t}{m}\right| \geq \epsilon \cdot \frac{w(G) \cdot t}{m}\right]$$
$$\leq 2 \cdot \exp\left(-\epsilon^2 \cdot \frac{w(G) \cdot t}{3m}\right) .$$

Theorem 5 implies $w(G) \geq k/(\alpha+1)$. Hence, setting $t = 3m\epsilon^{-2}(\alpha+1) \cdot \log(2n)/k$ ensures $\Pr[|W - w(G)| \geq \epsilon \cdot w(G)] \leq 1/n$. The result follows from Theorem 5. The space bound follows from the space complexity of edge sampling (Theorem 4). ◀

We can then combine the approach above with the result in Theorem 3 to yield a two-pass algorithm whose space complexity does not depend on upper or lower bounds on $\mu(G)$. Specifically:

1. We run the algorithm in Theorem 3 with $p = 2$ passes, and $k = m^{2/5}$, $\epsilon = 1$. This uses $O(\alpha^2 \cdot n^{3/5} \cdot \log n)$ space and returns a 2-approximation if $\mu(G) \leq m^{2/5}$.
2. In parallel, we run Algorithm 2 with $t = 12m\epsilon^{-2}(\alpha+1)\log(2n)/m^{2/5}$. This uses $O(\epsilon^{-2} \cdot \alpha \cdot m^{3/5} \cdot \log^3 n)$ space and returns an $(\alpha+2)(1+\epsilon)$-approximation if $w(G) \geq m^{2/5}/(\alpha+1) \cdot 1/4$.
3. To determine whether to output the result from the first algorithm or the second algorithm, we consider the variable $X = \sum_{e \in E'} w_e$ defined in Algorithm 2. Note $\mathbb{E}[X] = w(G) \cdot t/m$ and by an application of Chernoff bounds, if $w(G) \geq m^{2/5}/(\alpha+1)$, then we have:

   $$\Pr[X \leq \theta] \leq \exp(-m^{2/5}/(\alpha+1) \cdot t/m \cdot (1/3)) \leq 1/n \text{ , where } \theta = m^{2/5}/(\alpha+1) \cdot (1/2) \cdot t/m$$

   whereas if $w(G) \leq m^{2/5}/(\alpha+1) \cdot (1/4)$, then we have:

   $$\Pr[X \geq \theta] \leq \exp(-m^{2/5}/(\alpha+1) \cdot t/m \cdot (1/12)) \leq 1/n \text{ .}$$

   Consider returning the result from the first algorithm if $X < \theta$, and the result from the second algorithm otherwise. If $w(G) \leq m^{2/5}/(\alpha+1) \cdot 1/4$ then (a) with probability $1 - 1/n$ we return the output of the first algorithm, and (b) $\mu(G) \leq m^{2/5}$ by appealing to Theorem 5. Hence, we achieve a 2-approximation with high probability. If $w(G) \geq m^{2/5}/(\alpha+1)$, then with probability $1 - 1/n$ we return the output of the second algorithm and hence we achieve a $(\alpha+2)(1+\epsilon)$-approximation. If $(1/4) \cdot m^{2/5}/(\alpha+1) < w(G) < m^{2/5}/(\alpha+1)$, then the approximation factor from either algorithm is at most $(\alpha+2)(1+\epsilon)$.

▶ **Theorem 2.** *There exists a two-pass algorithm using $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{3/5} \cdot \log^3 n)$ space that returns an $(\alpha+2)(1+\epsilon)$-approximation for* MSE *with high probability.*

Algorithm 3 A 3-pass algorithm for approximating matching size.

1. *First Pass:* Use CountMin Sketch with $O(\alpha \cdot n^{1/2} \cdot \log n)$ space to find approximations of all degrees, such that with high probability, for all $u \in V$:

   $$\deg(u) \le \widetilde{\deg}(u) \le \deg(u) + \sqrt{n}/2 \ .$$

   Let $H = \{v : \widetilde{\deg}(u) \ge \sqrt{n}\}$, and note that H contains all vertices with degree at least $\sqrt{n}$ and no vertices with degree strictly less than $\sqrt{n}/2$. Let E_H be the set of edges incident to a vertex in H.
2. *Second Pass:* Let $G_L = (V \setminus H, E_L)$ be the graph formed by removing all vertices in H.
   - Compute m_L, the number of edges in G_L.
   - Sample $t = 3\epsilon^{-2} \cdot n^{1/2} \cdot \ln(2n)$ edges E'_L from G_L with replacement via ℓ_0-sampling. Note that E'_L could be a multiset.
   - For each $v \in H$, pick $O(\alpha/\epsilon)$ incident edges. Let E'_H be the chosen edges.
3. *Third Pass:* For each edge $(u, v) \in E'_L \cup E'_H$, compute

   $$w'_{u,v} = \min(1/\deg'(u), 1/\deg'(v), 1/(\alpha+1)) \ ,$$

   where $\deg'(u)$ is the number of incident edges in $E'_H \cup E_L$
4. *Output:* $w_1 + w_2$ where $w_1 = \sum_{e \in E'_H} w'_e$ and $w_2 = \frac{m_L}{t} \sum_{e \in E'_L} w'_e$.

3.3 $O(\alpha)$-Approximation, Three Passes, and $\tilde{O}(m^{1/2})$ Space

Consider Algorithm 3. The analysis proceeds as follows. After the first pass, we have partitioned the vertices into H and $V \setminus H$, such that all vertices in H have degree at least $\sqrt{n}/2$, and all vertices in $V \setminus H$ have degree at most $\sqrt{n}$. We will argue via Theorem 6 that maintaining a few edges incident to each vertex in H (these edges are called E'_H in the algorithm) decreases the size of the maximum matching by at most a factor of $(1-\epsilon)$. We then approximate the matching in the resulting graph via fractional matchings. Let A be the weight of the fractional matching on edges incident to H, and B be the weight of the other edges. We can compute A exactly (this will be returned as w_1), and we can estimate B by sampling edges that are not incident to vertices in H. The next lemma shows that our estimate for B is sufficiently accurate. The proof exploits that the weights of the edges contributing to B are all at least $1/\sqrt{n}$.

▶ **Lemma 10.** $\Pr[|w_2 - B| \ge \epsilon B] \le 1/n$, *where* $B = \sum_{e \in E_L} w'_e$.

Proof. Let Δ_L be the maximum degree of a vertex in $V \setminus H$. The definition of H ensures that $\Delta_L < \sqrt{n}$. The weight of each edge in E_L is between $1/\Delta_L$ and 1, and the average is B/m_L. Let X be the sum of w'_e for each $e \in E'_L$. Hence:

$$\mathbb{E}[X] = tB/m_L \ge t/\Delta_L \ .$$

By an application of the Chernoff bound, we have:

$$\Pr[|X - \mathbb{E}[X]| \ge \epsilon \cdot \mathbb{E}[X]] \le 2 \cdot \exp\left(-\epsilon^2 \cdot \frac{\mathbb{E}[X]}{3}\right) \le 2 \cdot \exp\left(-\epsilon^2 \cdot \frac{t}{3\Delta_L}\right) \ .$$

Hence, setting $t = 3\epsilon^{-2} \cdot n^{1/2} \cdot \ln(2n)$ makes the failure probability $1/n$. ◀

▶ **Theorem 1.** *There exists a three-pass algorithm using* $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{1/2} \cdot \log^3 n)$ *space that returns an* $(\alpha+2)(1+\epsilon)$*-approximation for* MSE *with high probability.*

Proof. By Theorem 6, we have $\mu(E)/(1+\epsilon) \leq \mu(E'_H \cup E_L) \leq \mu(E)$. Hence, by Theorem 5:

$$\mu(E)/(1+\epsilon) \leq (\alpha+1) \cdot w'(E'_H \cup E_L) \leq (\alpha+2) \cdot \mu(E) \ .$$

Note that $w'(E'_H \cup E'_L) = w_1 + w_2$, and $w(E'_H \cup E_L) = w_1 + B$. But, by Lemma 10, with probability at least $1 - 1/n$, we have $(1-\epsilon) \cdot B \leq w_2 \leq (1+\epsilon) \cdot B$. Therefore:

$$\mu(E) \cdot \frac{1-\epsilon}{1+\epsilon} \leq w_1 + w_2 \leq (\alpha+2) \cdot (1+\epsilon) \cdot \mu(E) \ .$$

Reparameterizing $\epsilon \leftarrow \epsilon/4$ gives the claimed result. The space used by the algorithm is $O(|H| \cdot \alpha/\epsilon + \alpha \cdot n^{1/2} \log n + t \log^2 n)$. Note that $|H| \leq 2m/\sqrt{n}$, and so the space is as claimed. ◀

4 Estimating Rank of Sparse Matrices

In this section, we consider the problem of estimating the rank of a matrix. It is well-known (e.g., see [35]) that the rank of the Tutte matrix[4] of a graph $G = (V, E)$ is exactly $2 \cdot \mu(G)$. It is therefore natural to look for other connections between rank and matching size.

Given an arbitrary matrix A, we next define a bipartite graph G_A that captures the structure of the nonzero entries of A.

▶ **Definition 11.** *Given an arbitrary $n_1 \times n_2$ matrix A define a bipartite graph $G_A = (L, R, E)$ where $L = [n_1], R = [n_2]$ and $(i,j) \in E$ if $A[i,j] \neq 0$. Note that a matching in G_A corresponds to a set of nonzero entries in M such that no two of these entries fall in the same column or row of A.*

Unfortunately it is too much to hope that $\mathsf{rank}(A)$ and $\mu(G_A)$ are always closely related. To see this, suppose that A is an $n \times n$ matrix of all 1s. Then, $\mathsf{rank}(A) = 1$, but $\mu(G_A) = n$. However, we show a significantly closer relationship for a certain family of sparse matrices which, by analogy to graph terminology, we call α-arboricity matrices.

▶ **Definition 12.** *A matrix is said to have arboricity α if every $t \times t$ submatrix has at most $\alpha \cdot t$ nonzero entries.*

Note that the class of matrices of arboricity α is much larger than the class of matrices which have a bounded number of nonzero entries in every row and column. However, it is more restrictive than bounding the number of nonzero elements; an $n \times n$ matrix with at most $\alpha \cdot n$ nonzero entries does not necessarily have arboricity α. For example, let A be an $n \times n$ matrix which has all zeros, except for a $\sqrt{\alpha \cdot n} \times \sqrt{\alpha \cdot n}$ submatrix of 1s; note that A has only $\alpha \cdot n$ nonzero entries, but does not have arboricity β for any $\beta < \sqrt{\alpha \cdot n}$.

▷ Claim 13. If G is a graph with arboricity α, its adjacency matrix A_G has arboricity at most 2α.

Proof. Consider an arbitrary $t \times t$ submatrix A'_G of A_G, consisting of t rows and t columns from A_G. Suppose the indices of the *common* rows and columns are $I = \{i_1, \ldots, i_s\}$. In addition, A'_G has rows $r_{j_1}, \ldots, r_{j_{s'}}$ and columns $c_{j'_1}, \ldots, c_{j'_{s'}}$ from A_G, such that $J = \{j_1, \ldots, j_{s'}\}$ and $J' = \{j'_1, \ldots, j'_{s'}\}$ are disjoint. Note that I, J, and J' correspond to disjoint subsets of

[4] Recall that the *Tutte matrix* T_G of a graph $G = (V, E)$ is the skew-symmetric matrix where $T[i,j] = 0$ if $(i,j) \notin E$; $T[i,j] = x_{i,j}$ if $i < j$ and $(i,j) \in E$; and $T[i,j] = -x_{i,j}$ if $i > j$ and $(i,j) \in E$.

vertices of G, and so A'_G is a submatrix of the adjacency matrix for the induced subgraph $G[I \cup J \cup J']$, which has at most $2\alpha \cdot (|I|+|J|+|J'|)$ nonzero entries. Each edge between I and J (or between I and J') corresponds to exactly one nonzero entry in the submatrix. None of the edges in $G[J \cup J']$ shows up. Finally, each edge in $G[I]$ corresponds to two nonzero entries (corresponding to the standard adjacency submatrix of A_G). Altogether, counting the weights of each of these relevant pairwise disjoint submatrices separately, this gives us $\alpha \cdot (2|I| + |J| + |J'|)$ nonzero entries in A'_G, which is at most $\alpha(|I| + |J| + |I| + |J'|) = 2\alpha \cdot t$, as claimed. ◁

▶ **Theorem 14.** *For any matrix A, we have $\mu(G_A)/\alpha \leq \mathsf{rank}(A) \leq \mu(G_A)$.*

Proof. To prove $\mu(G_A) \geq \mathsf{rank}(A)$, let T be the Tutte matrix of G_A. Then, from [35], we know that $\mathsf{rank}(T) = 2 \cdot \mu(G_A)$. Furthermore, $\mathsf{rank}(T) \geq 2 \cdot \mathsf{rank}(A)$. Hence, $\mathsf{rank}(A) \leq \mu(G_A)$, as claimed.

To prove $\mu(G_A)/\alpha \leq \mathsf{rank}(A)$, note that we may permute the rows and columns of A such that the first $\mu(G_A)$ diagonal entries of A are all nonzero. Let F be the top left $k \times k$ submatrix where $k = \mu(G_A)$. Note $\mathsf{rank}(F) \leq \mathsf{rank}(A)$. To lower bound the rank of F, first note that the total number of non-diagonal entries that are nonzero is at most $(\alpha - 1) \cdot k$, and so, at least one of F or F^T has $(\alpha - 1) \cdot k/2$ or fewer nonzero entries below the diagonal. Assume this is the case for F (if not, we can apply the rest of the argument to F^T rather than F). We will show that F contains a $(k/\alpha) \times (k/\alpha)$ principal submatrix where all the diagonal entries are nonzero and all entries below the diagonal are nonzero.

The process for finding this submatrix is as follows. We maintain a set $D \subseteq [k]$, where $j \in D$ means that the jth row and jth column will *not* be included in the submatrix. D is initially empty. We say that the jth column and row are *marked* if $j \in D$. Let nz_i be the total number of nonzero entries in the ith row and column of this submatrix that are below the diagonal and are not in marked rows/columns , i.e.:

$$nz_i = |S_i|, \text{ where } S_i = \{j < i : A[i,j] \neq 0, j \notin D\} \cup \{j > i : A[j,i] \neq 0, j \notin D\} .$$

Note that $\sum_{i \notin D} nz_i \leq (\alpha - 1) \cdot (k - |D|)$, because F has arboricity α, and the sum counts each element twice. Hence, $\min(nz_i) \leq (\alpha - 1)$. Let $i^* = \text{argmin}_{i:nz_i>0} nz_i$ and update $D \leftarrow D \cup S_{i^*}$. We repeat this process until all nonzero entries under the diagonal are in marked rows/columns. In each iteration, $|D|$ increases by at most $\alpha - 1$, but there is at least one more value i such that $nz_i = 0$. Hence, at the end of the algorithm, we have $k - |D| \geq k/\alpha$. ◀

Therefore, all our matching algorithms (and all previous results on MSE) also give algorithms for estimating the rank of low-arboricity matrices, with an additional multiplicative factor of α. Note that for rank approximation, we assume the dynamic model where at each time step, some entry of the matrix is set to a nonzero value (if it was currently zero) or set to zero (if it was currently nonzero).

The following example shows that the above bound is tight up to constants. And furthermore, any approximation via $w(G)$ loses an $O(\alpha^2)$ factor.

▶ **Example 15.** Let A and B be an $n \times n$ binary matrices, where

$$A[i,j] = \begin{cases} 1 & \text{if } i \leq \alpha \text{ or } j \leq \alpha \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad B[i,j] = \begin{cases} 1 & \text{if } i \leq \alpha \text{ or } j \leq \alpha \text{ or } i = j \\ 0 & \text{otherwise.} \end{cases}$$

First, note that $\mathsf{rank}(A) = 2, \mathsf{rank}(B) = n - \alpha + 1$, $\mu(G_A) = 2\alpha$, and $\mu(G_B) = n$. This establishes that the quantity $\mathsf{rank}(M)/\mu(G_M)$ can vary by an $\Omega(\alpha)$ factor.

$$w(G_A) = \frac{\alpha^2}{n} + \frac{2(n-\alpha)\alpha}{n} \approx 2\alpha \qquad \text{and} \qquad w(G_B) = \frac{\alpha^2}{n} + \frac{2(n-\alpha)\alpha}{n} + \frac{(n-\alpha)}{\alpha+1} \approx \frac{n}{\alpha} .$$

Hence, $\mathsf{rank}(M)/w(G_M)$ can vary by an $\Omega(\alpha^2)$ factor.

We end by noting that for a matrix A, the value of $\mu(G_A)$ does not depend on the the values of the nonzero entries in A. This immediately implies the following curious corollary.

▶ **Corollary 16.** *Changing the nonzero values in a matrix of arboricity α can change its rank by at most a factor of α.*

5 Conclusion

In this paper, we gave new multi-pass streaming algorithms for MSE in dynamic graph streams on graphs of arboricity α. As our main result, we showed that an $O(\alpha)$-approximation can be achieved in three passes with space $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{1/2} \cdot \log n)$, and we also gave a two-pass algorithm with a similar approximation guarantee that uses space $O(\epsilon^{-2} \cdot \alpha^2 \cdot n^{3/5} \cdot \log n)$. Furthermore, we designed a multi-pass algorithm with approximation factor $(1 + \epsilon)$ that operates based on an upper bound k on the maximum matching size. For example, it can give an $O(\log \log n)$-pass algorithm that uses space $O(\epsilon^{-1} \cdot \alpha^2 \cdot k \cdot \log n)$. Lastly, we introduced the notion of low-arboricity matrices and argued that matching algorithms for low-arboricity graphs can be used to approximate the rank of low-arboricity matrices with an $O(\alpha)$ loss in the approximation factor.

We conclude with two open problems. First, we are particularly intrigued by whether the $\sqrt{n}$-barrier established by Assadi et al. [7] for one-pass algorithms persists when multiple passes over the input are allowed. For instance, is there a constant pass algorithm with approximation factor $O(\alpha)$, whose space dependency on n is $o(\sqrt{n})$? Second, can we tighten the bounds in the one-pass setting?

References

1 Kook Jin Ahn, Graham Cormode, Sudipto Guha, Andrew McGregor, and Anthony Wirth. Correlation clustering in data streams. *Algorithmica*, 83(7):1980–2017, 2021. doi:10.1007/S00453-021-00816-9.

2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013. doi:10.1016/j.ic.2012.10.006.

3 Sepehr Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. 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 708–742. SIAM, 2022. doi:10.1137/1.9781611977073.32.

4 Sepehr Assadi. A simple (1 - ϵ)-approximation semi-streaming algorithm for maximum (weighted) matching. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.

5 Sepehr Assadi, Soheil Behnezhad, Sanjeev Khanna, and Huan Li. On regularity lemma and barriers in streaming and dynamic matching. 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 131–144. ACM, 2023. doi:10.1145/3564246.3585110.

6 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. 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 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.

7 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. 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 1723–1742. SIAM, 2017. doi:10.1137/1.9781611974782.113.

8 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364, 2016. doi:10.1137/1.9781611974331.ch93.

9 Sepehr Assadi, Christian Konrad, Kheeran K. Naidu, and Janani Sundaresan. O(log log n) passes is optimal for semi-streaming maximal independent set. 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 847–858. ACM, 2024. doi:10.1145/3618260.3649763.

10 Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.9.

11 Sepehr Assadi and Janani Sundaresan. Hidden permutations to the rescue: Multi-pass streaming lower bounds for approximate matchings. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 909–932. IEEE, 2023. doi:10.1109/FOCS57990.2023.00058.

12 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 263–274, 2015. doi:10.1007/978-3-662-48350-3_23.

13 M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP*, pages 693–703, 2002.

14 Rajesh Hemant Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to dynamic graph streams. *CoRR*, abs/1505.01731, 2015. URL: http://arxiv.org/abs/1505.01731, arXiv:1505.01731.

15 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ESA.2017.29.

16 Graham Cormode and S. Muthukrishnan. Space efficient mining of multigraph streams. In *ACM Principles of Database Systems*, pages 271–282, 2005. doi:10.1145/1065167.1065201.

17 Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

18 Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 337–348, 2013. doi:10.1007/978-3-642-40450-4_29.

19 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM J. Discrete Math.*, 25(3):1251–1265, 2011. doi:10.1137/100801901.

20 Hossein Esfandiari, Mohammad Taghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1217–1233, 2015. doi:10.1137/1.9781611973730.81.

21 J. Feigenbaum, S. Kannan, McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005. doi:10.1016/J.TCS.2005.09.013.

22 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

23 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485, 2012. URL: http://portal.acm.org/citation.cfm?id=2095157&CFID=63838676&CFTOKEN=79617016, doi:10.1137/1.9781611973099.41.

24 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, Palo Alto, California, USA, 5-7 June, 2013*, pages 287–298, 2013. doi:10.1109/CCC.2013.37.

25 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In James M. Abello and Jeffrey Scott Vitter, editors, *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, volume 50 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 107–118. DIMACS/AMS, 1998. doi:10.1090/DIMACS/050/05.

26 Hossein Jowhari. An estimator for matching size in low arboricity graphs with two applications. *J. Comb. Optim.*, 45(1):21, 2023. doi:10.1007/S10878-022-00929-Z.

27 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '11, pages 49–58, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989284.1989289.

28 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013. doi:10.1137/1.9781611973105.121.

29 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751, 2014. doi:10.1137/1.9781611973402.55.

30 Christian Konrad. Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 840–852, 2015. doi:10.1007/978-3-662-48350-3_70.

31 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012. doi:10.1007/978-3-642-32512-0_20.

32 Christian Konrad and Kheeran K. Naidu. On two-pass streaming algorithms for maximum bipartite matching. In Mary Wootters and Laura Sanità, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*, volume 207 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.APPROX/RANDOM.2021.19.

33 Christian Konrad and Kheeran K. Naidu. An unconditional lower bound for two-pass streaming algorithms for maximum matching approximation. 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 2881–2899. SIAM, 2024. doi:10.1137/1.9781611977912.102.

34 Christian Konrad, Kheeran K. Naidu, and Arun Steward. Maximum matching via maximal matching queries. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 41:1–41:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.41.

35 László Lovász. On determinants, matchings, and random algorithms. In Lothar Budach, editor, *Fundamentals of Computation Theory, FCT 1979, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory, Berlin/Wendisch-Rietz, Germany, September 17-21, 1979*, pages 565–574. Akademie-Verlag, Berlin, 1979.

36 Andrew McGregor. Finding graph matchings in data streams. *APPROX-RANDOM*, pages 170–181, 2005. doi:10.1007/11538462_15.

37 Andrew McGregor and Sofya Vorotnikova. Planar matching in streams revisited. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 17:1–17:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.APPROX-RANDOM.2016.17.

38 Andrew McGregor and Sofya Vorotnikova. A simple, space-efficient, streaming algorithm for matchings in low arboricity graphs. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, volume 61 of *OASIcs*, pages 14:1–14:4. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASICS.SOSA.2018.14.

39 C. St.J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 1961. doi:10.1112/jlms/s1-36.1.445.

40 Shay Solomon. Local algorithms for bounded degree sparsifiers in sparse graphs. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 52:1–52:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ITCS.2018.52.

41 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. doi:10.1007/s00453-010-9438-5.

Improved Linearly Ordered Colorings of Hypergraphs via SDP Rounding

Anand Louis
Indian Institute of Science, Bengaluru, India

Alantha Newman
Université Grenoble Alpes, France

Arka Ray
Indian Institute of Science, Bengaluru, India

Abstract

We consider the problem of *linearly ordered* (LO) coloring of hypergraphs. A hypergraph has an LO coloring if there is a vertex coloring, using a set of ordered colors, so that (i) no edge is monochromatic, and (ii) each edge has a unique maximum color. It is an open question as to whether or not a 2-LO colorable 3-uniform hypergraph can be LO colored with 3 colors in polynomial time. Nakajima and Živný recently gave a polynomial-time algorithm to color such hypergraphs with $\widetilde{O}(n^{1/3})$ colors and asked if SDP methods can be used directly to obtain improved bounds. Our main result is to show how to use SDP-based rounding methods to produce an LO coloring with $\widetilde{O}(n^{1/5})$ colors for such hypergraphs. We show how to reduce the problem to cases with highly structured SDP solutions, which we call *balanced* hypergraphs. Then we discuss how to apply classic SDP-rounding tools in this case to obtain improved bounds.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases hypergraph coloring, SDP rounding, promise constraint satisfaction problems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.30

Related Version *Full Version*: https://arxiv.org/abs/2405.00427 [21]

Funding *Anand Louis*: Supported in part by SERB Award CRG/2023/002896 and the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).
Arka Ray: Supported in part by the Walmart Center for Tech Excellence at IISc (CSR Grant WMGT-23-0001).

1 Introduction

Approximate graph coloring is a well-studied "promise" optimization problem. Given a simple graph $G = (V, E)$ that is promised to be k-colorable, the goal is to find a coloring of G using the minimum number of colors. A (proper) *coloring* is an assignment of colors, which can be represented by positive integers, to the vertices of G so that for each edge ij in G, the vertices i and j are assigned different colors. The most popular case of this problem is when the input graph is promised to be 3-colorable. Even with this very strong promise, the gap between the upper and lower bounds are quite large: the number of colors used by the state-of-the-art algorithm is $\widetilde{O}\left(n^{0.19996}\right)$ [19], while it is NP-hard to color a 3-colorable graph with 5 colors [4]. There is also super constant hardness conditioned on assumptions related to the Unique Games Conjecture [11]. More generally, when we are promised that the graph G is k-colorable, it is NP-hard to color it using $\binom{k}{\lfloor k/2 \rfloor} - 1$ colors [25]. Regarding upper bounds, we note that almost all algorithms for coloring 3-colorable graphs use some combination of semidefinite programming (SDP) and combinatorial tools [17, 2, 19].

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 30; pp. 30:1–30:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Approximate hypergraph coloring is a natural generalization of the above problem to hypergraphs. Here, we want to assign each vertex a color such that there are no monochromatic edges, while using the minimum number of colors. In the case of hypergraph coloring, we know that for every pair of constants $\ell \geqslant k \geqslant 2$, it is NP-hard to ℓ-color a k-colorable 3-uniform hypergraph [12]. Even in the special case, when the 3-uniform hypergraph is promised to be 2-colorable, there is a large gap between the best algorithm, which uses at most $\widetilde{O}\left(n^{1/5}\right)$ colors [20, 1, 10] and the aforementioned (super constant) lower bound.

In this paper, we study a variant of the hypergraph coloring problem known as *linearly ordered coloring*, introduced in several different contexts by [18, 9, 3]. A linearly ordered (LO) k-coloring of an r-uniform hypergraph assigns an integer from $\{1, \ldots, k\}$ to every vertex so that, in each edge in the hypergraph, there is a unique vertex assigned the maximum color in the (multi)set of colors for that edge. Recently, there has been a renewed interest in studying this problem. This is because this problem constitutes a gap in the understanding of the complexity of an important class of problems called *promise constraint satisfaction problems* (PCSPs). To elaborate, [13, 7] classified the complexity of all (symmetric) PCSPs on the binary alphabet, showing that these problems are either polynomial-time solvable or NP-complete. Subsequently, [3] gave a complete classification for PCSPs of the form: given a 2-colorable 3-uniform hypergraph, find a 3-coloring. Here, the notion of "coloring" can have several definitions. As highlighted by [3], the only PCSP of this type whose complexity is unresolved is that of determining whether a 3-uniform hypergraph is 2-LO colorable or is not even 3-LO colorable. In contrast, it was recently shown that it is NP-complete to decide if a 3-uniform hypergraph is 3-LO colorable or not even 4-LO colorable [14].

The work [22] addresses the corresponding optimization problem by giving an algorithm to compute an LO coloring using at most $\widetilde{O}\left(n^{1/3}\right)$ colors for a 2-LO colorable 3-uniform hypergraph. [22] leave open the question of finding an LO coloring for such a hypergraph using fewer colors. Moreover, they state that they do not know how to directly use SDP-based methods[1] and remark that SDP-based approaches seem "less suited for LO colorings". In this paper, one of our main contributions is to show how to use SDP relaxations to give an improved bound for coloring such hypergraphs. Our main result improves this bound significantly by using at most $\widetilde{O}\left(n^{1/5}\right)$ colors to LO color a 2-LO colorable 3-uniform hypergraph.

▶ **Theorem 1.** *Let H be a 2-LO colorable 3-uniform hypergraph on n vertices. Then there exists a (randomized) polynomial-time algorithm that finds an LO coloring of H using $\widetilde{O}\left(n^{1/5}\right)$ colors.*

The SDP relaxation that we use is similar to the natural SDP used in the case of 2-colorable 3-uniform hypergraphs [20]. In fact, the upper bound on the number of colors used in Theorem 1 is the same as the upper bound given by [20] to color 2-colorable 3-uniform hypergraphs. It is the same SDP used by [8] who show that a straightforward hyperplane rounding algorithm yields a solution to the PCSP (1-IN-3-SAT, NAE-3-SAT), in which we are given a satisfiable instance of the first problem and we want to find a feasible solution for the second. Notice that a satisfiable (1-IN-3-SAT) instance on all positive literals is exactly a 2-LO colorable 3-uniform hypergraph.

[1] However, they do use [15] which is an indirect use of SDP-based methods.

General Framework for (Hyper)Graph Coloring

Most algorithms for coloring graphs and hypergraphs proceed iteratively, producing a partial coloring of the remaining (uncolored) vertices at each step. This was formalized by [5], following [23]. The goal is to color a significant number of vertices with few colors in each step, ensuring that the number of iterations and therefore, the overall number of colors used, is small. Typically, in each step, the method used to color the vertices is chosen according to the degree of the graph (or hypergraph) induced on the remaining vertices. In particular, if the induced graph (or hypergraph) has a low degree, then most algorithms use an SDP-based method to find a large independent set, which can be assigned a single color [17, 6, 2]. The algorithm for LO-coloring presented in [22], as well as ours, uses this general framework, except that in [22], they did not use an SDP-based method directly, and instead used [15] to find a large independent set. The improved upper bound on the number of colors output by our algorithm comes from using an SDP and rounding methods tailored to LO coloring.

Overview of our SDP-Based Approach

As noted, we first solve a natural SDP relaxation for 2-LO coloring. Then our rounding proceeds in two steps. In the first step we look at the projection of the vectors to a particular special vector (the vector $\mathsf{v}_\emptyset$ in Proposition 2) from the solution of the SDP, which signifies the color that is unique in all edges in the promised 2-LO coloring. For each of the three vertices in an edge, all three of the corresponding vectors can have a projection onto this special vector with roughly the same value (a *balanced* edge), or they can have very different values (an *unbalanced* edge). It is also possible to classify vertices into balanced and unbalanced (see Definition 4 for formal definitions) so that balanced edges contain only balanced vertices. We use a combinatorial rounding procedure to color all the unbalanced vertices with a small number of colors, leaving only a balanced (sub)hypergraph to be colored. Since this number of colors is much smaller than the bound stated in Theorem 1, this can be viewed as a reduction of the problem to the balanced case. To the best of our knowledge, this rounding method is not present in previous works on LO-coloring and thus, this tool can be considered a main contribution of this paper. We note that [20] showed that the vectors can be "bucketed" with respect to their projection onto a special vector, and used a simple argument to show that there is a large bucket on which they can focus. Our approach allows us to focus on a single bucket containing vectors with projection $\approx -1/3$ with the special vector, which have useful geometric properties.

In the second step we color the hypergraph containing the balanced edges. In this step, we produce (following [22]) an "even" independent set or an "odd" independent set at each round. An *even independent set* is one which intersects each hyperedge two or zero times, while an *odd independent set* intersects each hyperedge one or zero times. To find an even independent set, we use the same approach used by [22]. To find an odd independent set, we use a variant of the standard threshold rounding for a coloring SDP [15, 20]. As in [20] rather than use the vectors output by the SDP solution, we use a modified set of vectors, which have properties useful to obtain better bounds from the threshold rounding. Specifically, the set consists of the normalized projections of the vectors from the SDP solution onto the space orthogonal to the special vector; in the balanced case, the special vector seems to provide no information that is useful to construct a coloring. Combining all the colorings requires some technical care, since we need to always maintain an LO coloring, but it can be done and some of the work has already been done in [22].

Update on Independent and Subsequent Work

After the initial conference submission of our paper, the work [16] appeared on the arXiv. The second version appeared after we posted our paper to arXiv and pointed out that in fact we do not need to consider the balanced case. Indeed, the observation in Section 3 of [16] can be interpreted as giving an alternative and better SDP rounding in the balanced case, directly reducing the balanced case to the unbalanced case. We discuss this more at the end of Section 4.

2 Tools for LO Coloring and Proof of the Main Theorem

In this section, we give an overview of our approach to color a 2-LO colorable 3-uniform hypergraph $H = (V, E)$ with few colors. Following [22], we assume that the input hypergraph H is a *linear hypergraph*, which is defined as follows.

▶ **Definition 2.** *A 3-uniform hypergraph is linear if every pair of edges intersects in at most one vertex.*

This is not a restriction because we can construct an equivalent 3-uniform hypergraph.

▶ **Proposition 3** (Proposition 3 in [22]). *There is a polynomial-time algorithm that, if given an 2-LO colorable 3- uniform hypergraph H, constructs an 2-LO colorable linear 3-uniform hypergraph H' with no more vertices than H such that, if given an LO k-colouring of H', one can compute in polynomial time an LO k-colouring of H.*

Given an 2-LO colorable 3-uniform hypergraph $H = (V, E)$, one can consider LO coloring it with $\{-1, +1\}$, with the natural ordering. Then we have $x_a + x_b + x_c = -1$ for each edge $\{a, b, c\} \in E$, where x_a is the color assigned to vertex $a \in V$. Relaxing this constraint to a vector program we get SDP 2.[2] SDP

$$\mathsf{v}_a + \mathsf{v}_b + \mathsf{v}_c = -\mathsf{v}_\emptyset \qquad \forall \{a, b, c\} \in E, \tag{1}$$

$$\|\mathsf{v}_a\|^2 = 1 \qquad \forall a \in V \cup \{\emptyset\}. \tag{2}$$

For any $a \in V$, we now define $\gamma_a \overset{\text{def}}{=} \langle \mathsf{v}_a, \mathsf{v}_\emptyset \rangle$. The values $\{\gamma_a\}_{a \in V}$ might not be integral and could even be *perfectly balanced* (i.e., $\gamma_a = \gamma_b = \gamma_c = -\frac{1}{3}$ for an edge $\{a, b, c\} \in E$). Hence, these values might not contain any information as to how the colors should be assigned to the vertices, and they might not even reveal information as to which vertex in an edge should receive the largest color. However, when all edges contain balanced vertices (i.e., $\gamma_v \approx -\frac{1}{3}$ for all vertices), threshold rounding will be used. Formally, we have the following definition.

▶ **Definition 4.** *For $\varepsilon > 0$, we say a vertex $v \in V$ is* ε-balanced *if $\gamma_v \in [-1/3 - \varepsilon, -1/3 + \varepsilon]$.*

For the rest of this paper, we fix $\varepsilon = 1/n^{100}$, where n is the number of vertices in the (fixed) hypergraph that we are trying to LO color. This is an abuse of notation, but simplifies our presentation. If a vertex is not ε-balanced, we say that it is *unbalanced.* If all vertices of a hypergraph H are ε-balanced, we say that H is an ε-balanced hypergraph.

We observe that there is a combinatorial method to color all unbalanced vertices using relatively few colors. This rounding method uses a bisection-like strategy on $\{\gamma_a\}_{a \in V}$ to color the unbalanced vertices and outputs a *partial LO coloring*, which we define as follows.

[2] Observe that SDP 2 can equivalently be written in terms of dot products using the following constraints: (i) $\langle \mathsf{v}_a + \mathsf{v}_b + \mathsf{v}_c + \mathsf{v}_\emptyset, \mathsf{v}_a + \mathsf{v}_b + \mathsf{v}_c + \mathsf{v}_\emptyset \rangle = 0 \quad \forall \{a, b, c\} \in E,$ and (ii) $\langle \mathsf{v}_a, \mathsf{v}_a \rangle = 1 \quad \forall a \in V \cup \{\emptyset\}$.

▶ **Definition 5.** *A* partial LO coloring *of a 3-uniform hypergraph $H = (V, E)$ is a coloring of a subset of vertices $V_1 \subseteq V$ using the set of colors C such that for each edge $e \in E$, the set $e \cap V_1$ has a unique maximum color from C.*

The next lemma is proved in Section 3.

▶ **Lemma 6.** *Let $H = (V, E)$ be a 2-LO colorable 3-uniform hypergraph and let $\varepsilon > 0$. Then there exists a polynomial-time algorithm that computes a partial LO coloring of H using $O\left(\log\left(\frac{1}{\varepsilon}\right)\right)$ colors that colors all unbalanced vertices.*

We remark that the previous lemma can be viewed as a reduction from LO coloring in 2-LO colorable 3-uniform hypergraphs to LO coloring in 2-LO colorable 3-uniform *balanced* hypergraphs. To formalize this, let V_U denote the vertices that are colored in a partial LO coloring produced via Lemma 6. Let $V_B = V \setminus V_U$. Notice that V_B contains only ε-balanced vertices, while V_U contains all the unbalanced vertices but might also contain some ε-balanced vertices. Thus, the induced hypergraph $H_B = (V_B, E(V_B))$ is a balanced hypergraph.[3] We now show that we can combine a partial LO coloring for $H = (V, E)$ which colors V_U and an LO coloring for $H_B = (V_B, E(V_B))$ to obtain an LO coloring of H.

▶ **Proposition 7.** *Let $H = (V_B \cup V_U, E)$ be a 2-LO colorable 3-uniform hypergraphs, let $\varepsilon > 0$. Let c_U be a partial LO coloring of H using colors from the set C_U that only assigns colors to V_U and let c_B be an LO coloring of $H_B = (V_B, E(V_B))$ using colors from the set C_B. Then we can obtain an LO coloring of H using at most $|C_U| + |C_B|$ colors.*

Proof. We assume that the colors in the set C_U are larger than the colors in the set C_B. We want to show that the given assignment of colors from C_U for vertex set V_U and C_B for vertex set V_B taken together forms a proper LO coloring of H.

Any edge $e \in E$ with $|e \cap V_B| = 3$ or $|e \cap V_U| = 3$ has a unique maximum color by assumption since c_B is an LO coloring of H_B and c_U is a partial LO coloring of H. Suppose $|e \cap V_U| = 2$. Then, by definition of partial LO coloring, it has a unique maximum in C_U and will have a unique maximum in the output coloring. If $|e \cap V_U| = 1$, then e has a unique maximum color, because all colors in C_U are larger than the colors in C_B. ◀

Thus, if our goal is to LO color 2-LO colorable 3-uniform hypergraphs with a polynomial number of colors, we can focus on LO coloring *balanced* 2-LO colorable 3-uniform hypergraphs. The next corollary follows from Lemma 6 and Proposition 7.

▶ **Corollary 8.** *Let $\alpha \in (0, 1)$. Suppose we can LO color an ε-balanced 2-LO colorable 3-uniform hypergraph H with $\widetilde{O}(n^\alpha)$ colors. Then we can LO color a 2-LO colorable 3-uniform hypergraph with $\widetilde{O}(n^\alpha)$ colors.*

Now we can focus on balanced hypergraphs. We capitalize on the promised structure to prove the next lemma, in which we show that we can find an LO coloring for a balanced hypergraph, in particular for $H_B = (V_B, E(V_B))$.

▶ **Lemma 9.** *Let $H_B = (V_B, E_B)$ be an ε-balanced 2-LO colorable 3-uniform hypergraph. Then there exists a polynomial-time algorithm that computes an LO coloring using at most $\widetilde{O}(|V_B|^{1/5})$ colors.*

[3] Note that for a hypergraph $H = (V, E)$ and $S \subset V$, we say $H' = (S, E(S))$ contains the edges *induced* on S, meaning an edge belongs to H' if all of its vertices belong to S. In other words, an induced subhypergraph of a 3-uniform must also be 3-uniform (or empty). Notice that S can contain vertices that do not belong to any edge in $E(S)$. These vertices can receive any color in a valid LO coloring of H'.

We recall our main theorem.

▶ **Theorem 1.** *Let H be a 2-LO colorable 3-uniform hypergraph on n vertices. Then there exists a (randomized) polynomial-time algorithm that finds an LO coloring of H using $\widetilde{O}\left(n^{1/5}\right)$ colors.*

The proof of Theorem 1 follows from Corollary 8 and Lemma 9. It remains to prove Lemma 9, which we discuss next.

Coloring by Finding Independent Sets

In many graph coloring algorithms, we "make progress" by finding an independent set and coloring it with a new color [5, 17, 6, 20, 22]. When LO coloring a hypergraph, a similar idea may be used, but we need to consider certain types of independent sets. With the standard notion of independent set in a 3-uniform hypergraph, in which the independent set intersects each edge of the hypergraph at most twice, it is not clear how to obtain a coloring in which each edge contains a unique maximum color. Thus, for a 3-uniform hypergraph $H = (V, E)$, following the approach of [22], we consider the following two types of independent sets.[4]

Odd Independent Set: We call $S \subseteq V$ an *odd independent set* if $|S \cap e| \leqslant 1$ for each edge $e \in E$.

Even Independent Set: We call $S \subseteq V$ an *even independent set* if $|S \cap e| \in \{0, 2\}$ for each edge $e \in E$.

In Lemma 10, we show that we can make progress by coloring an odd independent set with a "large" color or by coloring an even independent set with a "small" color. This is formally stated in a proposition from [22]. Since we modify the presentation slightly to ensure compatibility with our framework, we include the statement and the proof here for the sake of completeness.

▶ **Lemma 10** (Corollary of Proposition 5 in [22])**.** *Let $H = (V, E)$ be a hypergraph, let $S_1 \subseteq V$ be an odd independent set and let $S_2 \subseteq V$ be an even independent set. Let $H_1 = (V_1, E_1)$, $H_2 = (V_2, E_2)$ be the hypergraphs induced by $V_1 = V \setminus S_1$ and $V_2 = V \setminus S_2$, respectively. Then,*

1. *An LO coloring of H_1 using a set of colors C_1 can be extended to an LO coloring of H by assigning a color c_1 that is strictly larger than all the colors in C_1 to the vertices in S_1.*
2. *Analogously, an LO coloring of H_2 using a set of colors C_2 can be extended to an LO coloring of H by assigning c_2 to the vertices in S_2 where c_2 is strictly smaller than all the colors in C_2.*

Proof. In the proposed extension of the coloring from H_1 to H, there is no edge $e \in E_1$ where the maximum color in e occurs more than once in e; otherwise, the promised coloring of H_1 using C_1 is not valid. Consider any edge $\{u, v, w\} \in E \setminus E_1$. By definition of S_1, we have $|\{u, v, w\} \cap S_1| \leqslant 1$. Note that $|\{u, v, w\} \cap S_1| \neq 0$ as $\{u, v, w\} \notin E_1$. Therefore, we must have $|\{u, v, w\} \cap S_1| = 1$. Without loss of generality, assume that $u \in S_1$ and $v, w \notin S_1$. Then, in the proposed coloring, c_1 is only used for u, while v, w are colored using some color(s) from C_1. So, c_1 is the largest color in $\{u, v, w\}$ and occurs exactly once. Hence, for every edge, the corresponding (multi)set of colors has a unique maximum, and we conclude that the proposed coloring is a proper LO coloring of H.

[4] We remark that what [22] refer to as an "independent set" is what we refer to here as an "odd independent set".

Similarly, in the proposed extension of coloring from H_2 to H there is no edge $e \in E_2$ where the maximum color in e occurs more than once in e. Again, consider any edge $\{u, v, w\} \in E \setminus E_2$. In this case, we have $|\{u, v, w\} \cap S_2| = 2$. Without loss of generality, assume that $u, v \in S_2$ and $w \notin S_2$. Then, in the proposed coloring, c_2 is only used on u, v, while w is colored using some color c from C_2. So, c is the largest color in $\{u, v, w\}$ and it occurs exactly once. Hence, for every edge, the corresponding (multi)set of color has a unique maximum, and the proposed coloring is therefore a proper LO coloring of H. ◀

The following proposition is essentially Lemma 1 in [5] and follows in a straight-forward manner from Lemma 10.

▶ **Proposition 11** (Proposition 5 in [22]). *Let $H = (V, E)$ be an ε-balanced, 2-LO colorable 3-uniform linear hypergraph on m vertices. Suppose we can find an odd independent set of size at least $f(m)$ in H or an even independent set of size at least $f(m)$ in H (where f is* nearly-polynomial[5]*), then there exists a polynomial-time algorithm that colors any ε-balanced, 2-LO colorable 3-uniform linear hypergraph on n vertices with $n/f(n)$ colors.*

Following this standard notion of "making progress" from [5], we simply need to show that we can find an even or an odd independent set of size at least $f(m)$ in a 2-LO colorable 3-uniform ε-balanced hypergraph on m vertices. This will imply that we can color H_B with $|V_B|/f(V_B)$ colors. We will show that we can set $f(m) = \widetilde{\Theta}(m^{4/5})$, which will yield the bound in Lemma 9.

As is typical, our coloring algorithm makes progress using two different methods and chooses between the two methods depending on the degree. In the high-degree case, we use the method from [22] to find a large even independent set. The method to find a large even independent from [22] requires the input hypergraph to be a linear hypergraph, which, as discussed previously, we can assume by Proposition 3.

▶ **Proposition 12** (Proposition 11 in [22]). *Let $H = (V, E)$ be a linear 2-LO colorable 3-uniform hypergraph and Δ be such that $|E| = \Omega(\Delta |V|)$. Then there is a polynomial-time algorithm that finds a even independent set of size at least $\Omega(\sqrt{|V|\Delta})$.*

In the low-degree case, we show how to use an SDP based rounding method to find a large odd independent set. Here, we capitalize on the assumption that our input hypergraph is ε-balanced to obtain an improvement over the analogous lemma from [22]. In Section 4, we prove Lemma 13.

▶ **Lemma 13.** *Let $H = (V, E)$ be a $\frac{1}{|V|^{100}}$-balanced 2-LO colorable 3-uniform hypergraph $H = (V, E)$ with average degree at most Δ. Then there exists a (randomized) polynomial-time algorithm to compute an odd independent set of size at least $\Omega\left(\frac{|V|}{\Delta^{1/3}(\ln \Delta)^{3/2}}\right)$.*

Finally, we are now ready to prove Lemma 9.

Proof of Lemma 9. We need to show that on a linear 2-LO-colorable 3-uniform ε-balanced hypergraph on m vertices, we can always find either an even independent set or an odd independent set of size at least $f(m) = \widetilde{\Omega}(m^{4/5})$. By Lemma 10, this will imply we can color H_B with $|V_B|/f(|V_B|)$ colors.

[5] Definition 1 in [5]. A function $f(m) = m^{\alpha}\mathsf{polylog} m$ for $\alpha > 0$ is nearly-polynomial.

Take Δ be a parameter (fixed later) so that we say we are in the high-degree regime if the average degree is higher than Δ. Otherwise, we say that we are in the low-degree regime. In the high degree-regime, use Proposition 12 to find an even independent set S of size at least $\Omega(\sqrt{m\Delta})$. In the low-degree regime, we invoke Lemma 13 to find an odd independent set S of size at least $\widetilde{\Omega}(m/\Delta^{1/3})$. Setting $\Delta = m^{3/5}$ implies that the independent set we find has size at least $m^{4/5}$. Finally, by Proposition 11 we have the desired bound on the number of colors used. ◀

3 Combinatorial Rounding for Unbalanced Vertices

In this section, we prove Lemma 6. In other words, we show that for any $\varepsilon > 0$, Algorithm 1 outputs a partial LO coloring using $O\left(\log\left(\frac{1}{\varepsilon}\right)\right)$ colors so that all the unbalanced vertices are assigned a color.

▶ **Lemma 6.** *Let $H = (V, E)$ be a 2-LO colorable 3-uniform hypergraph and let $\varepsilon > 0$. Then there exists a polynomial-time algorithm that computes a partial LO coloring of H using $O\left(\log\left(\frac{1}{\varepsilon}\right)\right)$ colors that colors all unbalanced vertices.*

To prove this lemma, we give an algorithm, which given the value $\{\gamma_v\}$ for each vertex v (from SDP 2), is then combinatorial. The algorithm also takes as input the value of ε, which is the parameter we use to define ε-balanced.

Algorithm 1 Combinatorial Rounding.

```
Input: A 2-LO colorable 3-uniform hypergraph $H = (V, E)$, $\varepsilon > 0$, the values $\{\gamma_a\}$ for all
$a \in V$ and set $C$ of linearly ordered colors.
Output: A partial LO coloring of all unbalanced vertices in $V$.
1. Set $j := 0, \ell_0 := -1, u_0 := 1, I_0 := [\ell_0, u_0]$.
2. While $I_j \not\subseteq [-1/3 - \varepsilon, -1/3 + \varepsilon]$ do:
   a. If $j$ is even then set $I_{j+1}$ to the lower half of $I_j$, if $j$ is odd then set $I_{j+1}$ to be the
      upper half of $I_j$. More precisely, set
      $\ell_{j+1} := \begin{cases} \frac{\ell_j+u_j}{2} & j \text{ is odd} \\ \ell_j & \text{otherwise} \end{cases}$    $u_{j+1} := \begin{cases} \frac{\ell_j+u_j}{2} & j \text{ is even} \\ u_j & \text{otherwise} \end{cases}$
      and set $I_{j+1} := [\ell_{j+1}, u_{j+1}]$.
   b. Set $S_{j+1} := \{a \in V | \gamma_a \in I_j \setminus I_{j+1}\}$ and color $S_{j+1}$ using the largest unused color from
      $C$.
   c. Set $j := j + 1$.
```

We will use the following observation.

▶ **Observation 14.** *For any $\{a, b, c\} \in E$, we have $\gamma_a + \gamma_b + \gamma_c = -1$.*

Proof. From constraint (1), we get $\gamma_a + \gamma_b + \gamma_c = \langle \mathsf{v}_a + \mathsf{v}_b + \mathsf{v}_c, \mathsf{v}_\emptyset \rangle = \langle -\mathsf{v}_\emptyset, \mathsf{v}_\emptyset \rangle = -1$. ◀

On a high level, the algorithm partitions the interval $[-1, 1]$ and assigns colors to vertices depending on where their corresponding γ_a values fall in this interval. For example, in the first iteration of the algorithm, we set S_1 to contain all vertices whose γ_a values fall into the interval $(0, 1]$. Notice that by Observation 14, at most one vertex from an edge will qualify. Now, all remaining vertices have γ_a values in the interval $[-1, 0]$. Next, we consider all vertices whose γ_a values fall into the interval $[-1,-1/2)$. Again, an edge with all three values in $[-1,0]$ can not have more than one vertex with γ_a value in $[-1,-1/2)$, and so on. We now formally analyze the algorithm.

▶ **Lemma 15.** *For even $j \geqslant 2$, the interval $[\ell_j, u_j]$ is*

$$\left[\frac{-(2^{j-1}-2)/3-1}{2^{j-1}}, \frac{-(2^{j-1}-2)/3}{2^{j-1}}\right].$$

For odd $j \geqslant 1$, the interval $[\ell_j, u_j]$ is

$$\left[\frac{-(2^{j-1}-1)/3-1}{2^{j-1}}, \frac{-(2^{j-1}-1)/3}{2^{j-1}}\right].$$

Proof. For $j = 1$ the interval is $[-1, 0]$ and $j = 2$ the interval is $[-1/2, 0]$. For odd j, we have

$$\ell_{j+1} = \frac{\ell_j + u_j}{2} = \frac{-(2^{j-1}-1)/3-1-(2^{j-1}-1)/3}{2^j} = \frac{-(2^j-2)/3-1}{2^j},$$

and

$$u_{j+1} = \frac{-(2^{j-1}-1)/3}{2^{j-1}} = \frac{-(2^j-2)/3}{2^j}.$$

For even j, we have

$$u_{j+1} = \frac{\ell_j + u_j}{2} = \frac{-(2^{j-1}-2)/3-1-(2^{j-1}-2)/3}{2^j} = \frac{-(2^j-1)/3}{2^j},$$

and

$$\ell_{j+1} = \frac{-(2^{j-1}-2)/3-1}{2^{j-1}} = \frac{-(2^j-4)/3-2}{2^j} = \frac{(-2^j+1-3)/3}{2^j} = \frac{-(2^j-1)/3-1}{2^j}.$$ ◀

As a consequence of Lemma 15, we immediately get a bound on the number of iterations in form of Corollary 16.

▶ **Corollary 16.** *For $j \geqslant \log\left(\frac{4}{3\varepsilon}\right)$, we have $I_j \subseteq [-1/3-\varepsilon, -1/3+\varepsilon]$.*

Proof. By Lemma 15 we have the following bounds on I_j. For even $j \geqslant 2$, the interval $I_j = [\ell_j, u_j]$ is

$$\left[-\frac{1}{3} - \frac{1}{3\cdot 2^{j-1}}, -\frac{1}{3} + \frac{2}{3\cdot 2^{j-1}}\right].$$

For odd $j \geqslant 1$, the interval $I_j = [\ell_j, u_j]$ is

$$\left[-\frac{1}{3} - \frac{2}{3\cdot 2^{j-1}}, -\frac{1}{3} + \frac{1}{3\cdot 2^{j-1}}\right].$$

Setting $\varepsilon = \frac{1}{3\cdot 2^{j-2}} = \frac{4}{3\cdot 2^j}$, we have $I_j \subseteq [-\frac{1}{3}-\varepsilon, -\frac{1}{3}+\varepsilon]$. Thus, $j = \log\left(\frac{4}{3\varepsilon}\right)$. ◀

In Lemma 17 we show that in each iteration Algorithm 1 colors an odd independent set. Lemma 17 also follows from Lemma 15.

▶ **Lemma 17.** *For each $j \geqslant 0$, let $H_j = (S_j, E_j)$ be a hypergraph with $E_j = \{e \in E : e \subseteq S_j\}$. Then for any $j \geqslant 0$, the set S_{j+1} is an odd independent set (i.e., we have $|S_{j+1} \cap e| \leqslant 1$ for any $e \in E_j$).*

Proof. Let $\{a,b,c\} \in E_j$. Suppose $a,b \in S_{j+1}$. If j is odd, then $\gamma_a, \gamma_b \in [\ell_j, \ell_{j+1})$. Therefore, we get $\gamma_a + \gamma_b < 2\ell_{j+1}$. This implies, by Observation 14, $\gamma_c > -1 - 2\ell_{j+1}$. Therefore, we have

$$\gamma_c > -1 - 2\ell_{j+1} = -1 - 2\left(\frac{-(2^j-2)/3 - 1}{2^j}\right) = -1 + 2\left(\frac{(2^j-2)/3+1}{2^j}\right)$$
$$= \frac{-3 \cdot 2^j + 2(2^j - 2) + 6}{3 \cdot 2^j} = \frac{-2^j + 2}{3 \cdot 2^j} = u_j,$$

which is a contradiction since $\gamma_c \in [\ell_j, u_j]$.

Similarly, if j is even, then $\gamma_a, \gamma_b \in (u_{j+1}, u_j]$ as $a, b \in S_{j+1}$. Therefore, we get $\gamma_a + \gamma_b > 2u_{j+1}$. This implies, by Observation 14, $\gamma_c < -1 - 2u_{j+1}$. Therefore, we have

$$\gamma_c < -1 - 2u_{j+1} = \frac{-3 \cdot 2^j + 2 \cdot 2^j - 2}{3 \cdot 2^j} = \frac{-2^j - 2}{3 \cdot 2^j} = \ell_j,$$

which is again a contradiction to the fact that $\gamma_c \in [\ell_j, u_j]$. ◀

Proof of Lemma 6. By Corollary 16, Algorithm 1 runs for $O\left(\log\left(\frac{1}{\varepsilon}\right)\right)$ iterations. In each iteration, it uses exactly one color, which yields the stated bound on the number of colors used. To show that the output coloring is a partial LO coloring, we apply Lemma 17, which states that each color corresponds to an odd independent set.

Now, we need to show that any edge with at least one colored vertex will have a unique maximum color. Consider such an edge $e = \{a,b,c\}$. If only one vertex in e is colored, then we are done. First, assume exactly two vertices in e (say a, b) were colored. Let a be colored in the j_a-th iteration and b be colored in the j_b-th iteration. Assume (without loss of generality) that $j_a \geqslant j_b$. Then, by Lemma 17 we have $j_a \neq j_b$ (i.e., $j_a > j_b$). As the color used in the iteration j is the j-th largest color in C (by a simple induction) color assigned to a is strictly larger than the color assigned to b. Finally, if all the vertices, a, b, c were colored at iterations $j_a \geqslant j_b \geqslant j_c$, respectively. Then, again by the same arguments we have $j_a > j_b$ and $j_a > j_c$, so the maximum color is assigned to only a. ◀

4 SDP Rounding for Balanced Hypergraphs

In this section we show that Algorithm 2 outputs an odd independent set in an ε-balanced 2-LO colorable 3-uniform hypergraph $H_B = (V_B, E_B)$. Thus, we will prove Lemma 13.

▶ **Lemma 13.** *Let $H = (V,E)$ be a $\frac{1}{|V|^{100}}$-balanced 2-LO colorable 3-uniform hypergraph $H = (V,E)$ with average degree at most Δ. Then there exists a (randomized) polynomial-time algorithm to compute an odd independent set of size at least $\Omega\left(\frac{|V|}{\Delta^{1/3}(\ln \Delta)^{3/2}}\right)$.*

Recall that we have a solution for the SDP 2. Let u_a be the unit vector along the component orthogonal to $\mathsf{v}_\emptyset$ (if the orthogonal component is zero, then we define u_a to be any arbitrarily chosen unit vector). Therefore,

$$\mathsf{u}_a = \frac{\mathsf{v}_a - \gamma_a \mathsf{v}_\emptyset}{\|\mathsf{v}_a - \gamma_a \mathsf{v}_\emptyset\|} = \frac{\mathsf{v}_a - \gamma_a \mathsf{v}_\emptyset}{\sqrt{\|\mathsf{v}_a\|^2 + \gamma_a^2 - 2\gamma_a \langle \mathsf{v}_a, \mathsf{v}_\emptyset \rangle}} = \frac{\mathsf{v}_a - \gamma_a \mathsf{v}_\emptyset}{\sqrt{1-\gamma_a^2}}. \tag{3}$$

Let function $\bar{\Phi} : \mathsf{R} \to [0,1]$ be defined as $\bar{\Phi}(t) \overset{\text{def}}{=} \mathbb{P}_{g \sim \mathcal{N}(0,1)}[g \geqslant t]$.

Algorithm 2 Randomized Rounding.

```
Input: $H_B$ a $\varepsilon$-balanced 2-LO colorable 3-uniform hypergraph and a parameter $\alpha$ (see Lemma 20 for values of $\varepsilon$ and $\alpha$ to be used).
Output: An odd independent set.
1. Let $t$ be such that $\alpha = \bar{\Phi}(t)$.
2. Sample $g \sim \mathcal{N}(0,1)^{|V_B|}$ and set $S(t) := \{a \in V_B : \langle \mathsf{u}_a, g\rangle \geqslant t\}$.
3. Set $S'(t) := S(t) \setminus \left( \bigcup_{\substack{e \in E_B \\ |e \cap S(t)| \geqslant 2}} e \right)$.
4. Output $S'(t)$.
```

In case of an edge $\{a,b,c\}$ with perfectly balanced vertices (i.e., if we have $\gamma_a = \gamma_b = \gamma_c = -1/3$), one can observe that the component orthogonal to $\mathsf{v}_\emptyset$ of the corresponding vectors sum to 0 (i.e., we have $\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c = 0$). In Lemma 18 we show a generalization of this observation for an ε-balanced hypergraph. Recall that in an ε-balanced hypergraph, we have $\gamma_a \in [-1/3 - \varepsilon, -1/3 + \varepsilon]$ for each vertex. The proof of the next lemma can be found in Appendix C.

▶ **Lemma 18.** *Let $\{a,b,c\}$ be an edge in an ε-balanced hypergraph H_B. Then $\|\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c\|^2 \leqslant 18\varepsilon$.*

When all the vertices in $\{a,b,c\}$ are perfectly balanced then the event that both a and b belong to $S(t)$ is equivalent to $\langle \mathsf{u}_c, g\rangle \leqslant -2t$ as $\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c = 0$. Therefore, we can use bounds on Gaussians to bound the probability of the aforementioned event. Again, Lemma 19 generalizes this to ε-balanced vector for small enough ε.

▶ **Lemma 19.** *Take $\varepsilon = \frac{1}{|V_B|^{100}}$ and let a, b be adjacent vertices in H_B. Then*

$$\mathbb{P}\left[a \in S(t) \wedge b \in S(t)\right] \leqslant \bar{\Phi}(2t) + \frac{2}{|V_B|^{25}}.$$

Proof. Suppose $e = \{a,b,c\}$ is an edge in H_B containing both a and b. If both a and b belong to $S(t)$, then $\langle \mathsf{u}_a, g\rangle \geqslant t$ and $\langle \mathsf{u}_b, g\rangle \geqslant t$. Note that $\|\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c\| \leqslant 3\sqrt{2\varepsilon}$ by Lemma 18. If we additionally assume that $\|g\| \leqslant |V_B|^{25}$ (this assumption is violated with low probability) we have

$$\begin{aligned} 3\sqrt{2\varepsilon}|V_B|^{25} &\geqslant \langle \mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c, g\rangle && \text{(Cauchy-Schwarz)} \\ &= \langle \mathsf{u}_a, g\rangle + \langle \mathsf{u}_b, g\rangle + \langle \mathsf{u}_c, g\rangle \\ &\geqslant 2t + \langle \mathsf{u}_c, g\rangle && (\langle \mathsf{u}_a, g\rangle \geqslant t \text{ and } \langle \mathsf{u}_b, g\rangle \geqslant t) \end{aligned}$$

we get $\langle \mathsf{u}_c, g\rangle \leqslant -2t + 3\sqrt{2\varepsilon}|V_B|^{25}$.
Thus, we can upper bound $\mathbb{P}\left[(\langle \mathsf{u}_a, g\rangle \geqslant t) \wedge (\langle \mathsf{u}_b, g\rangle \geqslant t) \wedge \left(\|g\| \leqslant |V_B|^{25}\right)\right]$ by

$$\begin{aligned} &\mathbb{P}\left[\left(\langle \mathsf{u}_c, g\rangle \leqslant -2t + 3\sqrt{2\varepsilon}|V_B|^{25}\right) \wedge \left(\|g\| \leqslant |V_B|^{25}\right)\right] \\ &\leqslant \mathbb{P}\left[\langle \mathsf{u}_c, g\rangle \leqslant -2t + 3\sqrt{2\varepsilon}|V_B|^{25}\right] \\ &= \bar{\Phi}\left(2t - 3\sqrt{2\varepsilon}|V_B|^{25}\right) \\ &\leqslant \bar{\Phi}(2t) + \sqrt{\varepsilon}|V_B|^{25} && \text{(Fact 21)} \\ &\leqslant \bar{\Phi}(2t) + \frac{1}{|V_B|^{25}}. && \left(\varepsilon = \frac{1}{|V_B|^{100}}\right) \end{aligned}$$

Now, in the following step we look at the case when the assumption $\|g\| \leqslant |V_B|^{25}$ is violated.

$$\begin{aligned}\mathbb{P}\left[(\langle \mathsf{u}_a, g\rangle \geqslant t) \wedge (\langle \mathsf{u}_b, g\rangle \geqslant t) \wedge \left(\|g\| > |V_B|^{25}\right)\right] &\leqslant \mathbb{P}\left[\|g\| > |V_B|^{25}\right]\\ &\leqslant \mathbb{P}\left[\|g\|^2 > |V_B|^{50}\right]\\ &\leqslant \frac{1}{|V_B|^{49}} \qquad \left(\mathsf{E}\left[\|g\|^2\right] = |V_B| \text{ and Markov bound}\right)\end{aligned}$$

Adding up the two disjoint cases we get the required bound. ◀

▶ **Lemma 20.** *Let $\Delta \geqslant 4$ be an upper bound on the average degree of a vertex in H_B (i.e., $|E_B| \leqslant \frac{\Delta|V_B|}{3}$). Take $\alpha = \frac{1}{32}\frac{1}{\Delta^{\frac{1}{3}}(\ln \Delta)^{1/2}}$ and $\varepsilon = \frac{1}{|V_B|^{100}}$. Then, we have $\mathsf{E}\left[|S'(t)|\right] \geqslant \frac{3}{4}\alpha\,|V_B|$.*

Proof. To lower bound the expected size of $S'(t)$ we lower bound the expected size of $S(t)$ and upper bound the expected number of vertices participating in a bad edge (i.e., an edge e such that $|e \cap S(t)| \geqslant 2$) separately.

First, we lower bound the size of $|S(t)|$ as follows.

$$\mathsf{E}\left[|S(t)|\right] = \sum_{a\in V_B} \mathbb{P}\left[\langle \mathsf{u}_a, g\rangle \geqslant t\right] = \alpha\,|V_B|\,.$$

Now, to get an upper bound we note that each bad edge can contribute at most 3 vertices in the total number of vertices participating in some bad edge. Formally, we have the following.

$$\begin{aligned}\left|\bigcup_{\substack{e\in E_B\\ |e\cap S(t)|\geqslant 2}} e\right| &\leqslant \sum_{\substack{e\in E_B\\ |e\cap S(t)|\geqslant 2}} |e| && \text{(Union Bound)}\\ &\leqslant 3\left|\{e \in E_B \text{ s.t. } \ |e\cap S(t)| \geqslant 2\}\right| && (|e| = 3)\end{aligned}$$

If an edge $\{a, b, c\}$ is bad, i.e., we have $|\{a, b, c\} \cap S(t)| \geqslant 2$, then either $\{a, b\} \subseteq S(t)$ or $\{a, c\} \subseteq S(t)$ or $\{b, c\} \subseteq S(t)$. Therefore, by union bound $\mathsf{E}\left[|\{e \in E_B \text{ s.t. } \ |e \cap S(t)| \geqslant 2\}|\right]$ is at most

$$\begin{aligned}&\sum_{\{a,b,c\}\in E_B} \Big(\mathbb{P}\left[a \in S(t) \wedge b \in S(t)\right] + \mathbb{P}\left[a \in S(t) \wedge c \in S(t)\right] + \mathbb{P}\left[c \in S(t) \wedge b \in S(t)\right]\Big)\\ &\leqslant \sum_{e\in E_B} 3\cdot\left(\bar{\Phi}\left(2t\right) + \frac{2}{|V_B|^{25}}\right)\\ &= 3|E_B|\cdot\bar{\Phi}\left(2t\right) + \frac{6|E_B|}{|V_B|^{25}},\end{aligned}$$

where the second inequality follows from Lemma 19. Let us now upper bound the first term as follows.

$$\begin{aligned}3|E_B|\bar{\Phi}\left(2t\right) &\leqslant \Delta|V_B|\cdot 512\bar{\Phi}\left(t\right)^4\cdot\left(\ln(1/\bar{\Phi}\left(t\right))\right)^{3/2} && \left(|E_B| \leqslant \frac{\Delta|V_B|}{3} \text{ and Corollary 25}\right)\\ &\leqslant \Delta|V_B|\cdot 512\alpha^4\cdot(\ln(1/\alpha))^{3/2} && (\bar{\Phi}\left(t\right) = \alpha)\\ &\leqslant \frac{1}{8}\alpha|V_B|\left(\frac{\ln(1/\alpha)}{4\ln\Delta}\right)^{3/2} && \text{(Substituting } \alpha)\\ &\leqslant \frac{1}{8}\alpha|V_B| && (\Delta \geqslant 4)\end{aligned}$$

Note that in the first inequality above we could use Corollary 25 as $\Delta \geqslant 4$ implies $t \geqslant 1$. It is easy to show that $\frac{6|E_B|}{|V_B|^{25}} \leqslant \frac{1}{8}\alpha|V_B|$. Therefore, we get

$$\mathsf{E}\left[|\{e \in E_B \text{ s.t. } |e \cap S(t)| \geqslant 2\}|\right] \leqslant \frac{1}{4}\alpha|V_B|.$$

Thus, by combining the two bounds we get that

$$\mathsf{E}\left[|S'(t)|\right] = \mathsf{E}\left[|S(t)|\right] - \mathsf{E}\left[\left|\bigcup_{\substack{e \in E_B \\ |e \cap S(t)| \geqslant 2}} e\right|\right] \geqslant \left(1 - \frac{1}{4}\right)\alpha\,|V_B| \geqslant \frac{3}{4}\alpha\,|V_B|\,. \qquad ◀$$

Proof of Lemma 13. This follows from Lemma 20 and the proof is standard Markov bound followed by an amplification argument where you repeat Algorithm 2 polynomially many times and choose the best odd independent set among all repetitions. The probability of even the best odd independent set not being of the required size is then inverse exponential with respect number of iterations. We refer the reader to Section 13.2 of [24] for further reference. ◀

A Better SDP Rounding

Here we note that there is in fact a better way to round the SDP in the balanced case, which follows from [16] and essentially reduces the balanced case to the unbalanced case. Let $H = (V, E)$ be an ε-balanced hypergraph on n vertices. Recall that $\varepsilon \leqslant 1/n^{100}$.

As in Algorithm 2, we sample a gaussian $g \sim \mathcal{N}(0,1)^n$. For each (unit) vector u_a for $a \in V$, let $\zeta_a = \langle \mathsf{u}_a, g\rangle$. Observe that $|\zeta_a| \in [1/n^2, n^2]$ with probability $1 - O(1/n^2)$. Now for all $a \in V$, set $\zeta'_a = \zeta_a/n^2$ and set $\gamma'_a = \gamma_a + \zeta'_a$. Thus, with probability at least (roughly) $1 - O(1/n)$, for *all* vertices $a \in V$, we have

$$|\zeta'_a| \in [1/n^4, 1] \quad \text{and} \quad \gamma'_a \notin (-1/3 - 1/n^{100}, -1/3 + 1/n^{100}).$$

Since for every hyperedge $\{a, b, c\} \in E$, we have $\zeta'_a + \zeta'_b + \zeta'_c = 0$ (because $\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c = \mathbf{0}$, which implies $\langle \mathsf{u}_a, g\rangle + \langle \mathsf{u}_b, g\rangle + \langle \mathsf{u}_c, g\rangle = 0$).

Then for every hyperedge $\{a, b, c\} \in E$, we have $\gamma'_a + \gamma'_b + \gamma'_c = -1$. Thus, we can run Algorithm 1 on the inputs $\{\gamma'_a\}_{a \in V}$ and $\varepsilon = 1/n^{100}$. By Lemma 6, it will output an LO-coloring of H using at most $O(\log \frac{1}{\varepsilon})$ colors.

5 Conclusion

We have presented an improved bound on the number of colors needed to efficiently LO color a 2-LO colorable 3-uniform hypergraph, and demonstrated that SDP-based rounding methods can indeed be applied to LO coloring. A natural question is if we can do better than $O(\log n)$ colors in the balanced case; this might be a step towards improving on the bound of $O(\log n)$ colors for the general case given in [16].

References

1 Noga Alon, Pierre Kelsen, Sanjeev Mahajan, and Hariharan Ramesh. Coloring 2-colorable hypergraphs with a sublinear number of colors. *Nordic Journal of Computing*, 3:425–439, 1996.

2 Sanjeev Arora, Eden Chlamtac, and Moses Charikar. New approximation guarantee for chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 215–224, 2006.

3 Libor Barto, Diego Battistelli, and Kevin M. Berg. Symmetric promise constraint satisfaction problems: Beyond the Boolean case. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 187 of *LIPIcs*, pages 10:1–10:16, 2021. doi:10.4230/LIPICS.STACS.2021.10.

4 Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Oprsal. Algebraic approach to promise constraint satisfaction. *Journal of the ACM*, 68(4):28:1–28:66, 2021. doi:10.1145/3457606.

5 Avrim Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41(3):470–516, 1994. doi:10.1145/176584.176586.

6 Avrim Blum and David R. Karger. An $O(n^{3/14})$-coloring algorithm for 3-colorable graphs. *Information Processing Letters*, 61(1):49–53, 1997. doi:10.1016/S0020-0190(96)00190-1.

7 Joshua Brakensiek and Venkatesan Guruswami. Promise constraint satisfaction: Algebraic structure and a symmetric Boolean dichotomy. *SIAM Journal on Computing*, 50(6):1663–1700, 2021. doi:10.1137/19M128212X.

8 Joshua Brakensiek, Venkatesan Guruswami, and Sai Sandeep. SDPs and robust satisfiability of promise CSP. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 609–622, 2023. doi:10.1145/3564246.3585180.

9 Panagiotis Cheilaris and Géza Tóth. Graph unique-maximum and conflict-free colorings. *Journal of Discrete Algorithms*, 9(3):241–251, 2011. doi:10.1016/J.JDA.2011.03.005.

10 Hui Chen and Alan Frieze. Coloring bipartite hypergraphs. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 345–358, 1996. doi:10.1007/3-540-61310-2_26.

11 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 344–353, 2006. doi:10.1145/1132516.1132567.

12 Irit Dinur, Oded Regev, and Clifford D. Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005. doi:10.1007/S00493-005-0032-4.

13 Miron Ficak, Marcin Kozik, Miroslav Olsák, and Szymon Stankiewicz. Dichotomy for symmetric Boolean PCSPs. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 57:1–57:12, 2019. doi:10.4230/LIPICS.ICALP.2019.57.

14 Marek Filakovský, Tamio-Vesa Nakajima, Jakub Opršal, Gianluca Tasinato, and Uli Wagner. Hardness of linearly ordered 4-colouring of 3-colourable 3-uniform hypergraphs. In *41st International Symposium on Theoretical Aspects of Computer Science, (STACS)*, volume 289 of *LIPIcs*, pages 34:1–34:19, 2024. doi:10.4230/LIPICS.STACS.2024.34.

15 Magnús M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. *J. Graph Algorithms Appl.*, 4(1):1–16, 2000. doi:10.7155/JGAA.00020.

16 Johan Håstad, Björn Martinsson, Tamio-Vesa Nakajima, and Stanislav Živný. A logarithmic approximation of linearly-ordered colourings. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London School of Economics, London, UK*, volume 317 of *LIPIcs*, pages 7:1–7:6. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.APPROX/RANDOM.2024.7.

17 David R. Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998. doi:10.1145/274787.274791.

18 Meir Katchalski, William McCuaig, and Suzanne Seager. Ordered colourings. *Discrete Mathematics*, 1(142):141–154, 1995. doi:10.1016/0012-365X(93)E0216-Q.

19 Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *Journal of the ACM*, 64(1):4:1–4:23, 2017. doi:10.1145/3001582.

20 Michael Krivelevich, Ram Nathaniel, and Benny Sudakov. Approximating coloring and maximum independent sets in 3-uniform hypergraphs. *Journal of Algorithms*, 41(1):99–113, 2001. doi:10.1006/jagm.2001.1173.

21 Anand Louis, Alantha Newman, and Arka Ray. Improved linearly ordered colorings of hypergraphs via SDP rounding. *CoRR*, abs/2405.00427, 2024. doi:10.48550/arXiv.2405.00427.

22 Tamio-Vesa Nakajima and Stanislav Živný. Linearly ordered colourings of hypergraphs. *ACM Trans. Comput. Theory*, 14(3-4):1–19, 2022. doi:10.1145/3570909.

23 Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, 1983. doi:10.1145/2157.2158.

24 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

25 Marcin Wrochna and Stanislav Živný. Improved hardness for H-colourings of G-colourable graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 1426–1435, 2020. doi:10.1137/1.9781611975994.86.

A Properties of Gaussian

Let function $\Phi : \mathsf{R} \to [0,1]$ be defined as $\Phi(t) \overset{\text{def}}{=} \mathbb{P}_{g\sim\mathcal{N}(0,1)}[g \leqslant t]$, and let function $\bar{\Phi} : \mathsf{R} \to [0,1]$ be defined as $\bar{\Phi}(t) \overset{\text{def}}{=} \mathbb{P}_{g\sim\mathcal{N}(0,1)}[g \geqslant t]$.

▶ **Fact 21.** *For any $a \leqslant b$, we have $\bar{\Phi}(b) - \bar{\Phi}(a) = \mathbb{P}_{g\sim\mathcal{N}(0,1)}[g \in [a,b]] \leqslant \frac{b-a}{\sqrt{2\pi}}$.*

Proof. The statement follows from the following computations.

$$\mathbb{P}[g \in [a,b]] = \int_a^b \frac{e^{-x^2/2}}{\sqrt{2\pi}}\,dx \leqslant \frac{1}{\sqrt{2\pi}} \int_a^b \sup_{y\in\mathbb{R}} e^{-y^2/2}\,dx = \frac{b-a}{\sqrt{2\pi}}. \qquad ◀$$

▶ **Fact 22** (Folklore)**.** *For every $t > 0$,*

$$\frac{t}{\sqrt{2\pi}(t^2+1)} e^{-\frac{1}{2}t^2} < \bar{\Phi}(t) < \frac{1}{\sqrt{2\pi}t} e^{-\frac{1}{2}t^2}.$$

▶ **Corollary 23** (Folklore)**.** *Fix $t \geqslant 1$ and let $\beta = \bar{\Phi}(t)$. Then we have*

$$\sqrt{2\ln\frac{1}{\beta} - \ln\ln\frac{1}{\beta} - \ln 16\pi} \leqslant t \leqslant \sqrt{2\ln\frac{1}{\beta} - \ln\ln\frac{1}{\beta}} \leqslant \sqrt{2\ln\frac{1}{\beta}}.$$

In fact, $t < \sqrt{2\ln\frac{1}{\beta}}$ holds even if $t \in (0,1)$.

Proof. Let $t > 0$ (note here we allow $t \in (0,1)$) and let $\beta = \bar{\Phi}(t)$. By taking logarithm and multiplying by -2, the inequalities in Fact 22 imply

$$2\ln\left(\frac{1}{\beta}\right) > t^2 + 2\ln\left(\sqrt{2\pi}t\right), \tag{4}$$

$$2\ln\left(\frac{1}{\beta}\right) < t^2 + 2\ln\left(\sqrt{2\pi}\left(\frac{t^2+1}{t}\right)\right). \tag{5}$$

We can now use (4) to get

$$2\ln\left(\frac{1}{\beta}\right) > t^2 + 2\ln\left(\sqrt{2\pi}t\right) \geqslant t^2.$$

Hence, we have $t < \sqrt{2\ln\frac{1}{\beta}}$ for any $t > 0$. Again, by multiplying by $\frac{1}{2}$ and taking logarithms, the (4), (5) imply

$$\ln\ln\left(\frac{1}{\beta}\right) > \ln\left(t^2/2 + \ln\left(\sqrt{2\pi}t\right)\right), \tag{6}$$

$$\ln\ln\left(\frac{1}{\beta}\right) < \ln\left(t^2/2 + \ln\left(\sqrt{2\pi}\left(\frac{t^2+1}{t}\right)\right)\right). \tag{7}$$

From hereon we assume $t \geqslant 1$. (4) $-$ (7) gives us

$$2\ln\left(\frac{1}{\beta}\right) - \ln\ln\left(\frac{1}{\beta}\right) > t^2 + \ln\left(\frac{2\pi t^2}{t^2/2 + \ln\left(\sqrt{2\pi}\left(\frac{t^2+1}{t}\right)\right)}\right) = t^2 + \ln\left(\frac{4\pi t^2}{t^2 + 2\ln\left(\sqrt{2\pi}\left(\frac{t^2+1}{t}\right)\right)}\right) \quad (8)$$

▷ Claim 24. $4\pi t^2 \geqslant t^2 + 2\ln\left(\sqrt{2\pi}\left(\frac{t^2+1}{t}\right)\right)$.

Proof. Note that the above inequality is equivalent to $\left(\frac{4\pi-1}{2}\right)t^2 \geqslant \ln\sqrt{2\pi} + \ln\left(t + \frac{1}{t}\right)$. Indeed we have

$$\begin{aligned} \ln\sqrt{2\pi} + \ln\left(t + \frac{1}{t}\right) &\leqslant \ln\sqrt{2\pi} + \ln(t+1) && (t \geqslant 1) \\ &\leqslant \ln\sqrt{2\pi} + t && (\ln(1+x) \leqslant x) \\ &\leqslant \ln\sqrt{2\pi} + t^2 && (t \geqslant 1) \\ &\leqslant \left(\frac{4\pi-3}{2}\right) + t^2 && \left(\frac{4\pi-3}{2} \geqslant \ln\sqrt{2\pi}\right) \\ &\leqslant \left(\frac{4\pi-1}{2}\right)t^2 && (t \geqslant 1) \end{aligned}$$

◁

Using this claim and (8) we get

$$t^2 \leqslant 2\ln\frac{1}{\beta} - \ln\ln\frac{1}{\beta}.$$

Hence, we have $t \leqslant \sqrt{2\ln\frac{1}{\beta} - \ln\ln\frac{1}{\beta}}$. For the remaining inequality, we again see that (5) $-$ (6) gives us

$$\begin{aligned} 2\ln\frac{1}{\beta} - \ln\ln\frac{1}{\beta} &< t^2 + \ln\left(\frac{4\pi\left(t + \frac{1}{t}\right)}{t^2 + 2\ln(\sqrt{2\pi}t)}\right) \\ &\leqslant t^2 + \ln 4\pi + \ln\left(\frac{(t+1)^2}{t^2}\right) && (t \geqslant 1) \\ &\leqslant t^2 + \ln 4\pi + 2\ln\left(1 + \frac{1}{t}\right) \\ &\leqslant t^2 + \ln 4\pi + 2\ln 2 = t^2 + \ln 16\pi && (t \geqslant 1) \end{aligned}$$

$\sqrt{2\ln\frac{1}{\beta} - \ln\ln\frac{1}{\beta} - \ln 16\pi} \leqslant t$ follows from the above inequality. Hence, we have all the required inequalities. ◀

▶ **Corollary 25** (Folklore). *Fix $t \geqslant 1$. Then, we have*

$$\bar{\Phi}(2t) \leqslant 512\left(\ln\left(\frac{1}{\bar{\Phi}(t)}\right)\right)^{3/2}\bar{\Phi}(t)^4.$$

Proof. For any $t \geqslant 1$ and $\delta \in (0,1)$ the following holds.

$$\begin{aligned}
\bar{\Phi}(2t) &\leqslant \frac{1}{2\sqrt{2\pi}t}e^{-2t^2} && \text{(Fact 22)}\\
&\leqslant \frac{1}{2\sqrt{2\pi}t} \cdot \frac{(2\pi)^2\left(t^2+1\right)^4}{t^4} \cdot \bar{\Phi}(t)^4 && \left(\frac{t}{\sqrt{2\pi}(t^2+1)}e^{-\frac{1}{2}t^2} \leqslant \bar{\Phi}(t) \text{ by Fact 22}\right)\\
&= (2\pi)^{3/2}\frac{1}{2t}\left(t+\frac{1}{t}\right)^4 \bar{\Phi}(t)^4 \\
&\leqslant (2\pi)^{3/2}(2t)^3\bar{\Phi}(t)^4 && (t \geqslant 1)\\
&\leqslant (4\sqrt{\pi})^3\left(\ln\left(\frac{1}{\bar{\Phi}(t)}\right)\right)^3 \cdot \bar{\Phi}(t)^4 && \text{(by Corollary 23)}\\
&\leqslant 512\left(\ln\left(\frac{1}{\bar{\Phi}(t)}\right)\right)^3 \bar{\Phi}(t)^4 && (\sqrt{\pi} \leqslant 2).
\end{aligned}$$

◀

B Coloring of 2-LO Colorable 3-Uniform Hypergraphs

In this section, we show how to color a 2-LO colorable 3-uniform hypergraph with 2 colors. For simplicity, we define *balanced* vertices to be those with $\gamma_a = -1/3$. It is straightforward to extend to the case of ε-balanced for small but (strictly) positive ε.

Algorithm 3 2-Coloring Algorithm.

```
Input: A solution to SDP 2.
Two-Sided Combinatorial Rounding
 1. Set $S_l := \{a \in V \mid \gamma_a < -\frac{1}{3}\}$, $S_r := \{a \in V \mid \gamma_a > -\frac{1}{3}\}$, and $S_b := \{a \in V \mid \gamma_a = -\frac{1}{3}\}$.
 2. Color $S_l$ using color 1 and $S_r$ using color 2.
Two-Sided Hyperplane Rounding
resume Choose a uniformly random unit vector $r$ over the sphere.
resume Set $H_r := \{a \in S_b \mid \langle r, \mathsf{u}_a\rangle \geqslant 0\}$ and $H_l := \{a \in S_b \mid \langle r, \mathsf{u}_a\rangle < 0\}$.
resume Color $H_l$ using 1 and $H_r$ using 2.
```

▶ **Lemma 26.** *Let S_l, S_r be as defined in Algorithm 3. Then, for any edge $e \in E$, we have $S_l \cap e \neq \emptyset$ if and only if $S_r \cap e \neq \emptyset$.*

Proof. Fix an edge $\{a,b,c\} = e \in E$. Observation 14 states that $\gamma_a + \gamma_b + \gamma_c = -1$ Suppose that $e \cap S_l \neq \emptyset$ and $e \subseteq S_l \cup S_b$. Then, we get $\gamma_a + \gamma_b + \gamma_c < -1$, a contradiction. Hence, $e \cap S_l \neq \emptyset$ implies $e \cap S_r \neq \emptyset$ The proof of the converse is similar. ◀

▶ **Lemma 27.** *Let r be unit vector distributed uniformly over the sphere and let H_l, and H_r be defined as in Algorithm 3. Then, for any edge $e \in E$, we have $|H_l \cap e| \leqslant 2$ and $|H_r \cap e| \leqslant 2$ (with probability 1).*

Proof. We can assume that $H_r = \{a \in S_b | \langle r, \mathsf{u}_a\rangle > 0\}$, as this is true with probability 1. Fix any $e \in E$. Assume that $\{a,b,c\} = e \subseteq S_b$; otherwise, we are done. Using Lemma 18 with $\varepsilon = 0$ we get $\|\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c\|^2 = 0$, which implies $\mathsf{u}_a + \mathsf{u}_b + \mathsf{u}_c = 0$. Therefore, we have $\langle r, \mathsf{u}_a\rangle + \langle r, \mathsf{u}_b\rangle + \langle r, \mathsf{u}_c\rangle = 0$. But, if $e \subseteq H_l$, then we have $\langle r, \mathsf{u}_a\rangle + \langle r, \mathsf{u}_b\rangle + \langle r, \mathsf{u}_c\rangle < 0$, a contradiction. Similarly, if $e \subseteq H_r$, then we have a contradiction in form of $\langle r, \mathsf{u}_a\rangle + \langle r, \mathsf{u}_b\rangle + \langle r, \mathsf{u}_c\rangle > 0$. ◀

▶ **Theorem 28.** *There is a polynomial-time algorithm that, if given a 2-LO colorable 3-uniform hypergraph H with n vertices, finds an 2-coloring of H.*

Proof. Solve SDP 2 and use Algorithm 3. Consider any edge $e = \{a, b, c\}$. If e is not completely contained in S_b, then by Lemma 26 we at least one vertex each from S_l and S_r; hence, e is non-monochromatic. Otherwise, e is non-monochromatic by Lemma 27. ◀

C Omitted Proofs

Proof of Lemma 18

Before we proceed to prove Lemma 18 we need the following lemma.

▶ **Lemma 29.** *Let $\{a, b, c\} \in E$ and $\gamma_a = -1/3 + \varepsilon_a$, $\gamma_b = -1/3 + \varepsilon_b$, $\gamma_c = -1/3 + \varepsilon_c$. Then the following hold.*

1. $\varepsilon_a + \varepsilon_b + \varepsilon_c = 0$.
2. $\langle \mathsf{v}_a, \mathsf{v}_b \rangle = -1/3 + \varepsilon_c$, $\langle \mathsf{v}_b, \mathsf{v}_c \rangle = -1/3 + \varepsilon_a$, *and* $\langle \mathsf{v}_c, \mathsf{v}_a \rangle = -1/3 + \varepsilon_b$.

Proof. Using Observation 14 we get

$$-1/3 + \varepsilon_a - 1/3 + \varepsilon_b - 1/3 + \varepsilon_c = -1,$$

which implies $\varepsilon_a + \varepsilon_b + \varepsilon_c = 0$. Taking inner products with $\mathsf{v}_a, \mathsf{v}_b, \mathsf{v}_c$ on both sides of constraint (1) of SDP 2 we get

$$\begin{aligned}
1 + \langle \mathsf{v}_a, \mathsf{v}_b \rangle + \langle \mathsf{v}_a, \mathsf{v}_c \rangle &= 1/3 - \varepsilon_a, \\
\langle \mathsf{v}_b, \mathsf{v}_a \rangle + 1 + \langle \mathsf{v}_b, \mathsf{v}_c \rangle &= 1/3 - \varepsilon_b, \\
\langle \mathsf{v}_c, \mathsf{v}_a \rangle + \langle \mathsf{v}_c, \mathsf{v}_b \rangle + 1 &= 1/3 - \varepsilon_c,
\end{aligned}$$

which imply

$$\langle \mathsf{v}_a, \mathsf{v}_b \rangle + \langle \mathsf{v}_a, \mathsf{v}_c \rangle = -(2/3 + \varepsilon_a), \tag{9}$$

$$\langle \mathsf{v}_b, \mathsf{v}_a \rangle + \langle \mathsf{v}_b, \mathsf{v}_c \rangle = -(2/3 + \varepsilon_b), \tag{10}$$

$$\langle \mathsf{v}_c, \mathsf{v}_a \rangle + \langle \mathsf{v}_c, \mathsf{v}_b \rangle = -(2/3 + \varepsilon_c). \tag{11}$$

(9)+(10)−(11) gives us

$$2 \langle \mathsf{v}_a, \mathsf{v}_b \rangle = -2/3 - \varepsilon_a - \varepsilon_b + \varepsilon_c.$$

Using Item 1 of this lemma and dividing by 2 we get $\langle \mathsf{v}_a, \mathsf{v}_b \rangle = -1/3 + \varepsilon_c$ as needed. Similarly, we get $\langle \mathsf{v}_a, \mathsf{v}_c \rangle = -1/3 + \varepsilon_b$, $\langle \mathsf{v}_c, \mathsf{v}_b \rangle_c = -1/3 + \varepsilon_a$. ◀

Proof of Lemma 18. Note that

$$\begin{aligned}
\langle \mathsf{u}_a, \mathsf{u}_b \rangle &= \left\langle \frac{\mathsf{v}_a - \gamma_a \mathsf{v}_\emptyset}{\sqrt{1 - \gamma_a^2}}, \frac{\mathsf{v}_b - \gamma_b \mathsf{v}_\emptyset}{\sqrt{1 - \gamma_b^2}} \right\rangle = \frac{\langle \mathsf{v}_a, \mathsf{v}_b \rangle - \gamma_a \langle \mathsf{v}_\emptyset, \mathsf{v}_b \rangle - \gamma_b \langle \mathsf{v}_\emptyset, \mathsf{v}_a \rangle + \gamma_a \gamma_b \langle \mathsf{v}_\emptyset, \mathsf{v}_\emptyset \rangle}{\sqrt{(1 - \gamma_a^2)(1 - \gamma_b^2)}} \\
&= \frac{\langle \mathsf{v}_a, \mathsf{v}_b \rangle - \gamma_a \gamma_b}{\sqrt{(1 - \gamma_a^2)(1 - \gamma_b^2)}}.
\end{aligned}$$

First let us upper-bound the denominator in the above expression using $\gamma_a, \gamma_b \in [-1/3 - \epsilon, -1/3 + \epsilon]$ as follows.

$$\begin{aligned}\sqrt{(1-\gamma_a^2)\,(1-\gamma_b^2)} &\leqslant \sqrt{(1-(1/3-\epsilon)^2)(1-(1/3-\epsilon)^2)}\\ &= \frac{8}{9} + \frac{2\epsilon}{3} - \epsilon^2\\ &\leqslant \frac{8}{9} + \frac{2\epsilon}{3}.\end{aligned}$$

This implies that

$$\begin{aligned}\frac{1}{\sqrt{(1-\gamma_a^2)\,(1-\gamma_b^2)}} &\geqslant \frac{1}{\frac{8}{9}\left(1+\frac{9\epsilon}{4}\right)}\\ &\geqslant \frac{9}{8}\left(1-\frac{9\epsilon}{4}+\frac{\left(\frac{9\epsilon}{4}\right)^2}{\left(1+\frac{9\epsilon}{4}\right)}\right)\\ &\geqslant \frac{9}{8}\left(1-\frac{9\epsilon}{4}\right).\end{aligned}$$

By Lemma 29, we have $\langle \mathsf{v}_a, \mathsf{v}_b\rangle \in [-1/3-\epsilon, -1/3+\epsilon]$. So, we can also bound the numerator in the expression for $\langle \mathsf{u}_a, \mathsf{u}_b\rangle$ by using the fact that $\langle \mathsf{v}_a, \mathsf{v}_b\rangle, \gamma_a, \gamma_b \in [-1/3-\epsilon, -1/3+\epsilon]$ as follows.

$$\begin{aligned}\langle \mathsf{v}_a, \mathsf{v}_b\rangle - \gamma_a\gamma_b &\leqslant -\frac{1}{3} + \epsilon - \left(\frac{1}{3}-\epsilon\right)^2\\ &= -\frac{4}{9} + \frac{5\epsilon}{3} - \epsilon^2\\ &\leqslant -\frac{4}{9} + \frac{5\epsilon}{3}.\end{aligned}$$

Therefore, we get

$$\begin{aligned}\langle \mathsf{u}_a, \mathsf{u}_b\rangle &\leqslant -\frac{4}{9}\left(1-\frac{15\epsilon}{4}\right)\cdot\frac{9}{8}\left(1-\frac{9\epsilon}{4}\right)\\ &\leqslant -\frac{1}{2}\left(1-6\epsilon\right).\end{aligned}$$

Finally, for the edge $\{a, b, c\}$ we get

$$\begin{aligned}\|\mathsf{u}_a+\mathsf{u}_b+\mathsf{u}_c\|^2 &= \|\mathsf{u}_a\|^2 + \|\mathsf{u}_b\|^2 + \|\mathsf{u}_c\|^2 + 2\langle \mathsf{u}_a, \mathsf{u}_b\rangle + 2\langle \mathsf{u}_b, \mathsf{u}_c\rangle + 2\langle \mathsf{u}_c, \mathsf{u}_a\rangle\\ &= 3 + 2\left(\langle \mathsf{u}_a, \mathsf{u}_b\rangle + \langle \mathsf{u}_b, \mathsf{u}_c\rangle + \langle \mathsf{u}_c, \mathsf{u}_a\rangle\right)\\ &\leqslant 18\varepsilon\end{aligned}$$

where the last inequality follows from the fact that $\langle \mathsf{u}_a, \mathsf{u}_b\rangle$, $\langle \mathsf{u}_b, \mathsf{u}_c\rangle$, and $\langle \mathsf{u}_c, \mathsf{u}_a\rangle$ are all at most $-\frac{1}{2}+3\varepsilon$. ◀

Parameterized Algorithms and Hardness for the Maximum Edge q-Coloring Problem

Rogers Mathew
Department of Computer Science and Engineering, IIT Hyderabad, India

Fahad Panolan
School of Computer Science, University of Leeds, UK

Seshikanth
Department of Computer Science and Engineering, IIT Hyderabad, India

Abstract

An *edge q-coloring* of a graph G is a coloring of its edges such that every vertex sees at most q colors on the edges incident on it. The *size* of an edge q-coloring is the total number of colors used in the coloring. Given a graph G and a positive integer t, the Maximum Edge q-Coloring problem is about whether G has an edge q-coloring of size t. Studies on this coloring problem were motivated by its application in the channel assignment problem in wireless networks.

Goyal, Kamat, and Misra (MFCS 2013) studied Maximum Edge 2-Coloring from the perspective of parameterized complexity. Given a graph on n vertices, they considered the standard parameter t, the number of colors in an optimal edge 2-coloring, and the dual parameter ℓ, where $n-\ell$ is the number of colors in an optimal edge 2-coloring. They designed FPT algorithms for Maximum Edge 2-Coloring parameterized by t and ℓ. In this paper, we revisit and study Maximum Edge 2-Coloring from the perspective of parameterized complexity and show the following results.

1. Let $\gamma(G)$ denote the maximum matching size in a given graph G. It is easy to see that a maximum edge 2-coloring of G is of size at least $\gamma(G)$. Goyal, Kamat, and Misra (MFCS 2013) had asked if there exists an FPT algorithm for Maximum Edge 2-Coloring parameterized by k, where $k :=$ (size of a maximum edge 2-coloring of G) $-\gamma(G)$. We show that Maximum Edge 2-Coloring parameterized by k is W[1] hard.
2. On the positive side, we show that there is an algorithm that, given a graph G on n vertices and a tree decomposition of width tw, runs in time $2^{O(q\mathsf{tw}\log q\mathsf{tw})}n$ and outputs a maximum edge q-coloring of G.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases FPT algorithm, Edge coloring, Treewidth, W[1]-hardness

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.31

1 Introduction

Given a graph G and a positive integer q, an *edge q-coloring* is a coloring (not necessarily proper) of the edges of G such that every vertex sees at most q colors on the edges incident on it. The *size of an edge q-coloring* is the total number of colors used in the coloring. Assigning every edge of G the same color is indeed a valid edge q-coloring. However, we are interested in obtaining a *maximum edge q-coloring* of G, an edge q-coloring of G of the maximum possible size.

In 2005, Raniwala, Chiueh, and Gopalan in [12] and [11] proposed *Hyacinth*, a multichannel wireless mesh network architecture that uses 802.11 Network Interface Cards (NICs) at each node of the mesh network. They observed that two NICs on each node may improve network throughput by a factor of 6 to 7 compared to conventional single-channel ad hoc networks. A network can be modeled as a graph where each computer is a vertex. Assigning channels

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 31; pp. 31:1–31:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to computers with two interface cards corresponds to an edge 2-coloring of the graph. The maximum number of colors in an edge 2-coloring represents the number of channels that can be used simultaneously in the network.

Maximum edge q-coloring of graphs has been studied from an algorithmic as well as a structural graph theoretic perspective. Adamaszek and Popa [1] proved that the decision version of the maximum edge q-coloring problem is NP-hard for every $q \geq 2$. Further they showed that, for every $q \geq 2$, the maximum edge q-coloring problem is APX-hard. For $q = 2$, the paper gives a 5/3-factor approximation algorithm for graphs having a perfect matching. For triangle-free graphs having a perfect matching, Chandran et al. [3] gave an 8/5-factor approximation algorithm for the maximum edge q-coloring problem. Later, in [2], Chandran et al. showed that the approximation factors of 5/3 and 8/5 can be improved under certain assumptions on the minimum degree of the graph under consideration. As for general graphs, Feng et al. in [8] gave a 2-factor approximation algorithm for the maximum edge 2-coloring problem. In the same paper, the authors showed that the maximum edge 2-coloring problem is polynomial-time solvable for trees and complete graphs. In [6], Dvorak et al. show that there is a PTAS known for the maximum edge q-coloring problem on minor-free graphs.

From a parameterized complexity perspective, Goyal, Kamat, and Misra in [10] gave fixed-parameter tractable algorithms for maximum edge 2-coloring of G for both the standard parameter (say t, if t is the size of an optimal edge 2-coloring of G) and the dual parameter (say ℓ, if $n - \ell$ is the size of an optimal edge 2-coloring, where n is the number of vertices in the input graph). It is known that the maximum number of colors used in an edge 2-coloring of a graph G is at most the number of vertices in G, and hence, with the dual parameter ℓ, the number of colors asked for is $n - \ell$. For the dual parameterization, the authors obtained a linear vertex kernel with $O(\ell)$ vertices and $O(\ell^2)$ edges.

The maximum edge q-coloring problem is related to another parameter called anti-Ramsey number, a concept introduced by Erdős, Simonovitz and Sós in 1975 [7]. Given a host graph G and a pattern graph H, the *anti-Ramsey number* $ar(G, H)$ is defined as the smallest positive integer k such that any coloring of the edges of G with k colors will have a rainbow subgraph (a subgraph no two of whose edges have the same color) isomorphic to H. In other words, $ar(G, H)$ is one more than the largest k for which there exists a coloring of the edges of G with k colors such that there is no rainbow subgraph isomorphic to H under this coloring. See [9] for a survey on anti-Ramsey numbers, a notion that has been extensively studied in extremal graph theory. From its definition, it is clear that the size of a maximum edge q-coloring of a graph G is one less than $ar(G, K_{1,q+1})$, where $K_{1,q+1}$ denotes the complete bipartite graph with 1 vertex in one part and $q + 1$ vertices in the other.

Our contributions

In Section 3 we give a fixed parameter tractable algorithm for Maximum Edge q-Coloring parameterized by the treewidth of the graph under consideration.

▶ **Theorem 1.** *There is an algorithm that, given an n-vertex graph G and its tree decomposition of width* tw, *runs in time $2^{O(\mathsf{tw}\cdot q \log(\mathsf{tw}\cdot q))}n$, and outputs a maximum edge q-coloring of G.*

To explain the significance of this result, let us recall the work of Goyal et al. [10] for Maximum Edge 2-Coloring with respect to the parameter k, the number of colors in the output edge 2-coloring of the graph. Let G be the input graph, and let $\gamma(G)$ denote the size of a maximum matching in G. They first observe that the size of a maximum edge 2-coloring of G is at least $\gamma(G)$ as coloring all the edges in a maximum matching with distinct colors and then coloring the remaining edges with a new color is indeed a valid edge 2-coloring of G.

Thus, the problem becomes challenging only when $\gamma(G) \leq k$. In this case, the vertex cover number of G is at most $2k$. Then, Goyal et al. [10] observed that maximum edge q-coloring can be expressed in Monadic Second Order (MSO_2) logic. Since the treewidth of G is at most its vertex cover number, which is upper bounded by $2k$, an application of Courcelle's theorem [4] provides a fixed parameter tractable (FPT) algorithm for the problem, but its running time will be impractical. Goyal et al. [10] gave a combinatorial algorithm with running time $2^{O(k \log k)} n^{O(1)}$, using the fact that the vertex cover number will be at most $2k$. As a corollary, one gets an FPT algorithm parameterized by the vertex cover number because the number of colors in an edge 2-coloring is at most two times the vertex cover number.

However, the case where the parameter is treewidth is not trivial. First of all, the edge 2-coloring number can be arbitrarily larger than the treewidth. Consider the graph, which is a path on n vertices. Here, the pathwidth (and hence the treewidth) is one, but coloring all the $n-1$ edges with distinct colors is a valid edge 2-coloring. Next, it is tempting to believe that since the problem is expressible in MSO_2, we get an FPT algorithm parameterized by treewidth. In fact, there is a caveat here. The length of the MSO_2 formula depends on the total number of colors used in the edge coloring (hence it is large), and the running time of the algorithm by the application of Courcelle's theorem depends exponentially on the length of the formula as well. Thus, it is important to design an algorithm for the problem when the parameter is treewidth. To the best of our knowledge, this is the first FPT algorithm for the problem when parameterized by treewidth.

Recall the the size of a maximum edge 2-coloring is at least $\gamma(G)$. Goyal et al. [10] asked if there exists an FPT algorithm for the maximum edge 2-coloring problem parameterized by k, where $k :=$ (size of a maximum edge 2-coloring of $G) - \gamma(G)$. In Section 4, we resolve this question by showing that the problem is W[1] hard.

▶ **Theorem 2.** *It is $W[1]$-hard parameterized by k to decide if the given graph G has an edge 2-coloring using at least $\gamma(G) + k$ colors.*

2 Preliminaries

We use $\mathbb{N}$ to denote the set of natural numbers. For $n \in \mathbb{N}$, we use $[n]$ to denote $\{1, \ldots, n\}$. For a function $f : A \to B$ and $A' \subseteq A$, $f(A') = \{f(a) : a \in A'\}$. For two functions $f : A \to B$ and $g : B \to C$, $g \circ f$ is the function from A to C defined as follows: $(g \circ f)(a) = g(f(a))$, for all $a \in A$. Let $f : A_1 \to B$ and $g : A_2 \to B$ be two functions. We use $f \oplus g$ to denote the function defined as follows: $(f \oplus g)(a) = f(a)$ if $a \in A_1$ and $(f \oplus g)(a) = g(a)$, if $a \in A_2 \setminus A_1$. If $f(a) = g(a)$ for all $a \in A_1 \cap A_2$, then $f \cup g$ denotes the the union of the functions f and g. Here, $(f \cup g)(a) = f(a)$ if $a \in A_1$ and $(f \cup g)(a) = g(a)$, otherwise. If $A_1 \cap A_2 = \emptyset$, then we may also write $f \uplus g$ instead of $f \cup g$.

Throughout this paper, we use simple, undirected graphs. We say a graph is connected if, for any pair of vertices in it, there is a path between them in the graph. Let G be a graph. We use $V(G)$ and $E(G)$ to denote its vertex and edge sets, respectively. We use $\{u, v\}$ as well as uv to denote the edge between vertex u and vertex v. For a vertex subset $U \subseteq V(G)$, $E_G(U)$ denotes the set of edges incident on U. That is, $E_G(U) = \{\{x, y\} \in E(G) : x \in U \vee y \in U\}$. For a vertex v, we use $E_G(v)$ to denote the set $E(\{v\})$. For a vertex subset $U \subseteq E(G)$, we use $E_G[U]$ to denote the set of edges with both endpoints in G. That is, $E_G[U] = \{\{u, v\} \in E(G) : u, v \in U\}$. For an edge subset $F \subseteq E(G)$, we use $V_G(F)$ to denote the set of endpoints of the edges in F. When the graph is clear from the context, we remove the subscript G in the above notations. For a vertex subset $U \subseteq V(G)$, we use $G[U]$ to denote the subgraph of G induced by U. That is, $V(G[U]) = U$ and $E(G[U]) = E_G[U]$. For an edge subset $F \subseteq E(G)$, we use $G[F]$ to denote the subgraph of G induced by F. That is, $V(G[F]) = V_G(F)$ and $E(G[F]) = F$. A *separation* of G is a pair (A, B) of vertex subsets such that $A \cup B = V(G)$, and there is no edge with one endpoint in $A \setminus B$ and the other in $B \setminus A$. The *separator* of this separation is $A \cap B$, and the *order* of the separation is $|A \cap B|$.

The following lemma is a folklore.

▶ **Lemma 3.** *Let G be a graph, $q \in \mathbb{N}$, and $d\colon E(G) \to \mathbb{N}$ be an edge q-coloring using the maximum number of colors. Then, for any color $r \in \mathbb{N}$ with $d^{-1}(r) \neq \emptyset$, $G[d^{-1}(r)]$ is connected.*

Proof. Suppose by contradiction, if $G[d^{-1}(r)]$ is disconnected, we have two disjoint components. It means we can give one new color (say $r' \neq r$) to any one component and strictly increase the number of colors used by one. This will be again a q-coloring of G. However, this is a contradiction as the given coloring uses the maximum number of colors. ◀

Tree decompositions. A tree decomposition of a graph G is a pair $\mathcal{T} = (\mathrm{T}, \{X_t\}_{t \in V(\mathrm{T})})$, where T is a tree and every node $t \in V(\mathrm{T})$ is assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

1. $\bigcup_{t \in V(\mathrm{T})} X_t = V(G)$.
2. For every $\{u, v\} \in E(G)$, there is a node $t \in V(\mathrm{T})$ such that $u, v \in X_t$.
3. For every $u \in V(G)$, the subgraph of T induced by $\mathrm{T}_u = \{t \in V(\mathrm{T}) : u \in X_t\}$ is connected.

The *width* of tree decomposition $\mathcal{T} = (\mathrm{T}, \{X_t\}_{t \in V(\mathrm{T})})$ is $\max_{t \in V(\mathrm{T})} |X_t| - 1$. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width among all tree decompositions of G. To explain dynamic programming in an easier way, we recall the definition of a nice tree decomposition. A *nice* tree decomposition if a $\mathcal{T} = (\mathrm{T}, \{X_t\}_{t \in V(\mathrm{T})})$ where T is a rooted tree and satisfies the following additional conditions. Let r be the root node in T.

1. $X_r = \emptyset$ and $X_\ell = \emptyset$ for every leaf node ℓ.
2. Each non-leaf node of T has one of the following three types:
   - **Introduce node:** A node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ and $v \notin X_{t'}$; we say that v is *introduced* at t.
   - **Forget node:** a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{w\}$ for some vertex $w \in X_{t'}$; we say that w is *forgotten* at t.
   - **Join node:** a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.

▶ **Lemma 4** (Lemma 7.4 in [5]). *If a graph G admits a tree decomposition of width at most k, then it also admits a nice tree decomposition of width at most k. Moreover, given a tree decomposition $\mathcal{T} = (\mathrm{T}, \{X_t\}_{t \in V(\mathrm{T})})$ of G of width at most k, one can in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G of width at most k that has at most $O(k|V(G)|)$ nodes.*

Parameterized complexity. A parameterized problem P is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the finite alphabet. Fixed parameter traceability of a problem P means whether we can decide the problem in $O(f(k) \cdot p(n))$ time, where k is the fpt parameter, n is the input size, $f(.)$ is some arbitrary function and $p(.)$ is a polynomial function.

▶ **Definition 5** (Parameterized reduction [5]). *Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A* parameterized reduction *from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that*

1. *$(x, k) \in A \iff (x', k') \in B$.*
2. *$k' \leq g(k)$ for some computable function $g(.)$, and*
3. *the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function $f(.)$.*

3 FPT algorithm parameterized by treewidth

We give a dynamic programming algorithm for Maximum Edge q-Coloring when a tree decomposition of width at most k is given as part of the input. Without loss of generality, we assume that a nice tree decomposition is part of the input. Let $(G, \mathcal{T} = (T, \{X_t\}_{t \in V(T)}))$ be the input. Let $n = |V(G)|$, $m = |E(G)|$, and the width of the tree decomposition $\mathcal{T}$ is k. Our algorithm should output an edge q-coloring of G using the maximum number of colors.

Let us define some notations which we use in this section. Recall that T is a rooted tree. Let r be the root of T. For a node $t \in V(T)$, V_t is the union of the bags in the subtree rooted at t. That is, if T_t is the subtree of T rooted at t, then $V_t = \bigcup_{t' \in V(T_t)} X_{t'}$. We use G_t to denote the graph $G[V_t]$. Notice that for any $t \in V(T)$, $(V_t, (V(G) \setminus V_t) \cup X_t)$ is a separation of G of order $|X_t|$, which is upper bounded by k. Now consider the following lemma.

▶ **Lemma 6.** *Let G be a graph, $q \in \mathbb{N}$, and $d \colon E(G) \to \mathbb{N}$ be an edge q-coloring using a maximum number of colors. Let (A, B) be a separation of G and there is a color r such that $r \in d(E_G[A \setminus B])$ and $r \notin d(E_G[A] \setminus E_G[A \setminus B])$. Then, $r \notin E_G[B]$.*

Proof. Let r be the color specified in the lemma and let $\{u, v\} \in d^{-1}(r)$. From the definition of $E_G[A \setminus B]$, we have $u, v \in A \setminus B$. We are given that $r \notin d(E_G[A] \setminus E_G[A \setminus B])$. It means there is no edge with at least one endpoint in $A \cap B$ with color r. The reason is that $E_G[A \setminus B]$ is the set of edges with both the endpoints in $A \setminus B$ and $E_G[A] = \{\{x, y\} : x, y \in A \setminus B\} \cup \{\{x, y\} : x \in A \cap B \vee y \in A \cap B\}$. This implies that there is no edge from $E_G[A]$ with at least one endpoint in $A \cap B$ and colored with r. We need to prove that there is no edge in $E_G[B]$ colored with r. Now, for the sake of contradiction, say we have an edge with color r, and it is in $E_G[B]$. By Lemma 3, there should be a path from the edge $\{u, v\}$ (as defined above) to this edge in $E_G[B]$ and all the edges in this path are colored r. This path must pass through at least one vertex in $A \cap B$ as (A, B) is a separation of G. But it leads to contradiction as we are given that $r \notin d(E_G[A] \setminus E_G[A \setminus B])$. It means $r \notin E_G[B]$. ◀

Lemma 6 helps us to design a dynamic programming algorithm. Because of Lemma 6, at any node $t \in V(T)$, for any coloring of G_t, we do not need to remember the colors of the edges incident on the vertices other than X_t, as it will not be used to color the "future" edges. More formally, in the dynamic programming for any node t we compute and store a set $\mathcal{C}_t$ of edge q-colorings of G_t. Here, we use two disjoint sets of colors; the first set is $[qk] = \{1, 2, \ldots, qk\}$ and the other set is $\{a_1, a_2, \ldots, a_{nq}\}$. We use only the colors from $[qk]$ to color the edges incident on X_t. Other edges in G_t can be colored with any colors from $[qk] \cup \{a_1, a_2, \ldots, a_{nq}\}$. But, we make sure that if an edge in G_t is colored with a color c from $[qk]$, then at least one edge incident on X_t is colored with c. Consider two edges q-coloring g_1 and g_2 of G_t that satisfy the conditions mentioned above. We say that g_1 and g_2 are equivalent, denoted by $g_1 \sim_t g_2$, if the following conditions hold.

(i) $|g_1(E(G_t))| = |g_2(E(G_t))|$. That is, the number of colors used by both g_1 and g_2 are the same.

(ii) For all $u \in X_t$, $g_1(E_{G_t}(u)) = g_2(E_{G_t}(u))$. That is, the colors seen by the edges incident on u are the same in both the colorings g_1 and g_2, for any vertex $u \in X_t$.

Lemma 6 implies that if $g_1 \sim_t g_2$, and g_1 can be extended to a maximum edge q-coloring of G, then g_2 can be extended to a maximum edge q-coloring of G. This is formulated in the following lemma.

▶ **Lemma 7.** *Suppose $g_1 \sim_t g_2$. If there is an edge q-coloring f_1 of G such that $f_1|_{E(G_t)} = g_1$, then there is an edge q-coloring f_2 of G such that $f_2|_{E(G_t)} = g_2$ and $|f_1(E(G))| = |f_2(E(G))|$.*

Proof. We know that both g_1 and g_2 satisfy the following conditions.

(a) All the edges incident on X_t are colored using the colors from $[qk]$ by g_1 and g_2.

(b) All the edges in G_t are colored using colors from $[qk] \cup \{a_1, \dots, a_{qn}\}$. Moreover, for each $i \in [2]$, if a color $c \in [qk]$ is used by g_i, then there is an edge $e \in E_{G_t}(X_t)$ such that $g_i(e) = c$.

Since $g_1 \sim_t g_2$ and condition (b) above implies that the number of colors from $\{a_1, \dots, a_{qn}\}$ used by both g_1 and g_2 are same. Without loss of generality let $a_1, \dots, a_\ell$ be the colors in $\{a_1, \dots, a_{qn}\}$ used by the coloring g_2. As mentioned before, there are exactly ℓ colors from $\{a_1, \dots, a_{qn}\}$ used by g_1. Let $i_1, \dots, i_\ell$ be the distinct indices in $[qn]$ such that the colors from $\{a_1, \dots, a_{qn}\}$ used by g_1 are $a_{i_1}, \dots, a_{i_\ell}$. Now, we obtain a coloring $\widehat{f_1}$ from f_1 as follows. Let $\beta : \{a_1, \dots, a_{qn}\} \cup [qk] \to \{a_1, \dots, a_{qn}\} \cup [qk]$ be an arbitrary bijection such that $\beta(r) = r$ for all $r \in [qk]$ and $\beta(a_{i_j}) = a_j$ for all $j \in [\ell]$. Now the coloring $\widehat{f_1}$ is defined as follows. For each $e \in E(G_t)$, $\widehat{f_1}(e) = \beta(f_1(e))$ and for each $e \in E(G) \setminus E(G_t)$, $\widehat{f_1}(e) = f_1(e)$. Let $\widehat{f_1}|_{E(G_t)} = \widehat{g}_1$. Since β is a bijection as defined above, $\widehat{f_1}$ is an edge q-coloring of G, $\widehat{g}_1 \sim_t g_2$, $|f_1(E(G))| = |\widehat{f_1}(E(G))|$, and $\widehat{g}_1(E(G_t)) = g_2(E(G_t))$.

We define an edge q-coloring f_2 of G as below:

$$f_2(e) = \begin{cases} \widehat{f_1}(e), & \text{if } e \in E(G) \setminus E(G_t) \\ g_2(e), & \text{otherwise, i.e., } e \in E(G_t) \end{cases}$$

Since $g_2(E(G_t)) = \widehat{g}_1(E(G_t))$ and $\widehat{f_1}|_{E(G_t)} = \widehat{g}_1$, we have $|f_2(E(G))| = |\widehat{f_1}(E(G))| = |f_1(E(G))|$. Next we prove that indeed f_2 is an edge q-coloring of G. Towards that we need to prove that for all $u \in V(G)$, $|f_2(E_G(u))| \leq q$. Fix a vertex $u \in V(G)$. First, consider the case when $u \in V_t \setminus X_t$. Then, $f_2(E(u)) = g_2(E(u))$ and g_2 is an edge q-coloring of the induced subgraph G_t. Hence, $|f_2(E(u))| \leq q$. Now, consider the case $u \in X_t$. Since $\widehat{g}_1 \sim_t g_2$, $\widehat{g}_1(E_{G_t}(u)) = g_2(E(G_t(u))$. Thus, by the definition of f_2, we get that $f_2(E_G(u)) = \widehat{f_1}(E_G(u))$. This implies that $|f_2(E(u))| \leq q$. Finally, consider the case when $u \in V(G) \setminus V_t$. In this case, $f_2(E_G(u)) = \widehat{f_1}(E_G(u))$ and hence $|f_2(E(u))| \leq q$. ◀

Because of Lemma 7, it is enough to keep one coloring from an equivalence class of $\sim_t$. Let $\mathcal{S}_t$ be the set of all edge q-colorings of G_t such that

(a) all the edges incident on X_t are colored from the set $[qk]$,

(b) all other edges are colored from the set $[qk] \cup \{a_1, \dots, a_{nq}\}$, and

(c) if an edge in G_t is colored with a color c from $[qk]$, then at least one edge incident on X_t is colored with c.

Two colorings in $\mathcal{S}_t$ are equivalent in $\sim_t$ if the conditions (i) and (ii) defined before, hold.

▶ **Lemma 8.** *The number of equivalence classes in $\sim_t$ is upper bounded by $\binom{qk}{q}^k \cdot q^{k+1} \cdot n$.*

Proof. From the definition of an equivalence class, the number of equivalence classes is determined by the product of the number of possibilities for conditions (i) and (ii) mentioned above in this section. For any n-vertex graph G, the number of colors used by any edge q-coloring is at most nq, because for any vertex, the number of colors used for the incident edges is at most q. Recall that $|X_t| \leq k$. Next we count the number of distict combination for the condition (2) (i.e., for all $u \in X_t$, $g_1(E_{G_t}(u)) = g_2(E_{G_t}(u))$). This number is upper bounded by $(\sum_{j=0}^{q} \binom{qk}{j})^k \leq q^k \binom{qk}{q}^k$, since for any vertex in X_t, we choose at most q out of qk colors for coloring the incident edges. Therefore, the total number of equivalence classes in $\sim_t$ is upper bounded by $\binom{qk}{q}^k \cdot n \cdot q^{k+1}$. ◀

Because of Lemmas 7 and 8, for each node t in T, we compute and store a family $\mathcal{C}_t \subseteq \mathcal{S}_t$ of edge q-colorings of G_t such that $|\mathcal{C}_t| \le 2^{O(qk \log qk)} \cdot n$ and at least one coloring in it can be extended to a maximum edge q-coloring of G. Now, we explain how to compute $\mathcal{C}_t$ in a bottom-up fashion. Let t be a node in T and assume that we have computed $\mathcal{C}_{t'}$ for all node t' such that $t \neq t'$ and t' is a node in the subtree rooted at t.

Leaf Node. In this case $V(G_t) = \emptyset$, and hence $\mathcal{C}_t$ contains only one coloring which is an empty function.

Introduce Node. Suppose t is an introduce node with a child t' such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$. Recall that we have already computed $\mathcal{C}_{t'}$. Notice that v is adjacent to at most $k-1$ vertices in G_t. Moreover, $N_{G_t}(v) \subseteq X_t$ and $E(G_t) = E(G_{t'}) \uplus E_{G_t}(v)$. Now construct a set $\mathcal{D}_t$ as follows. Initially we set $\mathcal{D}_t := \emptyset$. Now for each coloring $f \in \mathcal{C}_{t'}$ and each coloring $g : E_{G_t}(v) \to [qk]$ such that $f \uplus g$ is an edge q-coloring of G_t, we add $f \uplus g$ to $\mathcal{D}_t$. Finally, construct a minimal subfamily $\mathcal{C}_t \subseteq \mathcal{D}_t$ such that for each non-empty equivalence class of $\sim_t$ in $\mathcal{D}_t$, there is exactly one such coloring in $\mathcal{C}_t$. The subfamily $\mathcal{C}_t$ is easy to compute. Initially, we set $\mathcal{C}_t := \mathcal{D}_t$, and while there are two colorings in $\mathcal{C}_t$ which are equivalent, delete one of them and repeat this step.

Forget Node. Suppose t is a forget node with a child t' such that $X_t = X_{t'} \setminus \{w\}$ for some $w \in X_{t'}$. Notice that here $G_t = G_{t'}$ and we have already computed $\mathcal{C}_{t'}$. But $X_t = X_{t'} \setminus \{w\}$. We want each of the colorings in $\mathcal{C}_t$ to have the following property. If a color is used only to color the edges from $E(G_t) \setminus E_{G_t}(X_t)$, then that color should be from $\{a_1, \ldots, a_{qn}\}$. For each coloring $f \in \mathcal{C}_{t'}$, we construct a coloring f' as follows. If a color $c \in [qk]$ is used to color an edge in $E(G_t) \setminus E_{G_t}(X_t)$ and not used to color any edge in $E_{G_t}(X_t)$, then recolor those edges with an unused color from $\{a_1, \ldots, a_{qn}\}$. The set $\mathcal{C}_t$ is the collection of such colorings f'. Clearly $|\mathcal{C}_t| = |\mathcal{C}_{t'}|$.

Join Node. Let t_1 and t_2 be the children of t. In this case, we have $X_t = X_{t_1} = X_{t_2}$ and G_t is the union of G_{t_1} and G_{t_2}. That is, $V(G_t) = V(G_{t_1}) \cup V(G_{t_2})$ and $E(G_t) = E(G_{t_1}) \cup E(G_{t_2})$. Also, we have already computed $\mathcal{C}_{t_1}$ and $\mathcal{C}_{t_2}$. We want to construct $\mathcal{C}_t$ by combining functions from $\mathcal{C}_{t_1}$ and $\mathcal{C}_{t_2}$. Initially we set $\mathcal{D}_t := \emptyset$. For a function $f_1 \in \mathcal{C}_{t_1}$ and a function $f_2 \in \mathcal{C}_{t_2}$, we construct an edge coloring of G_t, which is described below. Let $A_j = f_j(E_{G_{t_j}}) \cap \{a_1, \ldots, a_{nq}\}$ and $\ell_j = |A_j|$, for all $j \in \{1,2\}$. Let $\beta_1 : A_1 \to \{a_1, \ldots, a_{\ell_1}\}$ and $\beta_2 : A_2 \to \{a_{\ell_1+1}, \ldots, a_{\ell_1+\ell_2}\}$ be two arbitrary fixed bijections. Notice that since f_1 and f_2 are edge q-colorings from $\mathcal{C}_{t_1} \subseteq \mathcal{S}_{t_1}$ and $\mathcal{C}_{t_2} \subseteq \mathcal{S}_{t_2}$, respectively, and $(V_{t_1} \cap V_{t_2}) \setminus X_t = \emptyset$, we have that $\ell_1 + \ell_2 \le nq$, and hence β_1 and β_2 are well defined. Now, for all $j \in \{1,2\}$, $f'_j : V_{t_j} \to [qk] \cup \{a_1, \ldots, a_{nq}\}$ be the edge q-colorings defined as follows: $f'_j(e) = f_j(e)$, if $f_j(e) \in [qk]$, and $f'_j(e) = \beta_j(f_j(e))$ if $f_j(e) \in \{a_1, \ldots, a_{nq}\}$. It is easy to see that $f_j \sim_{t_j} f'_j$ and f'_j is indeed an edge q-coloring of G_{t_j}. Moreover $f'_1(E(G_{t_1})) \cap f'_2(E(G_{t_2})) \subseteq [qk]$. Now define an edge coloring $f'_1 \oplus f'_2$ of G_t as follows.

$$f'_1 \oplus f'_2(e) = \begin{cases} f'_1(e), & \text{if } e \in E(G_{t_1}) \\ f'_2(e), & \text{otherwise, i.e., } e \in E(G_{t_2}) \setminus E(G_{t_1}) \end{cases}$$

Notice that each edge in G_t gets exactly one color in $f'_1 \oplus f'_2$, and hence $f'_1 \oplus f'_2$ is edge coloring of G_t. If $f'_1 \oplus f'_2 \in \mathcal{S}_t$, then we include $f'_1 \oplus f'_2$ in $\mathcal{D}_t$. Notice that when $f'_1 \oplus f'_2 \in \mathcal{S}_t$, $f'_1 \oplus f'_2$ is an edge q-coloring of G_t. Finally, construct a minimal subfamily $\mathcal{C}_t \subseteq \mathcal{D}_t$ such that for each non-empty equivalence class of $\sim_t$ in $\mathcal{D}_t$, there is exactly one such coloring in $\mathcal{C}_t$.

This completes the construction of $\mathcal{C}_t$. Recall that r is the root of T. Finally, we output a coloring from $\mathcal{C}_r$ that uses maximum number of colors.

Correctness. For the correctness proof, it is enough to prove the following statement. For any maximum edge q-coloring h of G, any node $t \in V(T)$, and any injective function $\beta : h(E(G)) \to [qk] \cup \{a_1, \ldots, a_{nq}\}$ such that $\beta \circ h(E_{G_t}(X_t)) \subseteq [qk]$, there is an edge q-coloring $f \in \mathcal{C}_t$ such that $f \sim_t (\beta \circ h)|_{E(G_t)}$. We prove the statement using mathematical induction, where the base case is when t is a leaf node. The base case is trivially true, because $V(G_t) = \emptyset$ for a leaf node t.

Consider the case when t is labelled as an introduce node. Let t' be the child of t and $X_{t'} = X_t \setminus \{v\}$. Let h be a maximum edge q-coloring of G and $\beta : h(E(G)) \to [qk] \cup \{a_1, \ldots, a_{nq}\}$ be an injective function such that $(\beta \circ h)(E_{G_t}(X_t)) \subseteq [qk]$. Let $f = (\beta \circ h)|_{E(G_{t'})}$ and $g = (\beta \circ h)|_{E_{G_t}(v)}$. By induction hypothesis, there is $f' \in \mathcal{C}_{t'}$ with $f' \sim_{t'} f = (\beta \circ h)|_{E(G_{t'})}$. Then, $f' \uplus g \in \mathcal{D}_t$ and there is a function $f'' \in \mathcal{C}_t$ such that $f'' \sim_t f' \uplus g$. Since, $f' \sim_{t'} f$, we get that $f'' \sim_t (\beta \circ h)|_{E(G_t)}$.

Consider the case when t is labelled as forget node and t' be the child of t. Notice that in this case for any coloring f in $\mathcal{C}_{t'}$, we changed some colors that are not used to color the edges in $E_{G_t}(X_t)$ to some other unused colors. Hence, the proof is simple and we omit here.

Consider the case when t is labelled as a join node. Let t_1 and t_2 be the children of t. Here, we have $X_t = X_{t_1} = X_{t_2}$. Let h be a maximum edge q-coloring of G and $\beta : h(E(G)) \to [qk] \cup \{a_1, \ldots, a_{nq}\}$ be an injective function such that $(\beta \circ h)(E_{G_t}(X_t)) \subseteq [qk]$. Let $g_1 = (\beta \circ h)|_{E(G_{t_1})}$ and $g_2 = (\beta \circ h)|_{E(G_{t_2})}$. By induction hypothesis, there are function $f_1 \in \mathcal{C}_{t_1}$ and $f_2 \in \mathcal{C}_{t_2}$ such that $f_1 \sim_{t_1} g_1$ and $f_2 \sim_{t_2} g_2$. Recall the functions f_1' and f_2' constructed in the algorithm. Since $g_1 \oplus g_2 = g_1 \cup g_2$ is an edge q-coloring of G_t, $f_1 \sim_{t_1} g_1$ and $f_2 \sim_{t_2} g_2$, we get that $f_1' \oplus f_2'$ is an edge q-coloring of G_t. Moreover, by the construction of f_1' and f_2', the number of colors used by $f_1' \oplus f_2'$ is same as the number of colors used by $g_1 \cup g_2$. Also, since $f_1 \sim_{t_1} g_1$ and $f_2 \sim_{t_2} g_2$, we get that $(f_1' \oplus f_2') \sim_t (g_1 \cup g_2) = (\beta \circ h)|_{E(G_t)}$. This completes the correctness proof.

Runtime analysis. Lemma 8 implies that for all $t \in V(T)$, $|\mathcal{C}_t| \leq \binom{qk}{q}^k \cdot q^{k+1} \cdot n$. There are $O(kn)$ nodes in T and the bottleneck in the computation is the computation of $\mathcal{C}_t$ for a join node. This running time is upper bounded by $O(|\mathcal{C}_{t_1}| \cdot |\mathcal{C}_{t_2}| \cdot n)$, where t_1 and t_2 are the children of t. This is upper bounded by $2^{O(kq \log kq)} n^3$. Thus the total running time is upper bounded by $2^{O(kq \log kq)} n^4$.

Improving the running time. Recall the definition of $\sim_t$. Two functions g_1 and g_2 are equivalent under $\sim_t$, if the following conditions hold.

(i) $|g_1(E(G_t))| = |g_2(E(G_t))|$. That is, the number of colors used by both g_1 and g_2 are the same.

(ii) For all $u \in X_t$, $g_1(E_{G_t}(u)) = g_2(E_{G_t}(u))$. That is, the colors seen by the edges incident on u are the same in both the colorings g_1 and g_2.

Instead of this, we may define the equivalence only when condition (ii) is satisfied. Then, in the computation we may store one from the equivalence class that uses maximum number of colors. This will reduce the number of equivalence classes to $\binom{qk}{q}^k \cdot q^k$ and thus the total running time to $2^{O(kq \log kq)} n$.

4 Hardness result

A *matching* M in a graph G is a collection of pairwise vertex disjoint edges, and the *size* of M is the number of edges in M. A *maximum matching* is a matching of the largest size. We shall use $\gamma(G)$ to denote the size of a maximum matching in G. It is known that every graph G has an edge 2-coloring of size at least $\gamma(G)$ as assigning every edge in a maximum matching a distinct color and the rest of the edges another new color is indeed a valid edge 2-coloring. We thus consider the following above-guarantee version of the maximum edge 2-coloring problem.

Above-Guarantee Edge 2-coloring **Parameter:** k
Input: An undirected graph G and $k \in \mathbb{N}$.
Question: Does G have an edge 2-coloring of size $\gamma(G) + k$?

An *independent set* I in a graph G is a subset of its vertices such that no two vertices in I are adjacent to each other in G. The *size* of I is its cardinality. A *maximum independent set* in a graph G is an independent set of largest size. We shall use $\alpha(G)$ to denote the size of a largest independent set in G. It is known [5] that the following problem on whether a graph G has an independent set of size ℓ parameterized by ℓ is W[1]-hard.

Independent Set **Parameter:** ℓ
Input: An undirected graph G and $\ell \in \mathbb{N}$
Question: Does G have an independent set of size ℓ?

In this section, we prove that the Above-Guarantee Edge 2-coloring problem is W[1]-hard by giving an parameterized reduction from the Independent Set problem to the former.

4.1 Construction

Let (H, ℓ) be an instance of the Independent Set problem. Let $V(H) = \{1, \ldots, n\}$. From H we construct a bipartite graph G as described below:

1. For each vertex $i \in V(H)$, we have an edge $u_i u_i'$ in G. Let $U = \{u_i : i \in V(H)\} \cup \{u_i' : i \in V(H)\}$.
2. For each edge $ij \in E(H)$, we have a vertex $x_{i,j}$ that is adjacent to u_i and u_j. Let $X = \{x_{i,j} : ij \in E(H)\}$.
3. Finally, we have an edge ww' in G with w adjacent to every vertex in X. Let $W = \{w, w'\}$.

See Figure 1 for an example of the construction of the graph G from $H = K_4$. Here, $V(H) = [4]$ and $E(H) = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\}\})$.

Thus, $V(G) = U \cup X \cup W$, and the set of edges of G is as defined above. In the rest of this section, we shall use U-X *edges* to denote the set of edges having one endpoint in U and the other in X. In a similar way, we define X-W edges. Finally, for any vertex v in G, we shall use v-U (resp., v-X, v-W) edges to denote the set of edges from v to U (resp., X, W).

4.2 The proof

Throughout this section, we assume that (i) H is a graph on n vertices, and (ii) G is the graph constructed from H as described in Section 4.1.

▶ **Proposition 9.** $\gamma(G) = n + 1$.

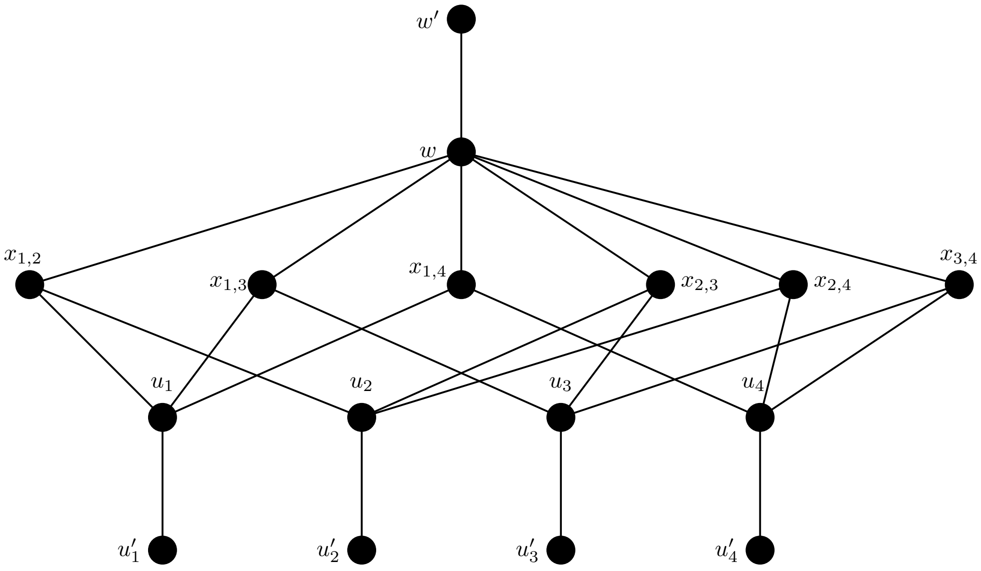


Figure 1 Graph G constructed from $H = K_4$.

Proof. The set $M = \{u_i u'_i \ : \ i \in [n]\} \cup \{ww'\}$ is a matching of size $n+1$. Thus, $\gamma(G) \geq n+1$. To show that $\gamma(G) \leq n+1$, consider any matching A of G. We construct a matching A' out of A with $|A'| = |A|$ below. If A contains any w-X edge, replace it with ww' in A'. If A contains any $x_{i,j}u_i$ edge, replace it with $u_i u'_i$ in A'. Thus, every edge in A' is a pendant edge of G. Since the number of pendant edges in G equals $n+1$, we have $|A| = |A'| \leq n+1$. ◀

▶ **Lemma 10.** *Given any edge 2-coloring f of G, one can obtain another edge 2-coloring f' of G such that*

1. *Every w-X edge in G has color c_F under f',*
2. *For every u_i, all u_i-X edges are of the same color under f'.*
3. *f' uses the same number of colors as f*

Proof. Below, we construct the coloring f' from f. Without loss of generality, assume that c_F is a color seen by the vertex w under f. If w sees only one color under f, then do nothing. Suppose w sees two colors, say c and c_F. In that case, replace every occurrence of c in the given coloring of G with c_F and finally assign the color c to the edge ww'. The resultant coloring is a valid edge 2-coloring as every vertex continues to see the same number of colors. Further, we have managed to satisfy Condition 1. The size of the new coloring obtained is the same as that of f. Now, consider each vertex u_i, $1 \leq i \leq n$, individually. If u_i sees only one color under f, then do nothing.

Suppose u_i sees two colors. In that case, (i) if c_F and, without loss of generality, c are the two colors seen by u_i, then replace every occurrence of c in the given coloring of G with c_F and finally assign the color c to the edge $u_i u'_i$, or (ii) if, without loss of generality, c and c' are the two colors seen by u_i, then replace every occurrence of say c' in the given coloring with c and finally assign the color c' to the edge $u_i u'_i$. Let us call the resultant coloring f'. Note that f' satisfies Condition 2. Lastly, note that the size of f' is the same as that of f. ◀

▶ **Lemma 11.** *Any maximum edge 2-coloring of G is of size $n + \alpha(H) + 2$.*

Proof. Assign color c_i to every $u_i u'_i$ edge, color c_0 to the ww' edge, and color c_F to every w-X edge. Let $S \subseteq V(H)$ be a maximum independent set in H. For each $i \in S$, assign the color c'_i to every u_i-X edge. For all the remaining X-U edges, assign the color c_F. Note that in the above coloring, (i) the vertex w sees the colors c_0 and c_F, (ii) every $u_i \in S$ sees colors c_i and c'_i, and (iii) every $u_i \in U \setminus S$ sees colors c_F and c_i. Now consider a vertex $x_{i,j} \in X$. Since S is an independent set in H, S won't contain both i and j. If S contains neither, then $x_{i,j}$ sees only color c_F. Without loss of generality, assume $i \in S$. Then, $x_{i,j}$ sees the colors c'_i and c_F. Thus, the above coloring is a valid edge 2-coloring. Note that the size of the above coloring is $n + \alpha(H) + 2$. We have thus shown that the size of a maximum edge 2-coloring of G is at least $n + \alpha(H) + 2$.

We now show that any edge 2-coloring of G is of size at most $n + \alpha(H) + 2$. Let f be an edge 2-coloring of G. Apply Lemma 10 to obtain the coloring f' from f. Let $P_{f'}$ be the set of colors seen by the pendant vertices of G under f'. Since G has $n + 1$ pendant vertices, $|P_{f'}| \leq n + 1$. We know that, under f', every w-X edge is assigned the color c_F, and for every u_i, all u_i-X edges are of the same color. Let $R_{f'}$ denote the set of all colors used by the coloring f' that are not present in $P_{f'} \cup \{c_F\}$. Observe that every color in $R_{f'}$ is used to color U-X edges (as the color of every other edge is present in $P_{f'} \cup \{c_F\}$). Let $U_R \subseteq U$ be a set of size $r := |R_{f'}|$ such that (i) for any two distinct $u_i, u_j \in U_R$, the color of u_i-X edges is different from the color of u_j-X edges, and (ii) the set of colors used to color the U_R-X edges is equal to $R_{f'}$. Without loss of generality, assume $U_R = \{u_1, \dots, u_r\}$. We claim that the set $\{1, \dots, r\}$ is an independent set in H. Suppose not. Then $ij \in E(H)$, for some $1 \leq i < j \leq r$. However, this would mean that the vertex $x_{i,j}$ sees three distinct colors (on its edges $x_{i,j}u_i, x_{i,j}u_j$, and $x_{i,j}w$) which is a contradiction to the fact that f' is a valid edge 2-coloring. Hence, $\{1, \dots, r\}$ is an independent set in H and therefore, $r \leq \alpha(H)$. The set of colors used by f' is $P_{f'} \cup \{c_F\} \cup R_{f'}$ which is at most $(n+1)+(1)+(\alpha(H)) = n+\alpha(H)+2$. ◀

▶ **Theorem 12.** *Above-Guarantee Edge 2-coloring is W[1]-hard.*

Proof. Let (H, ℓ) be an instance of Independent Set. We construct G from H as described in Section 4.1. Note that this construction can be done in $O(n^2)$ time. Let $k = \ell + 1$. We claim that $(H, \ell) \in$ Independent Set if and only $(G, k) \in$ Above-Guarantee Edge 2-coloring. If H has an independent set of size ℓ, then, by Lemma 11, G has an edge 2-coloring of size $n + \alpha(H) + 2 \geq \gamma(G) + k$ (because $\alpha(H) \geq \ell$ and from Proposition 9, we have $\gamma(G) = n + 1$). From such a coloring, it is easy to obtain a valid edge 2-coloring of size exactly $\gamma(G) + k$ (see Proposition 1 in [10] for a proof of this statement). To prove the converse, suppose H does not have any independent set of size ℓ. Then, by Lemma 11, G has no edge 2-coloring of size $n + \ell + 2 = (n + 1) + (\ell + 1) = \gamma(G) + k$. This completes the reduction. Note that this is a parameterized reduction from Independent Set to Above-Guarantee Edge 2-coloring as it satisfies all the three conditions of Definition 5. Since Independent Set is known to be W[1]-hard, we have the theorem. ◀

5 Concluding remarks

In this work, we resolve an open question of Goyal et al. [10]. Further, we give an FPT algorithm of running time $2^{O(\mathsf{tw} \cdot q \log(\mathsf{tw} \cdot q))} n$ for Maximum Edge q-Coloring, where tw is the treewidth of the input graph. It is natural to ask if this running time is optimal. We would like to mention that as a corollary of a result of Goyal et al. [10] (as well as our result above), one gets an FPT algorithm of running time $2^{O(\mathsf{vc} \log(\mathsf{vc}))} n$ for Maximum Edge 2-Coloring, where vc is the vertex cover number of the input graph. It would be interesting to obtain a single exponential FPT algorithm and a polynomial kernel even when the parameter is vertex cover number.

References

1 Anna Adamaszek and Alexandru Popa. Approximation and hardness results for the maximum edge q-coloring problem. *Journal of Discrete Algorithms*, 38:1–8, 2016. doi:10.1016/J.JDA.2016.09.003.

2 L. Sunil Chandran, Talha Hashim, Dalu Jacob, Rogers Mathew, Deepak Rajendraprasad, and Nitin Singh. New bounds on the anti-ramsey numbers of star graphs via maximum edge q-coloring. *Discret. Math.*, 347(4):113894, 2024. doi:10.1016/J.DISC.2024.113894.

3 L. Sunil Chandran, Abhiruk Lahiri, and Nitin Singh. Improved approximation for maximum edge colouring problem. *Discret. Appl. Math.*, 319:42–52, 2022. doi:10.1016/J.DAM.2021.05.017.

4 B Courcelle. The monadic second-order theory of graphs i: Recognizable sets of finite graphs. *Information and Computation*, 1990.

5 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5(4). Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Zdeněk Dvořák and Abhiruk Lahiri. Maximum edge colouring problem on graphs that exclude a fixed minor. In Daniël Paulusma and Bernard Ries, editors, *Graph-Theoretic Concepts in Computer Science*, pages 291–304, Cham, 2023. Springer Nature Switzerland.

7 Pál Erdős, Miklós Simonovits, and Vera T Sós. Anti-ramsey theorems. In *Infinite and finite sets: To Paul Erdős on his 60th birthday.* North-Holland Publishing Company, 1975.

8 Wangsen Feng, Hanpin Wang, et al. Approximation algorithm for maximum edge coloring. *Theoretical computer science*, 410(11):1022–1029, 2009. doi:10.1016/J.TCS.2008.10.035.

9 Shinya Fujita, Colton Magnant, and Kenta Ozeki. Rainbow generalizations of ramsey theory: a survey. *Graphs and Combinatorics*, 26:1–30, 2010. doi:10.1007/S00373-010-0891-3.

10 Prachi Goyal, Vikram Kamat, and Neeldhara Misra. On the parameterized complexity of the maximum edge 2-coloring problem. In *Mathematical Foundations of Computer Science 2013: 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings 38*, pages 492–503. Springer, 2013. doi:10.1007/978-3-642-40313-2_44.

11 Ashish Raniwala and Tzi-cker Chiueh. Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 2223–2234. IEEE, 2005. doi:10.1109/INFCOM.2005.1498497.

12 Ashish Raniwala, Kartik Gopalan, and Tzi-cker Chiueh. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(2):50–65, 2004. doi:10.1145/997122.997130.

Additive Word Complexity and `Walnut`

Pierre Popoli
Department of Mathematics, University of Liège, Belgium

Jeffrey Shallit
School of Computer Science, University of Waterloo, Canada

Manon Stipulanti
Department of Mathematics, University of Liège, Belgium

Abstract

In combinatorics on words, a classical topic of study is the number of specific patterns appearing in infinite sequences. For instance, many works have been dedicated to studying the so-called factor complexity of infinite sequences, which gives the number of different factors (contiguous subblocks of their symbols), as well as abelian complexity, which counts factors up to a permutation of letters. In this paper, we consider the relatively unexplored concept of additive complexity, which counts the number of factors up to additive equivalence. We say that two words are additively equivalent if they have the same length and the total weight of their letters is equal. Our contribution is to expand the general knowledge of additive complexity from a theoretical point of view and consider various famous examples. We show a particular case of an analog of the long-standing conjecture on the regularity of the abelian complexity of an automatic sequence. In particular, we use the formalism of logic, and the software `Walnut`, to decide related properties of automatic sequences. We compare the behaviors of additive and abelian complexities, and we also consider the notion of abelian and additive powers. Along the way, we present some open questions and conjectures for future work.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words

Keywords and phrases Combinatorics on words, Abelian complexity, Additive complexity, Automatic sequences, Walnut software

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.32

Funding *Pierre Popoli*: supported by ULiège's Special Funds for Research, IPD-STEMA Program.
Jeffrey Shallit: supported by NSERC grants 2018-04118 and 2024-03725.
Manon Stipulanti: FNRS Research Associate supported by the Research grant 1.C.104.24F.

Acknowledgements We thank the reviewers for providing valuable remarks leading to a better version of the paper. We also thank Matthieu Rosenfeld and Markus Whiteland for helpful discussions, and Eric Rowland for implementing useful Mathematica code.

1 Introduction

Combinatorics on words is the study of finite and infinite sequences, also known as *streams* or *strings* in other theoretical contexts. Although it is rooted in the work of Axel Thue, who was the first to study regularities in infinite words in the early 1900's, words became a systematic topic of combinatorial study in the second half of the 20th century [8]. Since then, many different approaches have been developed to analyze words from various points of view. One of them is the celebrated *factor* or *subword complexity function*: given an infinite word $\mathbf{x}$ and a length $n \geq 0$, we compute the size of $\mathcal{L}_n(\mathbf{x})$, which contains all length-n contiguous subblocks of $\mathbf{x}$, also called *factors* or *subwords* in the literature. One of the most famous theorems in combinatorics on words related to the factor complexity function is due to Morse and Hedlund in 1940 [27], where they obtained a characterization of ultimately periodic words. As a consequence of this result, combinatorists defined binary aperiodic infinite words having the smallest possible factor complexity function, the so-called *Sturmian words*.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 32; pp. 32:1–32:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Many other complexity functions have been defined on words depending on the properties combinatorists wanted to emphasize; see, for instance, the non-exhaustive list in the introduction of [1]. As the literature on the topic is quite large, we only cite the so-called *abelian complexity function.* Instead of counting all distinct factors, we count them up to abelian equivalence: two words u and v are *abelian equivalent*, written $u \sim_{\mathrm{ab}} v$, if they are permutations of each other. For instance, in English, `own`, `now`, and `won` are all abelian equivalent. For an infinite word $\mathbf{x}$, we let $\rho_{\mathbf{x}}^{\mathrm{ab}}$ denote its abelian complexity function. In this paper, we study yet another equivalence relation on words; namely, *additive equivalence.* Roughly, two words are additively equivalent if the total weight of their letters is equal. In the following, for a word $w \in \Sigma^*$, we let $|w|$ denote its *length*, i.e., the number of letters it is composed of. Furthermore, for each letter $a \in \Sigma$, we let $|w|_a$ denote the number of a's in w.

▶ **Definition 1.** *Fix an integer $\ell \geq 1$ and the alphabet $\Sigma = \{0, 1, \ldots, \ell\}$. Two words $u, v \in \Sigma^*$ are* additively equivalent *if $|u| = |v|$ and $\sum_{i=0}^{\ell} i|u|_i = \sum_{i=0}^{\ell} i|v|_i$, which we write as $u \sim_{\mathrm{add}} v$.*

▶ **Example 2.** Over the three-letter alphabet $\{0, 1, 2\}$, we have $020 \sim_{\mathrm{add}} 101$.

▶ **Definition 3.** *Fix an integer $\ell \geq 1$ and the alphabet $\Sigma = \{0, 1, \ldots, \ell\}$. Let $\mathbf{x}$ be an infinite word on Σ. The* additive complexity *of $\mathbf{x}$ is the function $\rho_{\mathbf{x}}^{\mathrm{add}} \colon \mathbb{N} \to \mathbb{N}, n \mapsto \#(\mathcal{L}_n(\mathbf{x})/{\sim_{\mathrm{add}}})$, i.e., length-$n$ factors of $\mathbf{x}$ are counted up to additive equivalence.*

Surprisingly, not so many results are known for additive equivalence and the corresponding complexity function, in contrast with the abundance of abelian results in combinatorics; see [32, 33, 39, 43, 44], for example. Notice that over a two-letter alphabet, the concepts of abelian and additive complexity coincide. Additive complexity was first introduced in [5], where the main result states that bounded additive complexity implies that the underlying infinite word contains an additive k-power for every k (an *additive k-power* is a word w that can be written as $x_1 x_2 \cdots x_k$ where the words $x_1, x_2, \ldots, x_k$ are all additively equivalent).

Later, words with bounded additive complexity were studied: first in [5], and with more attention in [6]. In particular, equivalent properties of bounded additive complexity were found. We also mention work on the particular case of constant additive and abelian complexities [6, 20, 33, 37]. As already observed, combinatorists expanded the notion of pattern avoidance to additive powers. See the most recent preprint [4] for a nice exposition of the history. For instance, Cassaigne et al. [14] proved that the fixed point of the morphism $0 \mapsto 03, 1 \mapsto 43, 3 \mapsto 1, 4 \mapsto 01$ avoids additive cubes (see Section 2 for concepts not defined in this introduction). Rao [31] proved that it is possible to avoid additive cubes over a ternary alphabet and mentioned that the question about additive squares (in one dimension and over the integers) is still open. Furthermore, we mention the following related papers. Brown and Freedman [11] also talked about the open problem about additive squares. A notion of "close" additive squares is defined in [12]. In [24], the authors showed that in every infinite word over a finite set of non-negative integers there is always a sequence of factors (not necessarily of the same length) having the same sum. In [30], the more general setting of k-power modulo a morphism was studied. Finally, in terms of computing the additive complexity of specific infinite words, to our knowledge, only that of a fixed point of a Thue–Morse-like morphism is known [17].

In this paper, we expand our general knowledge of additive complexity functions of infinite words. After giving some preliminaries in Section 2, we obtain several general results in Section 3, and in Theorem 10 we prove a particular case of the conjecture below. Note that it is itself a particular case of the long-standing similar conjecture in an abelian context: the abelian complexity of a k-automatic sequence is a k-regular sequence [29].

▶ **Conjecture 4.** *The additive complexity of a k-automatic sequence is a k-regular sequence.*

In particular, the proof of our Theorem 10 relies on the logical approach to combinatorics on words: indeed, many properties of words can be phrased in first-order logic. Based on this, Mousavi [28] designed the free software `Walnut` that allows one to automatically decide the truth of assertions about many properties for a large family of words. See [40] for the formalism of the software and a survey of the combinatorial properties that can be decided. In Section 3, we show how `Walnut` may be used in an ad-hoc way to give partial answers to Conjecture 4. In Section 4, we compare the behaviors of the additive and abelian complexity functions of various words. We highlight the fact that they may behave quite differently, sometimes making use of `Walnut`. Motivated by the various behaviors we observe, we study in Section 5 some words for which the additive and abelian complexity functions are in fact equal. We end the paper by considering the related notions of abelian and additive powers.

2 Preliminaries

For a general reference on words, we guide the reader to [26]. An *alphabet* is a finite set of elements called *letters*. A *word* over an alphabet Σ is a finite or infinite sequence of letters from Σ. The *length* of a finite word w, denoted $|w|$, is the number of letters it is made of. The *empty word* is the only 0-length word, denoted by ε. For all $n \geq 0$, we let Σ^n denote the set of all length-n words over Σ. We let Σ^* denote the set of finite words over A, including the empty word, and equipped with the concatenation. In this paper, we distinguish finite and infinite words by writing the latter in bold. For each letter $a \in \Sigma$ and a word $w \in \Sigma^*$, we let $|w|_a$ denote the number of a's in w. Let us assume that the alphabet $\Sigma = \{a_1 < \cdots < a_k\}$ is ordered. For a word $w \in \Sigma^*$, we let $\Psi(w)$ denote the *abelianization* or *Parikh vector* $(|w|_{a_1}, \ldots, |w|_{a_k})$, which counts the number of different letters appearing in w. For example, over the alphabet $\{\mathtt{e} < \mathtt{l} < \mathtt{s} < \mathtt{v}\}$, we have $\Psi(\mathtt{sleeveless}) = (4, 2, 3, 1)$.

A *factor* of a word is one of its (contiguous) subblocks. For a given word $\mathbf{x}$, for all $n \geq 0$, we let $\mathcal{L}_n(\mathbf{x})$ denote the set of length-n factors of $\mathbf{x}$. A *prefix* (resp., *suffix*) is a starting (resp., ending) factor. A prefix or a suffix is *proper* if it is not equal to the initial word. Infinite words are indexed starting at 0. For such a word $\mathbf{x}$, we let $\mathbf{x}(n)$ denote its nth letter with $n \geq 0$ and, for $0 \leq m \leq n$, we let $\mathbf{x}[m..n]$ denote the factor $\mathbf{x}(m) \cdots \mathbf{x}(n)$.

Let Σ and Γ be finite alphabets. A *morphism* $f\colon \Sigma^* \to \Gamma^*$ is a map satisfying $f(uv) = f(u)f(v)$ for all $u, v \in \Sigma^*$. In particular, $f(\varepsilon) = \varepsilon$, and f is entirely determined by the images of the letters in Σ. For an integer $k \geq 1$, a morphism is *k-uniform* if it maps each letter to a length-k word. A 1-uniform morphism is called a *coding*. A sequence $\mathbf{x}$ is *morphic* if there exist a morphism $f\colon \Sigma^* \to \Sigma^*$, a coding $g\colon \Sigma^* \to \Gamma^*$, and a letter $a \in \Sigma$ such that $\mathbf{x} = g(f^\omega(a))$, where $f^\omega(a) = \lim_{n\to\infty} f^n(a)$. The latter word $f^\omega(a)$ is a *fixed point* of f.

Introduced by Cobham [18] in the early 1970s, automatic words have several equivalent definitions depending on the point of view one wants to adopt. For the case of integer base numeration systems, a comprehensive presentation of automatic sequences is [3], while [34, 38] treat the case of more exotic numeration systems. We start with the definition of positional numeration systems. Let $U = (U(n))_{n\geq 0}$ be an increasing sequence of integers with $U(0) = 1$. A positive integer n can be decomposed, not necessarily uniquely, as $n = \sum_{i=0}^{t} c(i)\, U(i)$ with non-negative integer coefficients $c(i)$. If these coefficients are computed greedily, then for all $j < t$ we have $\sum_{i=0}^{j} c(i)\, U(i) < U(j+1)$ and $\mathrm{rep}_U(n) = c(t) \cdots c(0)$ is said to be the *(greedy) U-representation* of n. By convention, that of 0 is the empty word ε, and the greedy representation of $n > 0$ starts with a non-zero digit. A sequence U satisfying all the above conditions defines a *positional numeration system*. Let $U = (U(n))_{n\geq 0}$ be such a numeration system. A sequence $\mathbf{x}$ is *U-automatic* if there exists a deterministic finite automaton with output (DFAO) $\mathcal{A}$ such that, for all $n \geq 0$, the nth term $\mathbf{x}(n)$ of $\mathbf{x}$ is given by the output $\mathcal{A}(\mathrm{rep}_U(n))$ of $\mathcal{A}$. In the particular case where U is built on powers of an integer $k \geq 2$, then $\mathbf{x}$ is said to be *k-automatic*. It is known that a sequence is k-automatic if and only if it is the image, under a coding, of a fixed point of a k-uniform morphism [3].

A generalization of automatic sequences to infinite alphabets is the notion of regular sequences [3, 34, 38]. Given a positional numeration system $U = (U(n))_{n\geq 0}$, a sequence $\mathbf{x}$ is *U-regular* if there exist a column vector λ, a row vector γ and *matrix-valued* morphism μ, i.e., the image of each letter is a matrix, such that $\mathbf{x}(n) = \lambda\mu(\mathrm{rep}_U(n))\gamma$. Such a system of matrices forms a *linear representation* of $\mathbf{x}$. In the particular case where U is built on powers of an integer $k \geq 2$, then $\mathbf{x}$ is said to be *k-regular*. Another definition of k-regular sequences is the following one [3]. Consider a sequence $\mathbf{x}$ and an integer $k \geq 2$. The *k-kernel* of $\mathbf{x}$ is the set of subsequences of the form $(\mathbf{x}(k^e n + r))_{n\geq 0}$ where $e \geq 0$ and $r \in \{0, 1, \ldots, k^e - 1\}$. Equivalently, a sequence is k-regular if the $\mathbb{Z}$-module generated by its k-kernel is finitely generated. A sequence is then k-automatic if and only if its k-kernel is finite [3].

Introduced in 2001 by Carpi and Maggi [13], synchronized sequences form a family between automatic and regular sequences. Given a positional numeration system $U = (U(n))_{n\geq 0}$, a sequence $\mathbf{x}$ is *U-synchronized* if there exists a deterministic finite automaton (DFA) that recognizes the language of U-representations of n and $\mathbf{x}(n)$ in parallel.

3 General results

In this section, we gather general results on the additive complexity of infinite words. Since abelian equivalence implies additive equivalence, we have the following lemma.

▶ **Lemma 5.** *For all infinite words $\mathbf{x}$, we have $\rho^{\mathrm{add}}_{\mathbf{x}}(n) \leq \rho^{\mathrm{ab}}_{\mathbf{x}}(n)$ for all $n \geq 0$.*

As in the case of abelian complexity, we have the following lower and upper bounds for additive complexity. See [19, Rk. 4.07] and [33, Thm 2.4].

▶ **Lemma 6.** *Let $k \geq 1$ be an integer and let $\mathbf{x}$ be an infinite word on $\{a_1 < \cdots < a_k\}$. We have $1 \leq \rho^{\mathrm{add}}_{\mathbf{x}}(n) \leq \binom{n+k-1}{k-1}$ for all $n \geq 0$.*

Note that the lower bound of the previous result is reached for (purely) periodic sequences. The story about the upper bound is a little more puzzling. In fact, for a window length $N \geq 1$, we can find an alphabet and a sequence over this alphabet for which its additive complexity reaches the stated upper bound on its first N values. Indeed, fix an integer $k \geq 3$ and an alphabet $\Sigma = \{a_1 < \cdots < a_k\}$ of integers. Consider the Champernowne-like sequence defined on Σ by concatenating all words of Σ^* in lexicographic order. Then, for all $N \geq 1$, we can find a valuation of Σ (i.e., a distribution of integral values for the letters of Σ) such that $\rho^{\mathrm{add}}_{\mathbf{x}}(n) = \binom{n+k-1}{k-1}$ for all $n \leq N$. However, it does not seem possible to find a sequence for which its additive complexity always reaches the upper bound. This already highlights the unusual fact that the underlying alphabet of the words plays a crucial role in additive complexity.

The classical theorem of Morse and Hedlund [27] characterizes ultimately periodic infinite words by means of their factor complexity. With the notion of additive complexity, we no longer have a characterization, only the implication below. The converse of Proposition 7 does not hold, as illustrated by several examples in Section 4.

▶ **Proposition 7.** *The additive complexity of an ultimately periodic word is bounded.*

Similarly, balanced words may be characterized through their abelian complexity. A word $\mathbf{x}$ is said to be C-*balanced* if $||u|_a - |v|_a| \leq C$ for all $a \in \Sigma$ and all factors u, v of $\mathbf{x}$ of equal length. Richomme, Saari and Zamboni [33, Lemma 3] proved that an infinite word $\mathbf{x}$ is C-balanced for some $C \geq 1$ if and only if $\rho^{\text{ab}}_{\mathbf{x}}$ is bounded. In our case, we only have one implication, as stated in Proposition 8, and we also provide an upper bound.

▶ **Proposition 8.** *Let $\Sigma = \{a_1 < \cdots < a_k\}$ and let $\mathbf{x}$ be a C-balanced word on Σ. Then the additive complexity of $\mathbf{x}$ is bounded by a constant. More precisely, we have $\rho^{\text{add}}_{\mathbf{x}}(n) \leq C\sum_{i=1}^{\lceil k/2 \rceil}(a_i - a_{k+1-i}) + 1$ for all $n \geq 0$.*

Proof. For all length-n factors y, z of $\mathbf{x}$ and $a \in \Sigma$, we have $||y|_a - |z|_a| \leq C$. So the largest possible gap between the sum of letters of y and the sum of letters of z is when, for all $i \in \{1, \ldots, \lceil k/2 \rceil\}$, $|y|_{a_i} = |z|_{a_i} + C$ and $|y|_{a_{k+1-i}} = |z|_{a_{k+1-i}} - C$, or vice versa (in short, we swap C letters from a_k to a_1, C others from a_{k-1} to a_2, and so on and so forth). ◀

Note that Proposition 8 is a particular case of [6, Theorem 4]. However, there are infinite words with bounded additive complexity and unbounded abelian complexity, making them not balanced. For an example, see Section 4.2.

Computing additive and abelian complexity might be "easy" in some cases. Recently, Shallit [39] provided a general method to compute the abelian complexity of an automatic sequence under some hypotheses.

▶ **Theorem 9** ([39, Thm. 1]). *Let $\mathbf{x}$ be a sequence that is automatic in some regular numeration system. Assume that*

1. *the abelian complexity $\rho^{\text{ab}}_{\mathbf{x}}$ of $\mathbf{x}$ is bounded above by a constant, and*
2. *the Parikh vectors of length-n prefixes of $\mathbf{x}$ form a synchronized sequence.*

Then $\rho^{\text{ab}}_{\mathbf{x}}$ is an automatic sequence and the DFAO computing it is effectively computable.

We obtain an adapted version in the framework of additive complexity.

▶ **Theorem 10.** *Let $\mathbf{x}$ be a sequence that is automatic in some additive numeration system. Assume that*

1. *the additive complexity $\rho^{\text{add}}_{\mathbf{x}}$ of $\mathbf{x}$ is bounded above by a constant, and*
2. *the Parikh vectors of length-n prefixes of $\mathbf{x}$ form a synchronized sequence.*

Then $\rho^{\text{add}}_{\mathbf{x}}$ is an automatic sequence and the DFAO computing it is effectively computable.

Proof. Let $\Sigma = \{a_1 < \cdots < a_k\} \subset \mathbb{N}$ be an ordered finite alphabet. The *weighted Parikh vector* of a finite word $w \in \Sigma^*$ is $\psi^*(w) = (a_1|w|_{a_1}, \ldots, a_k|w|_{a_k})$. Then two words x, y are additively equivalent if and only if $\sum_{a\in\Sigma}[\psi^*(x)]_a = \sum_{a\in\Sigma}[\psi^*(y)]_a$, where $[\psi^*(x)]_a$ designates the ath component of the vector $\psi^*(x)$. We adapt the proof of [39, Thm. 1] in the framework of the additive complexity. The steps to find the automaton computing the additive complexity $\rho^{\text{add}}_{\mathbf{x}}$ are the following:

1. Since the Parikh vectors of length-n prefixes of $\mathbf{x}$ form a synchronized sequence by assumption, so are the weighted Parikh vectors for arbitrary length-n factors $\mathbf{x}[i..i+n-1]$. This is expressible in first-order logic.
2. For $i \geq 0$ and $n \geq 1$, let us denote $\Delta_{\mathbf{x}}(i,n)$ the following integer

$$\Delta_{\mathbf{x}}(i,n) = \sum_{a\in\Sigma}[\psi^*(\mathbf{x}[i..i+n-1])]_a - \sum_{a\in\Sigma}[\psi^*(\mathbf{x}[0..n-1])]_a.$$

The additive complexity $\rho^{\text{add}}_{\mathbf{x}}$ is bounded if and only if there is a constant C such that the cardinality of the set $A^*_n := \{\Delta_{\mathbf{x}}(i,n) : i \geq 0\}$, is bounded above by C for all $n \geq 1$.

3. In this case, the range of possible values of A_n^* is finite (it may take at most $2C+1$ values) and can be computed algorithmically.
4. Once this range is known, there are finitely many possibilities for $\Delta_{\mathbf{x}}(i,n)$ for all $i \geq 0$. Then, we compute the set S of all of these possibilities.
5. Once we have S, we can test each of the finitely many values to see if it occurs for some n, and we obtain an automaton recognizing those n for which it does.
6. All the different automata can then be combined into a single DFAO computing $\rho_{\mathbf{x}}^{\mathrm{add}}(n)$, using the direct product construction.

This finishes the proof. ◀

▶ Remark 11. The advantage of the proof above is that it is constructive. However, in practice, it will be more convenient to use the so-called *semigroup trick* algorithm, as discussed in [40, § 4.11]. This algorithm should be used when a regular sequence is believed to be automatic, i.e., when it takes only finitely many values. The semigroup trick algorithm halts if and only if the sequence is automatic and produces a DFAO if this is the case. Therefore, Theorem 10 ensures that, under some mild hypotheses, the algorithm halts.

Theorem 10 may be applied to a particular family of infinite words: those that are generated by so-called Parikh-collinear morphisms. In recent years, combinatorists have been studying them; see, e.g., [16, 2, 35, 36].

▶ **Definition 12.** *A morphism $\varphi\colon \Sigma^* \to \Delta^*$ is* Parikh-collinear *if the Parikh vectors $\Psi(\varphi(a))$, $a \in \Sigma$, are collinear (or pairwise $\mathbb{Z}$-linearly dependent). In other words, the associated* adjacency matrix *of φ, i.e., the matrix whose columns are the vectors $\Psi(\varphi(a))$, for all $a \in \Sigma$, has rank* 1.

▶ **Theorem 13** ([35, 36])**.** *Let $\varphi\colon \Sigma^* \to \Sigma^*$ be a Parikh-collinear morphism prolongable on the letter a, and write $\mathbf{x} := \varphi^\omega(a)$. Then the abelian complexity function $\rho_{\mathbf{x}}^{\mathrm{ab}}$ of $\mathbf{x}$ is k-automatic for $k = \sum_{b\in\Sigma} |\varphi(b)|_b$. Moreover, the automaton generating $\rho_{\mathbf{x}}^{\mathrm{ab}}$ can be effectively computed given φ and a.*

Putting together Lemma 5 and Theorem 13, we obtain the following.

▶ **Corollary 14.** *Let $\mathbf{x}$ be a fixed point of a Parikh-collinear morphism. Then the abelian and additive complexity functions of $\mathbf{x}$ are bounded.*

▶ **Theorem 15.** *Let $\varphi\colon \Sigma^* \to \Sigma^*$ be a Parikh-collinear morphism prolongable on the letter a, and write $\mathbf{x} := \varphi^\omega(a)$. The additive complexity function $\rho_{\mathbf{x}}^{\mathrm{add}}$ of $\mathbf{x}$ is k-automatic for $k = \sum_{b\in\Sigma} |\varphi(b)|_b$. Moreover, the automaton generating $\rho_{\mathbf{x}}^{\mathrm{add}}$ can be effectively computed given φ and a.*

Proof. By Corollary 14, $\rho_{\mathbf{x}}^{\mathrm{add}}$ is bounded by a constant, so Item 1 of Theorem 10 is satisfied. Then Item 2 of of Theorem 10 holds by [36, Lemma 26]. Hence, Theorem 10 allows to finish the proof. ◀

We now give a detailed example of Theorem 15. Let $f\colon \{0,1,2\}^* \to \{0,1,2\}$ be defined by $0 \mapsto 012, 1 \mapsto 112002, 2 \mapsto \varepsilon$. Since the three vectors $\Psi(f(0)) = (1,1,1)$, $\Psi(f(1)) = (2,2,2)$ and $\Psi(f(2)) = (0,0,0)$ are collinear, it follows that f is Parikh-collinear. Consider $\mathbf{x} = 012112002112002\cdots$, the fixed point of f starting with 0. In [36], the authors proved that the abelian complexity of $\mathbf{x}$ is equal to the eventually periodic word $135(377)^\omega$. We have a similar result for additive complexity.

▶ **Proposition 16.** *Let $f\colon \{0,1,2\}^* \to \{0,1,2\}, 0 \mapsto 012, 1 \mapsto 112002, 2 \mapsto \varepsilon$. The additive complexity of the fixed point $\mathbf{x} = 0121120022112002\cdots$ of f is equal to $134(355)^\omega$.*

Proof. Computing $\sum_{a=0}^{2}|f(a)|_a = 3$, we know from classical results that $\mathbf{x}$ is 3-automatic. We thus know that $\mathbf{x}$ is generated by a 3-uniform morphism. Following the procedure of [35], we have $\mathbf{x} = \tau(h^\omega(0))$ with $h : 0,6 \mapsto 012,\ 1,4 \mapsto 134,\ 2,3,5 \mapsto 506$, and the coding $\tau : 0,5 \mapsto 0,\ 1,3 \mapsto 1$, and $2,4,6 \mapsto 2$.

In `Walnut`, we can compute the synchronized functions `fac0`, `fac1` and `fac2` that computes the number of letter 0, 1 and 2 in every factor of $\mathbf{x}$, see [36] for more details.

Next, we test whether the factors $u = \mathbf{x}[i..i+n-1]$ and $v = \mathbf{x}[j..j+n-1]$ of $\mathbf{x}$ are additively equivalent. For that, it is enough to check the equality between the quantities $|u|_1 + 2|u|_2$ and $|v|_1 + 2|v|_2$.

```
def addFacEq "?msd_3 Ep,q,r,s $fac1(i,n,p) & $fac2(i,n,q)
    & $fac1(j,n,r) & $fac2(j,n,s) & p+2*q=r+2*s":
```

Finally, we write that $\mathbf{x}[i..i+n-1]$ is a novel occurrence of a length-n factor of $\mathbf{x}$ representing its additive equivalence class and obtain a linear representation for the number of such positions i as follows:

```
eval addCompRepLin n "?msd_3 Aj j<i => ~$addFacEq(i,j,n)":
```

`Walnut` then returns a linear representation of size 55.

The first step is to take the linear representation computed by `Walnut`, and minimize it. The result is a linear representation of rank 7, using the algorithm in [9, § 2.3]. Once we have this linear representation, we can carry out the so-called semigroup trick algorithm, as discussed in [40, § 4.11]. As it terminates, we prove that the additive complexity of the word $\mathbf{x}$ is bounded, and takes on only the values $\{1,3,4,5\}$ for $n \geq 0$. Furthermore, it produces a 4-state DFAO computing the additive complexity, called `addCompExample`, that we display in Figure 1.

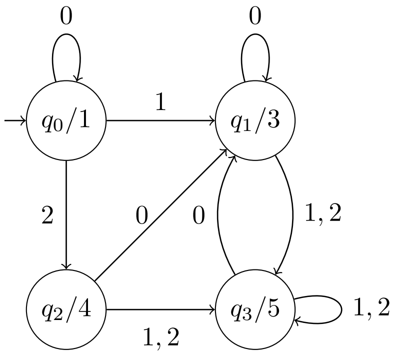


Figure 1 A four-state DFAO computing the additive complexity of the fixed point of $f\colon \{0,1,2\}^* \to \{0,1,2\}, 0 \mapsto 012, 1 \mapsto 112002, 2 \mapsto \varepsilon$.

By inspecting this DFAO, we easily prove that the additive complexity of $\mathbf{x}$ is $134(355)^\omega$. This could also be checked easily with `Walnut` with the following commands:

```
reg form3 msd_3 "0*(0|1|2)*0":
eval check3 "?msd_3 An ($form3(n) & n>=3) => addCompExample[n]=@3":
reg form5 msd_3 "0*(0|1|2)*(1|2)":
eval check5 "?msd_3 An ($form5(n) & n>=3) => addCompExample[n]=@5":
```

and both return `True`. Notice that these two forms cover all integers $n \geq 3$ and the first few values can be checked by hand. ◀

4 Different behaviors and curiosities

In this section, we exhibit different behaviors between the additive and abelian complexity functions by making use of the software Walnut. By Lemma 5, the behavior of additive complexity of a sequence is constrained by its abelian complexity. Here we show that the functions may behave differently; in particular, see Section 4.2.

4.1 Bounded additive and abelian complexities

4.1.1 The Tribonacci word

The *Tribonacci word* **tr** is the fixed point of the morphism $0 \mapsto 01$, $1 \mapsto 02$, $2 \mapsto 0$. This well-known word belongs to the family of episturmian words, a generalization of the famous Sturmian words. This word is *Tribonacci*-automatic, where the underlying numeration system is built on the sequence of *Tribonacci numbers* defined by $T(0) = 1$, $T(1) = 2$, $T(2) = 4$, and $T(n) = T(n-1) + T(n-2) + T(n-3)$ for all $n \geq 3$. Notice that this word is not the fixed point of a Parikh-collinear morphism; otherwise it would be k-automatic for some integer $k \geq 2$. A generalization of Cobham's theorem for substitutions [21] would then imply that **tr** is ultimately periodic. The possible values of the abelian complexity of the word **tr** were studied in [32, Thm. 1.4]. Also see Figure 2.

▶ **Theorem 17** ([32, Thm. 1.4]). *Let* **tr** *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$*,* $1 \mapsto 02$*,* $2 \mapsto 0$*. The abelian complexity function* $\rho^{\mathrm{ab}}_{\mathbf{tr}}$ *takes on only the values in the set* $\{3, 4, 5, 6, 7\}$ *for* $n \geq 1$*.*

This result was reproved by Shallit [39] using Walnut by providing an automaton computing $\rho^{\mathrm{ab}}_{\mathbf{tr}}$. Furthermore, this automaton allows us to prove that each value is taken infinitely often. We prove the following result concerning the additive complexity of the Tribonacci word. See again Figure 2.

▶ **Theorem 18.** *Let* **tr** *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$*,* $1 \mapsto 02$*,* $2 \mapsto 0$*. The additive complexity function* $\rho^{\mathrm{add}}_{\mathbf{tr}}$ *takes on only the values in the set* $\{3, 4, 5\}$ *for* $n \geq 1$*. Furthermore, each of the three values is taken infinitely often and it is computed by a 76-state Tribonacci DFAO.*

Proof. We reuse some ideas (especially, Walnut code) from [39, 41]. The Tribonacci word is stored as TRL in Walnut. The synchronized function rst takes the Tribonacci representations of m and n in parallel and accepts if $(n)_T$ is the right shift of $(m)_T$. In Walnut, the following three predicates allow us to obtain DFAO's that compute the maps $n \mapsto |\mathbf{tr}[0..n-1]|_a$ for $a \in \{0, 1, 2\}$, i.e., the number of letters $0, 1, 2$ in the length-n prefix of the Tribonacci word **tr**. Note that the predicates are obtained using a special property of **tr**; for a full explanation, see [39, Sec. 3].

```
def tribsync0 "?msd_trib Ea Eb (s=a+b) & ((TRL[n]=@0)=>b=0)
    & ((TRL[n]=@1)=>b=1) & $rst(n,a)":
def tribsync1 "?msd_trib Ea Eb Ec (s=b+c) & ((TRL[a]=@0)=>c=0)
    & ((TRL[a]=@1)=>c=1) & $rst(n,a) & $rst(a,b)":
def tribsync2 "?msd_trib Ea Eb Ec Ed (s=c+d) & ((TRL[b]=@0)=>d=0)
    & ((TRL[b]=@1)=>d=1) & $rst(n,a) & $rst(a,b) & $rst(b,c)":
```

From now on, we follow the same steps as Proposition 16. First, we compute the Tribonacci synchronized functions $n \mapsto |\mathbf{tr}[i..i+n-1]|_a$ for $a \in \{0, 1, 2\}$, that are

```
def tribFac0 "?msd_trib Aq Ar ($tribsync0(i+n,q)
    & $tribsync0(i,r)) => (q=r+s)":
def tribFac1 "?msd_trib Aq Ar ($tribsync1(i+n,q)
    & $tribsync1(i,r)) => (q=r+s)":
def tribFac2 "?msd_trib Aq Ar ($tribsync2(i+n,q)
    & $tribsync2(i,r)) => (q=r+s)":
```

Next, we compute the additive equivalence between two factors, that is the following Tribonacci synchronized function

```
def tribAddFacEq "?msd_trib Ep,q,r,s $tribFac1(i,n,p) & $tribFac2(i,n,q)
    & $tribFac1(j,n,r) & $tribFac2(j,n,s) & p+2*q=r+2*s":
```

Finally, we obtain a linear representation, as defined at the end of Section 2, of the additive complexity as follows

```
eval tribAddCompRepLin n "?msd_trib Aj j<i => ~$tribAddFacEq(i,j,n)":
```

And `Walnut` then returns a linear representation of size 184. Then we apply the same procedure than in Proposition 16.

After minimization, the result is a linear representation of rank 62 and we carry out the semigroup trick. This algorithm terminates, which proves that the additive complexity of the Tribonacci word is bounded, and takes on only the values $\{1, 3, 4, 5\}$ for $n \geq 0$. Furthermore, it produces a 76-state DFAO computing the additive complexity. In `Walnut`, let us import this DFAO under the name `TAC`. To show that each value appears infinitely often, we test the following three predicates

```
eval tribAddComp_3 "?msd_trib An Em (m>n) & TAC[m]=@3":
eval tribAddComp_4 "?msd_trib An Em (m>n) & TAC[m]=@4":
eval tribAddComp_5 "?msd_trib An Em (m>n) & TAC[m]=@5":
```

and `Walnut` then returns `TRUE` each time. ◀

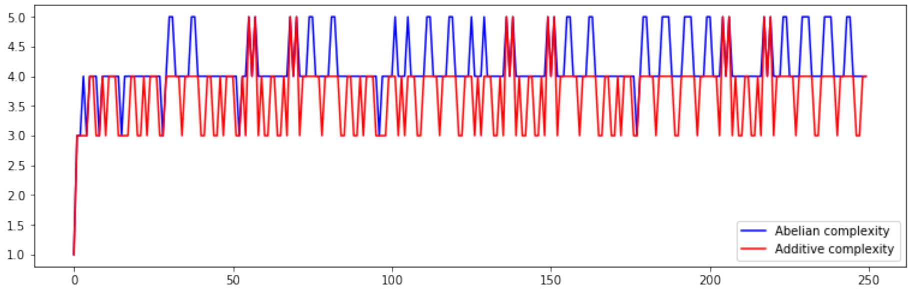


Figure 2 The first few values of the abelian and additive complexities for the Tribonacci word.

▶ Remark 19. From the automaton, which is too large to display here, it is easy to find infinite families for each value of the additive complexity function. Indeed, it suffices to detect a loop in the automaton leading to a final state for each value. For instance, we have the following infinite families:

(a) If $(n)_T = 100(100)^k$, for $k \geq 0$, then $\rho^{\mathrm{add}}_{\mathbf{tr}}(n) = 3$.

(b) If $(n)_T = 1101(01)^k$, for $k \geq 0$, then $\rho^{\mathrm{add}}_{\mathbf{tr}}(n) = 4$.

(c) If $(n)_T = 1101001100(1100)^k$, for $k \geq 0$, then $\rho^{\mathrm{add}}_{\mathbf{tr}}(n) = 5$.

One can check with Walnut that these infinite families are convenient with the following commands

```
reg form3 msd_trib "0*100(100)*":
reg form4 msd_trib "0*1101(01)*":
reg form5 msd_trib "0*1101001100(1100)*":
eval check3 "?msd_trib An ($form3(n) & n>=1) => TAC[n]=@3":
eval check4 "?msd_trib An ($form4(n) & n>=1) => TAC[n]=@4":
eval check5 "?msd_trib An ($form5(n) & n>=1) => TAC[n]=@5":
```

which returns TRUE for each command. One can also notice that from the automaton, we can build infinitely many infinite families of solutions of each of those values. However, the question about the respective proportion of solutions remains open.

▶ Remark 20. With Walnut, we can also build a DFAO computing the minimum (resp., maximum) possible sum of a length-n block occurring in $\mathbf{tr}$. Furthermore, for each n, every possible sum between these two extremes actually occurs for some length-n factor in $\mathbf{tr}$.

4.1.2 The generalized Thue–Morse word on three letters

We introduce a family of words over three letters that are closed to a generalization of the Thue–Morse word.

▶ **Definition 21.** *Let ℓ, m be integers such that $1 \leq \ell < m$. The (ℓ, m)-*Thue–Morse word *$\mathbf{t}_{\ell,m}$ is the fixed point of the morphism $0 \mapsto 0\ell m$, $\ell \mapsto \ell m 0$, $m \mapsto m 0 \ell$.*

In the case where $\ell = 1$ and $m = 2$, we find the so-called *ternary Thue–Morse word* $\mathbf{t}_3$, which is the fixed point of the morphism $0 \mapsto 012$, $1 \mapsto 120$, $2 \mapsto 201$. This word is a natural generalization of the ubiquitous Thue–Morse sequence, since it corresponds to the sum-of-digit function in base 3, taken mod 3.

▶ **Theorem 22** ([25, Thm. 4.1])**.** *Consider the ternary Thue–Morse word $\mathbf{t}_3$, i.e., the fixed point of the morphism $0 \mapsto 012$, $1 \mapsto 120$, $2 \mapsto 201$. The abelian complexity function $\rho^{\mathrm{ab}}_{\mathbf{t}_3}$ is the periodic infinite word $13(676)^\omega$.*

▶ **Theorem 23.** *Consider the ternary Thue–Morse word $\mathbf{t}_3$, i.e., the fixed point of the morphism $0 \mapsto 012$, $1 \mapsto 120$, $2 \mapsto 201$. The additive complexity function $\rho^{\mathrm{add}}_{\mathbf{t}_3}$ is the periodic infinite word 135^ω.*

Proof. The following Walnut provides a linear representation of size 138 for the additive complexity of $\mathbf{t}_3$:

```
morphism h "0->012 1->120 2->201":
promote TMG h:

def tmgPref0 "?msd_3 Er,t n=3*t+r & r<3 & (r=0 => s=t)
    & ((r=1 & TMG[n-1]=@0) => s=t+1)
    & ((r=1 & (TMG[n-1]=@1 | TMG[n-1]=@2)) => s=t)
    & ((r=2 & (TMG[n-1]=@0 | TMG[n-1]=@1)) => s=t+1)
    & ((r=2 & TMG[n-1]=@2) => s=t)":
def tmgPref1 "?msd_3 Er,t n=3*t+r & r<3 & (r=0 => s=t)
    & ((r=1 & TMG[n-1]=@1) => s=t+1)
    & ((r=1 & (TMG[n-1]=@0 | TMG[n-1]=@2)) => s=t)
    & ((r=2 & (TMG[n-1]=@1 | TMG[n-1]=@2)) => s=t+1)
    & ((r=2 & TMG[n-1]=@0) => s=t)":
def tmgPref2 "?msd_3 Eq,r $tmgPref0(n,q) & $tmgPref1(n,r) & q+r+s=n":

def tmgFac0 "?msd_3 Et,u $tmgPref0(i+n,t) & $tmgPref0(i,u) & s+u=t":
def tmgFac1 "?msd_3 Et,u $tmgPref1(i+n,t) & $tmgPref1(i,u) & s+u=t":
def tmgFac2 "?msd_3 Et,u $tmgPref2(i+n,t) & $tmgPref2(i,u) & s+u=t":

def tmgAddFacEq "?msd_3 Ep,q,r,s $tmgFac1(i,n,p) & $tmgFac2(i,n,q)
    & $tmgFac1(j,n,r) & $tmgFac2(j,n,s) & p+2*q=r+2*s":

eval tmgAddCompRepLin n "?msd_3 Aj j<i => ~$tmgAddFacEq(i,j,n)":
```

The end of the proof is the same as for Theorem 18. The size of the minimal linear representation is 13 and the semigroup trick algorithm terminates and produces the 3-state DFAO of Figure 3. The result follows immediately. ◀

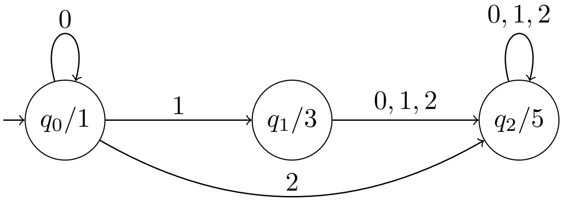


Figure 3 A DFAO computing the additive complexity of the $(1, 2)$-Thue–Morse word.

Changing the letters ℓ and m does not modify the abelian complexity, so for all $1 \le \ell < m$, we have $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}} = \rho^{\mathrm{ab}}_{\mathbf{t}_3}$. However, additive complexity might change over a different alphabet. In the particular case where $\ell = 1$ and $m = 2$, the following gives an alternative proof of Theorem 23 with only combinatorial tools. Note that the statement on $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}}$ was also proven in [25], but we provide here a simpler and more concise proof.

▶ **Theorem 24.** *Let ℓ, m be integers such that $1 \le \ell < m$. Consider the (ℓ, m)-Thue–Morse word, i.e., the fixed point of the morphism $0 \mapsto 0\ell m$, $\ell \mapsto \ell m 0$, $m \mapsto m 0 \ell$. Then its abelian complexity satisfies $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}} = 136(766)^{\omega}$ and its additive complexity satisfies $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}} = \rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}}$ if $m \neq 2\ell$, and $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}} = 135^{\omega}$ if $m = 2\ell$.*

Proof. We clearly have $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(0) = 1$, $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(1) = 3$, and $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(2)$ is equal to 5 or 6 depending on whether $m = 2\ell$ or not. We examine length-n factors of $\mathbf{t}_{\ell,m}$ for $n \ge 2$. Each such factor can be written as $y = pf(x)s$ where f is the morphism $0 \mapsto 0\ell m$, $\ell \mapsto \ell m 0$, $m \mapsto m 0 \ell$ of Definition 21 and p (resp., s) is a proper suffix (resp., prefix) of an image $f(a)$ for $a \in \{0, \ell, m\}$. In particular, note that $p, s \in \{\varepsilon, 0, \ell, m, 0\ell, \ell m, m0\}$. In the following, we examine the *weight* of y, which is the quantity $0 \cdot |y|_0 + \ell \cdot |y|_\ell + m \cdot |y|_m$. More precisely, we count how many different weights y can have, which in turn gives the number of different additive equivalence classes.

First assume that $|y| = 3n$ for some $n \ge 1$. Then we have two cases depending on whether p, s are empty or not. If $p = s = \varepsilon$, then $|x| = n$ and this case corresponds to the first line of Table 1. Otherwise, $|x| = n - 1$ and $|ps| = 3$. In that case, since the roles of p and s are symmetric when computing the weight of the factor, all the possible cases are depicted in Table 1.

Table 1 The possible weights of factors of the (ℓ, m)-Thue–Morse word $\mathbf{t}_{\ell,m}$ of the form $y = pf(x)s$ where $|y| = 3n$ for some $n \geq 1$.

p	s	$\lvert y\rvert_0$	$\lvert y\rvert_\ell$	$\lvert y\rvert_m$	$0 \cdot \lvert y\rvert_0 + \ell \cdot \lvert y\rvert_\ell + m \cdot \lvert y\rvert_m$
ε	ε	n	n	n	$\ell n + mn$
0	0ℓ	$n+1$	n	$n-1$	$\ell n + m(n-1)$
0	ℓm	n	n	n	$\ell n + mn$
0	$m0$	$n+1$	$n-1$	n	$\ell(n+1) + m(n-1)$
ℓ	0ℓ	n	$n+1$	$n-1$	$\ell n + m(n+1)$
ℓ	ℓm	$n-1$	$n+1$	n	$\ell(n-1) + mn$
ℓ	$m0$	n	n	n	$\ell n + mn$
m	0ℓ	n	n	n	$\ell n + mn$
m	ℓm	$n-1$	n	$n+1$	$\ell n + m(n+1)$
m	$m0$	n	$n-1$	$n+1$	$\ell(n-1) + m(n+1)$

From the third, fourth and fifth columns of the table, we observe that there are seven different abelian classes (only the class where $|y|_0 = |y|_\ell = |y|_m = n$ appears more than once) and this proves that $\rho^{\mathrm{ab}}_{\mathbf{t}_{\ell,m}}(3n) = 7$. The corresponding weights of these seven abelian classes can be written as $(n-1)\cdot(\ell+m)+\delta$ with $\delta \in \{\ell, m, 2\ell, \ell+m, 2m, 2\ell+m, \ell+2m\}$. Since

$$\begin{aligned}\ell < \min\{m, 2\ell\} \leq{}& \max\{m, 2\ell\} < \ell + m \\ &< \min\{2m, 2\ell+m\} \leq \max\{2m, 2\ell+m\} < \ell + 2m,\end{aligned}$$

this now proves that $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(3n)$ is equal to 7 if $m \neq 2\ell$, and to 5 otherwise.

The proof of the remaining cases $|y| = 3n+1$ and $|y| = 3n+2$ are very similar and are left to the reader. Finally, we obtain that $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(3n+1)$ and $\rho^{\mathrm{add}}_{\mathbf{t}_{\ell,m}}(3n+2)$ are equal to 6 if $m \neq 2\ell$, and to 5 otherwise. ◀

4.2 Bounded additive and unbounded abelian complexities: a variant of the Thue–Morse word

Thue introduced a variation of his sequence that is sometimes called the *ternary squarefree Thue–Morse word*, and abbreviated as **vtm** (the letter "v" stands for "variant"). It is the sequence [42, A036577] in the OEIS; for more on the word **vtm**, see [7].

▶ **Definition 25** (Variant of Thue–Morse). *We let* **vtm** *be the fixed point of* $f : 0 \mapsto 012, 1 \mapsto 02, 2 \mapsto 1$*, starting with* 0.

The abelian complexity of the variant of the Thue–Morse word is unbounded.

▶ **Theorem 26** ([10, Cor. 1]). *Let* **vtm** *be the fixed point of* $f : 0 \mapsto 012, 1 \mapsto 02, 2 \mapsto 1$*, starting with* 0*. Its abelian complexity is* $O(\log n)$ *with constant approaching* $3/4$ *(assuming base-*2 *logarithm), and it is* $\Omega(1)$ *with constant* 3.

However, we prove that the additive complexity of the word **vtm** is bounded.

▶ **Theorem 27.** *Let* **vtm** *be the fixed point of* $f : 0 \mapsto 012, 1 \mapsto 02, 2 \mapsto 1$*, starting with* 0*. Its additive complexity is the periodic infinite word* 13^ω.

Proof. Let $n \geq 1$ and $x \in \mathcal{L}_n(\mathbf{vtm})$. Let us prove that $\sum_{a=0}^{2} a \cdot |x|_a \in \{n-1, n, n+1\}$. Write $x = pf(y)s$ where p (resp., s) is a proper suffix (resp., prefix) of an image $f(a)$, $a \in \{0,1,2\}$. Then we have $p \in \{\varepsilon, 12, 2\}$ and $s \in \{\varepsilon, 0, 01\}$. By definition of the morphism f, observe that $|f(y)|_2 = |f(y)|_0$. Therefore, depending on the words p and s, $|x|_2 = |x|_0 + c$ with $c \in \{-1, 0, 1\}$, which suffices since $|x|_0 + |x|_1 + |x|_2 = n$. ◀

Therefore, the word **vtm** has unbounded abelian complexity and bounded additive complexity; also see Figure 4. In particular, [33, Lemma 3] implies that **vtm** cannot be balanced, so there exist non-balanced infinite words with bounded additive complexity. Another example exhibiting the same behavior for its abelian and additive complexity is given in [5].

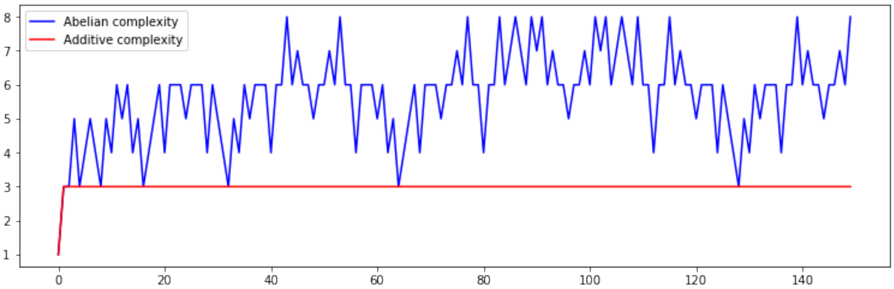


Figure 4 The first few values of the abelian and additive complexities for the variant of the Thue–Morse word.

4.3 Unbounded additive and abelian complexities

In this short section, we exhibit a word such that both its additive and abelian complexities are both unbounded.

▶ **Theorem 28** ([17, Thm. 1 and Cor. 1]). *Let $\mathbf{x}$ be the fixed point of the Thue–Morse-like morphism $0 \mapsto 01$, $1 \mapsto 12$, $2 \mapsto 20$. Then $\rho_{\mathbf{x}}^{\text{add}}(n) = 2\lfloor \log_2 n \rfloor + 3$ for all $n \geq 1$. In particular, the sequence $(\rho_{\mathbf{x}}^{\text{add}}(n))_{n \geq 0}$ is 2-regular.*

Recall that, for an integer $k \geq 1$, a word w is an *abelian k-power* if we can write $w = x_1 x_2 \cdots x_k$ where each x_i, $i \in \{1, \ldots, k\}$, is a permutation of x_1. For instance, `reap`·`pear` and `de` · `ed` · `ed` are respectively an abelian square and cube in English. Similarly, w is an *additive k-power* if we can write $w = x_1 x_2 \cdots x_k$ with $|x_i| = |x_1|$ for all $i \in \{1, \ldots, k\}$ and $x_1 \sim_{\text{add}} x_2 \sim_{\text{add}} \cdots \sim_{\text{add}} x_k$. The length of each x_i, $i \in \{1, \ldots, k\}$, is called the *order* of w. As mentioned in the introduction, the following result is one of the main results known on additive complexity.

▶ **Theorem 29** ([5, Thm. 2.2]). *Let $\mathbf{x}$ be an infinite word over a finite subset of $\mathbb{Z}$. If $\rho_{\mathbf{x}}^{\text{add}}$ is bounded, then $\mathbf{x}$ contains an additive k-power for every positive integer k.*

▶ **Proposition 30.** *Let $\mathbf{w}$ be the fixed point of the morphism $0 \mapsto 03$, $1 \mapsto 43$, $3 \mapsto 1$, $4 \mapsto 01$. Then $\rho_{\mathbf{w}}^{\text{add}}$ is unbounded.*

Proof. In [14] it is shown that $\mathbf{w}$ is additive-cube-free. The result then follows from Theorem 29. ◀

However, for the latter word $\mathbf{w}$, it seems interesting to study $\rho^{\text{ab}}_{\mathbf{w}} - \rho^{\text{add}}_{\mathbf{w}}$, since these two complexities are very close. Indeed, surprisingly the first time these two complexities are different appears at $n = 23$, as the two factors 11011031430110343430314 and 30310110110314303434303 are additively but not abelian equivalent. Also, notice that every additive square of the word $\mathbf{w}$ is an abelian square [14, Theorem 5.1]. Together with the fact that this word is additive-cube-free, it shows that abelian and additive properties of this word are relatively close. Indeed, Figure 5 illustrates that the values of the difference between the additive and abelian complexity functions is close to 0. This motivates the study of the next section.

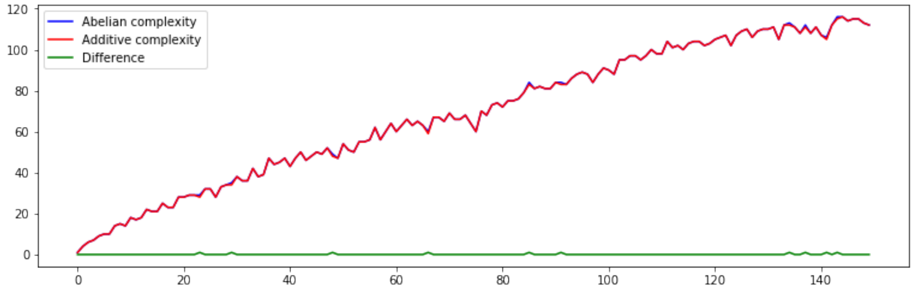


Figure 5 The first few values of the abelian and additive complexities as well as their difference for the fixed point of the morphism $0 \mapsto 03$, $1 \mapsto 43$, $3 \mapsto 1$, $4 \mapsto 01$.

5 Equality between abelian and additive complexities

It is clear that abelian complexity does not depend on the values of the alphabet, in contrast with additive complexity. A map $v : A \to \mathbb{N}$ is called a *valuation* over an alphabet A. One might consider the following question.

▶ **Question 31.** Given an alphabet A, is there a valuation such that the additive complexity of a given sequence is equal to its abelian complexity?

For instance for the word $\mathbf{vtm}$, defined originally over the alphabet $\{0, 1, 2\}$, we have already proved in Theorem 27 that $\rho^{\text{add}}_{\mathbf{vtm}}(n) = 3$ for all $n \geq 1$. However, over the alphabet $\{0, 1, 3\}$, i.e., changing 2 into 3, (resp., $\{0, 1, 4\}$), one can easily check that the first time that the additive and abelian complexities are not equal is for $n = 11$ (resp., $n = 43$). But, over the alphabet $\{0, 1, 5\}$, we have observed that both complexities are equal up to $n = 50000$. The main idea is that if the value of a letter is sufficiently large compared to the other values, then two additively equivalent factors are also abelian equivalent. Using this idea, we prove the following theorem.

▶ **Theorem 32.** *Consider the fixed point* $\mathbf{vtm}_\lambda$ *of the morphism* $f_\lambda : 0 \mapsto 01\lambda, 1 \mapsto 0\lambda, \lambda \mapsto 1$, *where* λ *is a non-negative integer and* $\lambda \geq 2$. *For all* $\lambda \geq 5$, *we have* $\rho^{\text{add}}_{\mathbf{vtm}_\lambda}(n) = \rho^{\text{ab}}_{\mathbf{vtm}_\lambda}(n)$.

Proof. By Lemma 5, it is sufficient to prove that two factors are additively equivalent if and only if they are abelian equivalent.

Let $x, y \in \mathcal{L}_n(\mathbf{vtm}_\lambda)$ such that $x \sim_{\text{add}} y$. Write $x = pf_\lambda(x')s$ where p (resp., s) is a proper suffix (resp., prefix) of an image $f_\lambda(a)$ with $a \in \{0, 1, \lambda\}$. Observe that $p \in \{\varepsilon, \lambda, 1\lambda\}$ and $s \in \{\varepsilon, 0, 01\}$. Also, by definition of the morphism f_λ, we have $|f_\lambda(x')|_\lambda = |f_\lambda(x')|_0$. Therefore, depending on the words p and s, we have $|x|_\lambda = |x|_0 + c_x$ for some $c_x \in \{-1, 0, 1\}$.

In a similar way, we have $|y|_\lambda = |y|_0 + c_y$ for some $c_y \in \{-1, 0, 1\}$. By the assumption that $x \sim_{\text{add}} y$, we have $0|x|_0 + 1|x|_1 + \lambda|x|_\lambda = 0|y|_0 + 1|y|_1 + \lambda|y|_\lambda$. From the previous observations, we may write this equality as

$$|x|_0 + |x|_1 + |x|_\lambda + (\lambda - 2)|x|_\lambda + c_x = |y|_0 + |y|_1 + |y|_\lambda + (\lambda - 2)|y|_\lambda + c_y.$$

Since $|x|_0 + |x|_1 + |x|_\lambda = n = |y|_0 + |y|_1 + |y|_\lambda$, we have $(\lambda - 2)(|x|_\lambda - |y|_\lambda) = c_y - c_x$. However, $c_y - c_x \in \{-2, \ldots, 2\}$ implies that $|x|_\lambda = |y|_\lambda$ and $c_y = c_x$, since $\lambda - 2 \geq 3$. Thus, $|x|_0 = |y|_0$. Since $|x| = n = |y|$, we also have $|x|_1 = |y|_1$, and then that x and y are abelian equivalent. This ends the proof. ◀

For C-balanced words over an alphabet of fixed size k, we prove that it is always possible to find a valuation for the alphabet such that the additive complexity is the same as the abelian complexity.

▶ **Theorem 33.** *Let $k, C \geq 1$ be two integers. There exists an alphabet $\Sigma \subset \mathbb{N}$ of size k such that, for each C-balanced word $\mathbf{w}$ over Σ, we have $\rho^{\text{add}}_{\mathbf{w}} = \rho^{\text{ab}}_{\mathbf{w}}$.*

Proof. Such as in the proof of Theorem 32, we prove that over the alphabet Σ, two additively equivalent same-length factors of $\mathbf{w}$ are also abelian equivalent. Let $\Sigma = \{a_1, \ldots, a_k\}$ be a subset of $\mathbb{N}$ such that

$$a_1 = 0, a_2 = 1 \quad \text{and} \quad (a_1 + \cdots + a_{j-1})C < a_j \quad \text{for all } 2 \leq j \leq k. \tag{1}$$

Now take $x, y \in \mathcal{L}_n(\mathbf{w})$ with $x \sim_{\text{add}} y$. This condition can be rewritten as

$$a_1(|x|_{a_1} - |y|_{a_1}) + \cdots + a_{k-1}(|x|_{a_{k-1}} - |y|_{a_{k-1}}) = a_k(|y|_{a_k} - |x|_{a_k}). \tag{2}$$

Observe that the balancedness of $\mathbf{w}$ together with Inequalities (1) imply that the left-hand side of Equality (2) belongs to the set $\{-a_k + 1, \ldots, a_k - 1\}$. Since the right-hand side of Equality (2) is a multiple of a_k, we must have

$$\begin{cases} |x|_{a_k} = |y|_{a_k}, \\ a_1(|x|_{a_1} - |y|_{a_1}) + \cdots + a_{k-1}(|x|_{a_{k-1}} - |y|_{a_{k-1}}) = 0. \end{cases} \tag{3}$$

Using similar reasoning, replacing Equality (2) with Equalities (3), we deduce that $|x|_{a_{k-1}} = |y|_{a_{k-1}}$. Continuing in this fashion, we prove that $|x|_a = |y|_a$ for every $a \in \Sigma$, which is enough. ◀

▶ Remark 34. Since the Tribonacci word $\mathbf{tr}$ is 2-balanced, Theorem 33 implies that over the alphabet $\{0, 1, 3\}$, its additive complexity is equal to its abelian complexity.

6 Abelian and additive powers

In [22, 23], abelian powers of Sturmian words were examined. In particular, the following result was obtained for the Fibonacci word $\mathbf{f} = 0100101001001010010100\cdots$, which is the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 0$. Also, see the sequence [42, A336487] in the OEIS.

▶ **Proposition 35** ([22, 23]). *Let $k \geq 1$ be an integer and consider the Fibonacci word $\mathbf{f}$, i.e., the fixed point of the morphism $0 \mapsto 01, 1 \mapsto 0$. Then $\mathbf{f}$ has an abelian k-power of order n if and only if $\lfloor k\varphi n \rfloor \equiv 0, -1 \pmod{k}$, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.*

For instance, when $k = 3$, we can compute an 11-state DFA accepting, in Fibonacci representations, exactly those n for which there is an abelian cube of order n in $\mathbf{f}$.

As Arnoux-Rauzy and episturmian sequences generalize Sturmian sequences, it is quite natural to try to understand the orders of abelian and additive powers in these sequences. An archetypical example is the Tribonacci word $\mathbf{tr}$ (recall Section 4.1.1). We obtain the following results on squares and cubes using Walnut and the fact that the frequency of each letter $0, 1, 2$ in $\mathbf{tr}$ is Tribonacci-synchronized (see [40, § 10.12] and/or Section 4.1.1).

▶ **Theorem 36** ([40, Thm. 10.13.5]). *Let* $\mathbf{tr}$ *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$, $1 \mapsto 02$. *There are abelian squares of all orders in* $\mathbf{tr}$. *Furthermore, if we consider two abelian squares* xx' *and* yy' *to be equivalent if* $x \sim_{\mathrm{ab}} y$, *then every order has either one or two abelian squares. Both possibilities occur infinitely often.*

▶ **Theorem 37.** *Let* $\mathbf{tr}$ *be the Tribonacci word, i.e., the fixed point of the morphism* $0 \mapsto 01$, $1 \mapsto 02$. *There is a (minimal) Tribonacci automaton of* 1169 *(resp.,* 4927*) states recognizing the Tribonacci representation of those* n *for which there is an abelian (resp., additive) cube of order* n *in* $\mathbf{tr}$.

Proof. For the part about abelian cubes, see [40, p. 295]. See also the respective sequences [42, A345717,A347752] in the OEIS. For the additive cubes, we can determine the orders of additive cubes in $\mathbf{tr}$ with the following function:

```
def tribAddCube "?msd_trib Ei $tribAddFacEq(i,i+n,n)
    & $tribAddFacEq(i,i+2*n,n)":
```

where `tribAddFacEq` is the function defined in the proof of Theorem 18. This leads to a Tribonacci automaton of 4927 states. ◀

We also note that Theorems 18 and 29 imply the existence of additive k-powers in $\mathbf{tr}$ for all $k \geq 1$. When $k = 2$, additive squares exist for all orders by Theorem 36. For $k = 3$, orders of additive cubes are given in Theorem 37 by a large automaton, and no simple description seems to be possible. When $k = 4$, the same procedure on Walnut requires a much larger memory, and it appears that a simple desk computer cannot achieve it. We naturally wonder about larger powers and leave the following as a relatively difficult open question.

▶ **Problem 38.** Characterize the orders of additive k-powers in the Tribonacci word $\mathbf{tr}$.

It is shown in [15] that the behavior of the abelian complexity of Arnoux-Rauzy words might be erratic. In particular, there exist such words with unbounded abelian complexity. We leave open the research direction of studying the additive complexity of such words and episturmian sequences. For instance, is there a result similar to Proposition 35 in the framework of additive powers?

References

1 Jean-Paul Allouche, John Campbell, Jeffrey Shallit, and Manon Stipulanti. The reflection complexity of sequences over finite alphabets. arXiv preprint. doi:10.48550/arXiv.2406.09302.

2 Jean-Paul Allouche, Michel Dekking, and Martine Queffélec. Hidden automatic sequences. *Comb. Theory*, 1:15, 2021. Id/No 20. doi:10.5070/C61055386.

3 Jean-Paul Allouche and Jeffrey Shallit. *Automatic sequences: Theory, applications, generalizations*. Cambridge University Press, Cambridge, 2003. doi:10.1017/CBO9780511546563.

4 Jonathan Andrade and Lucas Mol. Avoiding abelian and additive powers in rich words, 2024. arXiv preprint. doi:10.48550/arXiv.2408.15390.

5 Hayri Ardal, Tom Brown, Veselin Jungić, and Julian Sahasrabudhe. On abelian and additive complexity in infinite words. *Integers*, 12(5):795–804, 2012. doi:10.1515/integers-2012-0005.

6 Graham Banero. On additive complexity of infinite words. *J. Integer Seq.*, 16(1):Article 13.1.5, 20, 2013.

7 Jean Berstel. Sur les mots sans carré définis par un morphisme. In *Automata, languages and programming (Sixth Colloq., Graz, 1979)*, volume 71 of *Lecture Notes in Comput. Sci.*, pages 16–25. Springer, Berlin-New York, 1979. doi:10.1007/3-540-09510-1_2.

8 Jean Berstel and Dominique Perrin. The origins of combinatorics on words. *Eur. J. Comb.*, 28(3):996–1022, 2007. doi:10.1016/j.ejc.2005.07.019.

9 Jean Berstel and Christophe Reutenauer. *Noncommutative Rational Series With Applications*, volume 137 of *Encyclopedia of Mathematics and Its Applications*. Cambridge Univ. Press, 2011.

10 Francine Blanchet-Sadri, James D. Currie, Narad Rampersad, and Nathan Fox. Abelian complexity of fixed point of morphism $0 \mapsto 012$, $1 \mapsto 02$, $2 \mapsto 1$. *Integers*, 14:A11, 2014. URL: http://math.colgate.edu/%7Eintegers/o11/o11.Abstract.html.

11 Thomas C. Brown and Allen R. Freedman. Arithmetic progressions in lacunary sets. *Rocky Mountain J. Math.*, 17:587–596, 1987.

12 Tom Brown. Approximations of additive squares in infinite words. *Integers*, 12(5):805–809, a22, 2012. doi:10.1515/integers-2012-0006.

13 Arturo Carpi and Cristiano Maggi. On synchronized sequences and their separators. *Theor. Inform. Appl.*, 35(6):513–524, 2001. doi:10.1051/ita:2001129.

14 Julien Cassaigne, James D. Currie, Luke Schaeffer, and Jeffrey Shallit. Avoiding three consecutive blocks of the same size and same sum. *J. ACM*, 61(2):Art. 10, 17, 2014. doi:10.1145/2590775.

15 Julien Cassaigne, Sébastien Ferenczi, and Luca Q. Zamboni. Imbalances in Arnoux-Rauzy sequences. *Ann. Inst. Fourier*, 50(4):1265–1276, 2000. doi:10.5802/aif.1792.

16 Julien Cassaigne, Gwénaël Richomme, Kalle Saari, and Luca Q. Zamboni. Avoiding abelian powers in binary words with bounded abelian complexity. *Int. J. Found. Comput. Sci.*, 22(4):905–920, 2011. doi:10.1142/S0129054111008489.

17 Jin Chen, Zhixiong Wen, and Wen Wu. On the additive complexity of a Thue-Morse-like sequence. *Discrete Appl. Math.*, 260:98–108, 2019. doi:10.1016/j.dam.2019.01.008.

18 Alan Cobham. Uniform tag sequences. *Math. Systems Theory*, 6:164–192, 1972. doi:10.1007/BF01706087.

19 Ethan M. Coven and G. A. Hedlund. Sequences with minimal block growth. *Math. Systems Theory*, 7:138–153, 1973. doi:10.1007/BF01762232.

20 James Currie and Narad Rampersad. Recurrent words with constant abelian complexity. *Adv. Appl. Math.*, 47(1):116–124, 2011. doi:10.1016/j.aam.2010.05.001.

21 Fabien Durand. Cobham's theorem for substitutions. *Journal of the European Mathematical Society*, 13(6):1799–1814, September 2011. doi:10.4171/jems/294.

22 Gabriele Fici, Alessio Langiu, Thierry Lecroq, Arnaud Lefebvre, Filippo Mignosi, Jarkko Peltomäki, and Élise Prieur-Gaston. Abelian powers and repetitions in Sturmian words. *Theoret. Comput. Sci.*, 635:16–34, 2016. doi:10.1016/j.tcs.2016.04.039.

23 Gabriele Fici, Alessio Langiu, Thierry Lecroq, Arnaud Lefebvre, Filippo Mignosi, and Élise Prieur-Gaston. Abelian repetitions in Sturmian words. In *Developments in Language Theory*, volume 7907 of *Lecture Notes in Comput. Sci.*, pages 227–238. Springer, Heidelberg, 2013. doi:10.1007/978-3-642-38771-5_21.

24 Lorenz Halbeisen and Norbert Hungerbühler. An application of Van der Waerden's theorem in additive number theory. *INTEGERS*, 0:#A7, 2000. Available online at https://math.colgate.edu/~integers/a7/a7.pdf.

25 Idrissa Kaboré and Boucaré Kientéga. Abelian complexity of Thue-Morse word over a ternary alphabet. In *Combinatorics on words*, volume 10432 of *Lecture Notes in Comput. Sci.*, pages 132–143. Springer, Cham, 2017. doi:10.1007/978-3-319-66396-8_13.

26 M. Lothaire. *Combinatorics on words*. Cambridge Mathematical Library. Cambridge University Press, Cambridge, 1997. doi:10.1017/CBO9780511566097.

27 Marston Morse and Gustav A. Hedlund. Symbolic dynamics. II: Sturmian trajectories. *Am. J. Math.*, 62:1–42, 1940. doi:10.2307/2371431.

28 Hamoon Mousavi. Automatic theorem proving in Walnut, 2016. arXiv preprint. doi:10.48550/arXiv.1603.06017.

29 Aline Parreau, Michel Rigo, Eric Rowland, and Élise Vandomme. A new approach to the 2-regularity of the ℓ-abelian complexity of 2-automatic sequences. *Electron. J. Comb.*, 22(1):research paper p1.27, 44, 2015. URL: www.combinatorics.org/ojs/index.php/eljc/article/view/v22i1p27.

30 Giuseppe Pirillo and Stefano Varricchio. On uniformly repetitive semigroups. *Semigroup Forum*, 49:125–129, 1994.

31 Michaël Rao. On some generalizations of abelian power avoidability. *Theoret. Comput. Sci.*, 601:39–46, 2015. doi:10.1016/j.tcs.2015.07.026.

32 Gwénaël Richomme, Kalle Saari, and Luca Q. Zamboni. Balance and abelian complexity of the Tribonacci word. *Adv. in Appl. Math.*, 45(2):212–231, 2010. doi:10.1016/j.aam.2010.01.006.

33 Gwénaël Richomme, Kalle Saari, and Luca Q. Zamboni. Abelian complexity of minimal subshifts. *J. Lond. Math. Soc. (2)*, 83(1):79–95, 2011. doi:10.1112/jlms/jdq063.

34 Michel Rigo and Arnaud Maes. More on generalized automatic sequences. *Journal of Automata, Languages, and Combinatorics*, 7(3):351–376, 2002. doi:10.25596/jalc-2002-351.

35 Michel Rigo, Manon Stipulanti, and Markus A. Whiteland. Automaticity and Parikh-collinear morphisms. In *Combinatorics on words*, volume 13899 of *Lecture Notes in Comput. Sci.*, pages 247–260. Springer, Cham, 2023. doi:10.1007/978-3-031-33180-0_19.

36 Michel Rigo, Manon Stipulanti, and Markus A. Whiteland. Automatic abelian complexities of Parikh-collinear fixed points, 2024. To be published in Theory Comput. Syst. doi:10.48550/arXiv.2405.18032.

37 Julian Sahasrabudhe. Sturmian words and constant additive complexity. *Integers*, 15:Paper No. A30, 8, 2015.

38 Jeffrey Shallit. A generalization of automatic sequences. *Theoret. Comput. Sci.*, 61(1):1–16, 1988. doi:10.1016/0304-3975(88)90103-X.

39 Jeffrey Shallit. Abelian complexity and synchronization. *Integers*, 21:Paper No. A36, 14, 2021.

40 Jeffrey Shallit. *The logical approach to automatic sequences—exploring combinatorics on words with* Walnut, volume 482 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 2023.

41 Jeffrey Shallit. Note on a Fibonacci parity sequence. *Cryptogr. Commun.*, 15(2):309–315, 2023. doi:10.1007/s12095-022-00592-5.

42 Neil J. A. Sloane and et al. The On-Line Encyclopedia of Integer Sequences. URL: https://oeis.org.

43 Ondřej Turek. Abelian complexity and abelian co-decomposition. *Theoret. Comput. Sci.*, 469:77–91, 2013. doi:10.1016/j.tcs.2012.10.034.

44 Ondřej Turek. Abelian complexity function of the Tribonacci word. *J. Integer Seq.*, 18(3):Article 15.3.4, 29, 2015.

Pseudo-Deterministic Construction of Irreducible Polynomials over Finite Fields

Shanthanu S. Rai
Tata Institute of Fundamental Research, Mumbai, India

Abstract

We present a polynomial-time pseudo-deterministic algorithm for constructing irreducible polynomial of degree d over finite field $\mathbb{F}_q$. A pseudo-deterministic algorithm is allowed to use randomness, but with high probability it must output a canonical irreducible polynomial. Our construction runs in time $\tilde{O}(d^4 \log^4 q)$.

Our construction extends Shoup's deterministic algorithm (FOCS 1988) for the same problem, which runs in time $\tilde{O}(d^4 p^{\frac{1}{2}} \log^4 q)$ (where p is the characteristic of the field $\mathbb{F}_q$). Shoup had shown a reduction from constructing irreducible polynomials to factoring polynomials over finite fields. We show that by using a fast randomized factoring algorithm, the above reduction yields an efficient pseudo-deterministic algorithm for constructing irreducible polynomials over finite fields.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory

Keywords and phrases Algebra and Computation, Finite fields, Factorization, Pseudo-deterministic, Polynomials

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.33

Related Version *arXiv Version*: `https://arxiv.org/abs/2410.04071`

Funding Research supported by the Department of Atomic Energy, Government of India, under project 12-R&D-TFR-5.01-0500.

Acknowledgements The author would like to thank Mrinal Kumar and Ramprasad Saptharishi for introducing him to the question of pseudo-deterministic construction of irreducible polynomials and for the many insightful discussions along the way.

1 Introduction

A polynomial $f(X)$ over a finite field $\mathbb{F}_q$ (q is a prime power) is said to be irreducible if it doesn't factor as $f(X) = g(X)h(X)$ for some non-trivial polynomials $g(X)$ and $h(X)$. Irreducible polynomials over finite fields are algebraic analogues of primes numbers over integers. It is natural to ask if one can construct an irreducible polynomial of degree d over $\mathbb{F}_q$ efficiently. Constructing these irreducible polynomials are important since they yield explicit construction of finite fields of non-prime order. Working over such non-prime finite fields is crucial in coding theory, cryptography, pseudo-randomness and derandomization. Any algorithm that constructs irreducible polynomials of degree d over $\mathbb{F}_q$ would output $d \log q$ bits, so we expect an efficient algorithm for constructing irreducible polynomials would run in time poly$(d, \log q)$.

About $\frac{1}{d}$ fraction of polynomials of degree d are irreducible over $\mathbb{F}_q$ [9, Ex. 3.26 and 3.27]. This gives a simple "trial and error" randomized algorithm for constructing irreducible polynomials, namely, pick a random degree d polynomial and check if it is irreducible. We can use Rabin's algorithm [10] for checking if a polynomial is irreducible, which can be implemented in $\tilde{O}(d \log^2 q)$ [7, Section 8.2][1]. In order to improve the probability of finding

[1] $\tilde{O}$ notation omits log factors in d and $\log q$.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 33; pp. 33:1–33:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an irreducible polynomial to $\frac{1}{2}$, we sample about d polynomials of degree d and check if any one of them is irreducible. Thus, the "trial and error" algorithm runs in time $\tilde{O}(d^2 \log^2 q)$. Couveignes and Lercier [5] give an alternative randomized algorithm that runs in time $\tilde{O}(d \log^5 q)$, which is optimal in the exponent of d. Their algorithm constructs irreducible polynomials by using isogenies between elliptic curves.

Motivated by this, it is natural to ask if there is also an efficient deterministic algorithm for constructing irreducible polynomials. In the 80s, some progress was made towards this problem. Adleman and Lenstra [1] gave an efficient deterministic algorithm for this problem conditional on the generalized Riemann hypotheses. They also gave an unconditional deterministic algorithm which outputs an irreducible polynomial of degree approximately d. Shoup [12] gives a deterministic algorithm of constructing degree d irreducible polynomial which runs in time $\tilde{O}(d^4 p^{\frac{1}{2}} \log^4 q)$ (where p is the characteristic of $\mathbb{F}_q$). So, Shoup's algorithm is efficient for fields of small characteristic ($p << d$). But when p is large (say super exponential in d), the algorithm does not run in polynomial time due to the $p^{\frac{1}{2}}$ factor in the run time. Since then, there hasn't been much progress towards this problem and in particular, the problem of efficient and unconditional deterministic construction of irreducible polynomials over $\mathbb{F}_q$ remains open! In fact, the special case of efficient and unconditional deterministic construction of quadratic non-residues in $\mathbb{F}_p$ is also open.

One can ask similar questions in the integer world, namely, "How to efficiently construct n-bit prime numbers?". By the Prime Number Theorem, there are about $\frac{1}{n}$ n-bit prime numbers (note the similarity between density of primes and density of irreducible polynomials over $\mathbb{F}_q$). Again this gives a simple randomized algorithm of just sampling a random n-bit number and checking if it's prime using AKS primality test [2]. But here too, there is no known efficient deterministic algorithm for constructing n-bit prime numbers [13].

Due to the difficulty in finding deterministic algorithms for these problems, we ask a slightly weaker but related question. Are there efficient pseudo-deterministic algorithms for these problems?

▶ **Definition 1.1.** *A pseudo-deterministic algorithm is a randomized algorithm which for a given input, generates a canonical output with probability at least $\frac{1}{2}$.*

Gat and Goldwasser [6] first introduced the notion of pseudo-deterministic algorithm (they had called it Bellagio algorithm). Pseudo-deterministic algorithm can be viewed as a middle ground between a randomized and a deterministic algorithm. From an outsider's perspective, a pseudo-deterministic algorithm seems like a deterministic algorithm in the sense that with high probability it outputs the same output for a given input. The breakthrough result of Chen et al. [4] gave a polynomial-time pseudo-deterministic algorithm for constructing n-bit prime numbers in the infinitely often regime.

▶ **Theorem 1.2.** *There is a randomized polynomial-time algorithm B such that, for infinitely many values of n, $B(1^n)$ outputs a canonical n-bit prime p_n with high probability.*

In particular, their algorithm doesn't give valid outputs for all values of the input n. Surprisingly, their algorithm is based on complexity theoretic ideas, and not number theoretic ideas. In fact, they show a more general result that if a set of strings Q are "dense" and it is "easy" to check if a string x is in Q, then there is an efficient pseudo-deterministic algorithm for generating elements of Q of a particular length in the infinitely often regime. Both prime numbers and irreducible polynomials over $\mathbb{F}_q$ satisfy this property. Thus, this gives an efficient pseudo-deterministic algorithm for constructing irreducible polynomials over $\mathbb{F}_q$ in the infinitely often regime.

But not only does this algorithm not work for all d, there are no good density bounds for the fraction of d where the algorithm gives valid output. So it is natural to ask if we can extend this result to all values of degree d over all finite fields $\mathbb{F}_q$. In this paper, we present a more direct pseudo-deterministic algorithm for constructing irreducible polynomials over $\mathbb{F}_q$ (for all degrees d) which crucially relies on the structure of irreducible polynomials. Our result extends Shoup's [12] deterministic algorithm for constructing irreducible polynomials. Shoup reduces the problem of constructing irreducible polynomials to factoring polynomials over $\mathbb{F}_q$. We observe that by making use of the fast randomized factoring algorithm, and the "canonization" process described by Gat and Goldwasser [6] for computing q-th residues over $\mathbb{F}_p$, the above reduction yields an efficient pseudo-deterministic algorithm for constructing irreducible polynomials over $\mathbb{F}_q$.

▶ **Theorem 1.3.** *There is a pseudo-deterministic algorithm for constructing an irreducible polynomial of degree d over $\mathbb{F}_q$ (q is prime power) in expected time $\tilde{O}(d^4 \log^4 q)$.*

2 Overview

As mentioned earlier, Shoup's deterministic algorithm [12] is efficient for fields of small characteristic. We extend Shoup's algorithm and make it efficient over all fields, but at the cost of making the algorithm pseudo-deterministic. In order to see the main ideas involved, let's consider a toy problem of constructing irreducible polynomial of degree 2 over $\mathbb{F}_p$ (p is prime). Suppose we could get our hands on some quadratic non residue α, then $X^2 - \alpha$ would be irreducible. There are $\frac{p-1}{2}$ quadratic non residues in $\mathbb{F}_p$, so if we randomly pick an $\alpha \in \mathbb{F}_p$ and output $X^2 - \alpha$, it would be irreducible with about $\frac{1}{2}$ probability. But this approach wouldn't be pseudo-deterministic, since in each run we will very likely choose different α.

In order to obtain a canonical quadratic non residue α, we first set $\alpha = -1$ and repeatedly perform $\alpha \leftarrow \sqrt{\alpha}$ (choosing the smallest square root) until α is a quadratic non residue. Here, β is a square root of α if $\beta^2 = \alpha \pmod{p}$. For computing the square root, we can use Cantor-Zassenhaus randomized factoring algorithm [3]. In Example 2.1, we illustrate the above strategy over a specific finite field. Algorithm 1 implements this strategy.

▶ **Example 2.1.** Let's try to pseudo-deterministically construct a quadratic non-residue in $\mathbb{F}_{73}$. We first set $\alpha = -1$. Square roots of $-1 \pmod{73}$ are 27 and 46. The square roots are computed using Cantor-Zassenhaus randomized factoring algorithm [3].

We choose the smallest square root 27 and set $\alpha = 27$. Square roots of $27 \pmod{73}$ are 10 and 63. We choose the smallest square root 10 and set $\alpha = 10$. Since 10 is a quadratic non-residue, we output 10 (we use Euler's criterion[2] to check if 10 is a quadratic non-residue).

Algorithm 1 Pseudo-deterministically constructing irreducible polynomial of degree 2 over $\mathbb{F}_p$.

```
$\alpha \leftarrow -1$
while $\alpha$ is a quadratic residue do
    Factorize $X^2 - \alpha = (X - \beta_1)(X - \beta_2)$
    $\alpha \leftarrow \min(\beta_1, \beta_2)$
end while
Output $X^2 - \alpha$
```

[2] Euler's criterion: For odd prime p, a is a quadratic non residue iff $a^{(p-1)/2} = -1$

Suppose $p - 1 = 2^k l$ (where l is odd). Each time we take square root, the order[3] of $\alpha \pmod{p}$ doubles. Since the order of α divides $|\mathbb{F}_p^*| = 2^k l$ (by Lagrange's theorem), we can repeatedly take square roots in Algorithm 1 at most k times. Thus, Algorithm 1 will terminate with at most $\log p$ iterations of the while loop. This algorithm is based on Gat and Goldwasser's algorithm [6] for computing q-th residues over $\mathbb{F}_p$. The algorithm is pseudo-deterministic since at each iteration of the while loop, we "canonize" our choice of square root by picking the smallest one among the two choices. Note that we used Euler's criterion for checking if α is a quadratic residue or not in Line 2.

We can generalize the above ideas for constructing irreducible polynomials over finite fields. Shoup [12] showed that constructing irreducible polynomials over $\mathbb{F}_p$ reduces to finding q-th non residues over appropriate field extensions (q is prime). These q-th non residues can be pseudo-deterministically constructed using similar techniques as in Algorithm 1.

The rest of the paper is organized as follows. We start with some preliminaries in Section 3. In Section 4, we will reduce the problem of constructing irreducible polynomials over extensions fields $\mathbb{F}_{p^k}$ to constructing them over $\mathbb{F}_p$. Section 5 will make use of Shoup's observation mentioned in previous paragraph to construct irreducible polynomials over $\mathbb{F}_p$. Finally, in Section 6 we conclude with some open problems.

3 Preliminaries

3.1 Pseudo-deterministic algorithms

We defined pseudo-deterministic algorithm to be randomized algorithm which for a given input, generates a canonical output with probability at least $\frac{1}{2}$. In this paper, whenever the pseudo-deterministic algorithm doesn't generate a canonical output, it just fails and doesn't give any valid output. In such cases, we can just rerun the algorithm until we get some valid output (which is bound to be canonical). Now the runtime of the algorithm will be random, but the expected runtime will be (asymptotically) same as the original runtime.

For all the pseudo-deterministic algorithms in this paper, we report the expected run time in the above sense. These algorithms always generate a canonical output, but the amount of time they take to do so is random.

3.2 Finite Field primer

In this subsection, we go over some basic facts about finite fields that will be useful in later sections.

3.2.1 Splitting field

A polynomial $h(X) \in \mathbb{K}[X]$ may not factorize fully into linear factors over the field $\mathbb{K}$. Suppose $\mathbb{F}$ is the smallest extension of $\mathbb{K}$ such that $h(X)$ fully factorizes into linear factors over $\mathbb{F}$. In other words, there exists $\alpha_1, \alpha_2, \ldots \alpha_k \in \mathbb{F}$ such that,

$$h(X) = (X - \alpha_1)(X - \alpha_2) \cdots (X - \alpha_k)$$

Then $\mathbb{F}$ is the called the splitting field of $h(X)$ over $\mathbb{K}$ [9, Definition 1.90]. Note that for any other extension of $\mathbb{K}$ that is a proper subfield of $\mathbb{F}$, $h(X)$ will not fully factorize into linear factors.

[3] Order of α is the least integer $k > 0$ such that $\alpha^k = 1$ in $\mathbb{F}_p$

3.2.2 Structure of Finite Fields

For every prime power p^n (p is prime), there exists a finite field of size p^n and all finite fields of size p^n are isomorphic to each other [9, Theorem 2.5].

▶ **Theorem 3.1** (Existence and Uniqueness of Finite Fields). *For every prime p and every positive integer n there exists a finite field with p^n elements. Any finite field with $q = p^n$ elements is isomorphic to the splitting field of $X^q - X$ over $\mathbb{F}_p$.*

Thus, elements of $\mathbb{F}_{p^n}$ are roots of $X^{p^n} - X$. From this, we get the following generalization of Fermat's little theorem for finite fields:

▶ **Theorem 3.2** (Fermat's little theorm for finite fields). *If $\alpha \in \mathbb{F}_{p^n}$, then $\alpha^{p^n} = \alpha$. Conversely, if α is in some finite field and $\alpha^{p^n} = \alpha$, then $\alpha \in \mathbb{F}_{p^n}$.*

The below theorem gives the necessary and sufficient condition for a finite field $\mathbb{F}_{p^m}$ to be a subfield of another finite field $\mathbb{F}_{p^n}$ [9, Theorem 2.6].

▶ **Theorem 3.3** (Subfield Criterion). *Let $\mathbb{F}_q$ be a finite field with $q = p^n$ elements. Then every subfield of $\mathbb{F}_q$ has order p^m, where m is the positive divisor of n. Conversely, if m is the positive divisor of n, then there is exactly one subfield of $\mathbb{F}_q$ with p^m elements.*

From Theorem 3.2 and Theorem 3.3, we get the following useful lemma:

▶ **Lemma 3.4.** *Suppose α is some finite field element. Let k be the smallest integer greater than 0 such that $\alpha^{p^k} = \alpha$. Then, $\mathbb{F}_{p^k}$ is the smallest extension of $\mathbb{F}_p$ that contains α. In other words, $\alpha \in \mathbb{F}_{p^k}$ and for all $1 \le k' < k$, $\alpha \notin \mathbb{F}_{p^{k'}}$.*

3.2.3 Conjugates and Minimal polynomial

Let $f(X)$ be an irreducible polynomial of degree n over $\mathbb{F}_q$ (q is prime power). Then, $f(X)$ has some root $\alpha \in \mathbb{F}_{q^n}$. Also, the elements $\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{n-1}}$ are all distinct and are the roots of $f(X)$ [9, Theorem 2.14].

$$f(X) = (X - \alpha)(X - \alpha^q)(X - \alpha^{q^2}) \cdots (X - \alpha^{q^{n-1}})$$

The splitting field of $f(X)$ with respect to $\mathbb{F}_q$ is $\mathbb{F}_{q^n}$ [9, Corollary 2.15]. The minimal polynomial of α over $\mathbb{F}_q$ is $f(X)$.

Above, the roots of $f(X)$ are all of the form α^{q^i}. We will call such elements conjugates α with respect to $\mathbb{F}_q$:

▶ **Definition 3.5.** *Let $\mathbb{F}_{q^n}$ be an extension of $\mathbb{F}_q$ and let $\beta \in \mathbb{F}_{q^n}$. Then, $\beta, \beta^q, \beta^{q^2}, \ldots, \beta^{q^{n-1}}$ are called the conjugates of β with respect to $\mathbb{F}_q$.*

In the following sections, we will be using the below lemma to show that certain polynomials are irreducible.

▶ **Lemma 3.6** (Minimal polynomial of $\beta \in \mathbb{F}_{q^n}$). *Suppose $\beta \in \mathbb{F}_{q^n}$ and conjugates of β with respect to $\mathbb{F}_q$ are all distinct. Then the minimal polynomial of β over $\mathbb{F}_q$ has degree n and is of the form:*

$$g(X) = (X - \beta)(X - \beta^q)(X - \beta^{q^2}) \cdots (X - \beta^{q^{n-1}})$$

Thus, $g(X) \in \mathbb{F}_q[X]$ is an irreducible polynomial.

Proof. The minimal polynomial $g(X)$ of β over $\mathbb{F}_q$ is the smallest degree polynomial in $\mathbb{F}_q[X]$ such that $g(\beta) = 0$. Since, $g(\beta^{q^i}) = g(\beta)^{q^i} = 0$, all conjugates of β are roots of $g(X)$. Hence, degree of $g(X)$ is at least n (since the conjugates are all distinct). Also since $\beta \in \mathbb{F}_{q^n}$, degree of $g(X)$ is at most n. Thus, the degree of $g(X)$ is n.

Thus, $g(X) = (X - \beta)(X - \beta^q)(X - \beta^{q^2}) \cdots (X - \beta^{q^{n-1}})$. Since $g(X)$ is a minimal polynomial of β over $\mathbb{F}_q$, it will be in $\mathbb{F}_q[X]$ and is irreducible. ◀

3.2.4 Representing finite field elements

Throughout the paper, we assume that extension fields $\mathbb{F}_{p^k}$ are given to us as $\mathbb{F}_p[X]/(f(X))$, where $f(X)$ is an irreducible polynomial of degree k over $\mathbb{F}_p$ (refer [8] for working with other representations). Each element in $\mathbb{F}_p[X]/(f(X))$ can be viewed as a polynomial with degree at most k over $\mathbb{F}_p$. The coefficient vectors of these polynomials are in $\mathbb{F}_p^k$. This gives a natural isomorphism $\Phi : \mathbb{F}_{p^k} \to \mathbb{F}_p^k$. In $\mathbb{F}_p^k$, we can order elements in lexicographic order in the natural sense.

▶ **Definition 3.7.** *We say that $\alpha \in \mathbb{F}_{p^k}$ is lexicographically smaller than $\beta \in \mathbb{F}_{p^k}$, if $\Phi(\alpha) \in \mathbb{F}_p^k$ is lexicographically smaller than $\Phi(\beta) \in \mathbb{F}_p^k$.*

In the above definition, we compare the coordinates of $\Phi(\alpha)$ and $\Phi(\beta)$ by fixing some ordering on elements on $\mathbb{F}_p$ (for e.g., we can consider the natural ordering one gets from the additive group structure of $\mathbb{F}_p$). Checking if $\Phi(\alpha)$ is lexicographically smaller than $\Phi(\beta)$ requires k comparisons, with each comparison taking $O(\log p)$ time. Thus, overall it takes $O(k \log p)$ time to check if $\Phi(\alpha)$ is lexicographically smaller than $\Phi(\beta)$.

Similarly, we can define a lexicographic ordering on polynomials over $\mathbb{F}_{p^k}$.

▶ **Definition 3.8.** *Suppose we are given two polynomials $g(X)$ and $h(X)$ of degree d over $\mathbb{F}_{p^k}$. Then we say that $g(X)$ is lexicographically smaller than $h(X)$ if the coefficient vector of $g(X)$ is lexicographically smaller than coefficient vector of $h(X)$ (the coefficients are compared using Φ).*

Checking if $g(X)$ is lexicographically smaller than $h(X)$ requires $d + 1$ comparisons, with each comparison taking $O(k \log p)$ time. Thus, overall it takes $O(dk \log p)$ time to check if $g(X)$ is lexicographically smaller than $h(X)$.

▶ **Lemma 3.9** (Picking lexicographically smallest polynomial). *Suppose we are given n polynomials $f_1(X), f_2(X), \ldots, f_n(X)$ of degree d over $\mathbb{F}_{p^k}$. Then, there is an algorithm that outputs the lexicographically smallest polynomial among them in $O(ndk \log p)$ time.*

Proof. We go over each polynomial $f_i(X)$ one by one, checking if $f_i(X)$ is lexicographically smaller than the lexicographically smallest polynomial we have seen so far. Since each comparison takes $O(dk \log p)$ time, and we do at most n comparisons, the algorithm runs in $O(ndk \log p)$ time. ◀

3.3 Equal degree polynomial factorization

Shoup [12] reduced constructing irreducible polynomials to factoring polynomials over finite field. It turns out that the reduction factors polynomials whose irreducible factors all have same degree. Hence, equal degree factorization is a crucial sub-routine for constructing irreducible polynomials. There are several fast randomized equal degree factorization algorithms, and below we mention one of them:

▶ **Theorem 3.10** (Equal degree factorization). *Suppose $f(X)$ is a polynomial of degree d over $\mathbb{F}_q$ (q is prime power) which factors into irreducible polynomials of equal degree. Then, the equal degree factorization algorithm by von zur Gathen & Shoup [15] factors $f(X)$ in expected time $\tilde{O}(d\log^2 q)$.*

4 Construction of irreducible polynomials over extension fields $\mathbb{F}_{p^k}$

We first show in Algorithm 2 that constructing irreducible polynomials over extension fields $\mathbb{F}_{p^k}$ can be reduced to constructing irreducible polynomials over $\mathbb{F}_p$ (p is prime). Theorem 4.1 shows the correctness and running time of Algorithm 2.

Algorithm 2 Pseudo-deterministic construction of irreducible polynomials over $\mathbb{F}_{p^k}$.

```
Input: Degree $d$
Output: Irreducible polynomial of degree $d$ over $\mathbb{F}_{p^k}$
Pseudo-deterministically construct irreducible polynomial $f(X)$ over $\mathbb{F}_p$ of degree $dk$.
Factor $f(X) = \prod_{i=0}^{k-1} f_i(X)$ over $\mathbb{F}_{p^k}$ using Theorem 3.10.
Output the lexicographically smallest factor $f_i(X)$.
```

▶ **Theorem 4.1** (Correctness and Running time of Algorithm 2). *Suppose there is a pseudo-deterministic algorithm for constructing irreducible polynomials of degree l over $\mathbb{F}_p$ (p prime), that runs in expected time $T(l,p)$. Then Algorithm 2 pseudo-deterministically constructs irreducible polynomials of degree d over extension field $\mathbb{F}_{p^k}$ in expected time $T(dk,p) + \tilde{O}(dk^3\log p)$.*

Proof. Algorithm 2 first constructs an irreducible polynomial $f(X)$ of degree dk over F_p. Note that $\mathbb{F}_p[X]/(f(X))$ is isomorphic to $\mathbb{F}_{p^{dk}}$. Some $\alpha \in \mathbb{F}_{p^{dk}}$ will be a root of $f(X)$. The conjugates of α with respect to $\mathbb{F}_p$ are all distinct and are the roots of $f(X)$ (refer Section 3.2.3):

$$f(X) = (X-\alpha)(X-\alpha^p)(X-\alpha^{p^2})\cdots(X-\alpha^{p^{dk-2}})(X-\alpha^{p^{dk-1}})$$

Rearranging the above terms, we get:

$$\begin{aligned}
f(X) = &\Big[(X-\alpha)(X-\alpha^{p^k})(X-\alpha^{p^{2k}})\cdots(X-\alpha^{p^{(d-1)k}})\Big]\\
&\Big[(X-\alpha^p)(X-\alpha^{p^{k+1}})(X-\alpha^{p^{2k+1}})\cdots(X-\alpha^{p^{(d-1)k+1}})\Big]\\
&\Big[(X-\alpha^{p^2})(X-\alpha^{p^{k+2}})(X-\alpha^{p^{2k+2}})\cdots(X-\alpha^{p^{(d-1)k+2}})\Big]\\
&\vdots\\
&\Big[(X-\alpha^{p^{(k-1)}})(X-\alpha^{p^{k+(k-1)}})(X-\alpha^{p^{2k+(k-1)}})\cdots(X-\alpha^{p^{(d-1)k+(k-1)}})\Big]\\
= &\prod_{i=0}^{k-1}\prod_{j=0}^{d-1}(X-\alpha^{p^{jk+i}})\\
:= &\prod_{i=0}^{k-1} f_i(X)
\end{aligned}$$

Let $q = p^k$. $f_i(X)$ has degree d and its roots are conjugates of $\alpha^{p^i} \in \mathbb{F}_{q^d}$ with respect to $\mathbb{F}_q$ (which are all distinct). Thus, from Lemma 3.6, $f_i(X) \in \mathbb{F}_q[X]$ is the minimal polynomial of α^{p^i} over $\mathbb{F}_q$, and hence $f_i(X)$ is irreducible over $\mathbb{F}_q$. So, we can use Theorem 3.10 to factorize $f(X)$ over $\mathbb{F}_q$, obtaining all factors $f_i(X)$ of degree d. We then use Lemma 3.9 to output the lexicographically smallest factor among $f_i(X)$. Let the lexicographically smallest factor be denoted by $f_{i^*}(X)$. Given a polynomial $f(X)$ of degree dk, $f_{i^*}(X)$ is canonical. Thus, the above construction is pseudo-deterministic.

For the running time, it takes $T(dk, p)$ time to construct $f(X)$, and then $\tilde{O}(dk^3 \log p)$ time to factor $f(X)$ over field $\mathbb{F}_{p^k}$ (from Theorem 3.10). Finally, choosing $f_{i^*}(X)$ among $f_i(X)$ can be computed in time $O(dk^2 \log p)$ (from Lemma 3.9). Thus, the overall running time of the algorithm is $T(dk, p) + \tilde{O}(dk^3 \log p)$. ◀

5 Construction of irreducible polynomials over $\mathbb{F}_p$

Shoup's algorithm reduces constructing irreducible polynomials over $\mathbb{F}_p$ to finding q-th non residues in splitting field of $X^q - 1$, for all prime divisors q of d (and $q \neq p$). For completeness, we reproduce the theorem below and refer to Theorem 2.1 in [12] for it's proof.

▶ **Theorem 5.1** (Reduction to finding q-th non residues). *Assume that for each prime $q \mid d$, $q \neq p$, we are given a splitting field $\mathbb{K}$ of $X^q - 1$ over $\mathbb{F}_p$ and a q-th non residue in $\mathbb{K}$. Then we can find an irreducible polynomial over $\mathbb{F}_p$ of degree d deterministically with $\tilde{O}(d^4 \log p + \log^2 p)$ operations in $\mathbb{F}_p$.*

Shoup constructs the splitting field[4] $\mathbb{K}$ of $X^q - 1$ over $\mathbb{F}_p$ and a q-th non residue in $\mathbb{K}$ by reducing to deterministic polynomial factorization. Since no known efficient deterministic factoring algorithms are known, his algorithm is not efficient for finite fields of large characteristic. In this section, we will find a canonical splitting field $\mathbb{K}$ and a canonical q-th non residues by using a fast randomized factoring algorithm. Thus, we obtain an efficient algorithm for constructing irreducible polynomials over $\mathbb{F}_p$, but at the cost of making the algorithm pseudo-deterministic.

For each prime $q \mid d, q \neq p$, we will pseudo-deterministically construct a splitting field $\mathbb{K}$ of $X^q - 1$ over $\mathbb{F}_p$ and find a q-th non residue in $\mathbb{K}$. To this end, we first analyze the factorization of $X^q - 1 \in \mathbb{F}_p[X]$.

▶ **Lemma 5.2.** *Consider the polynomial $X^q - 1 \in \mathbb{F}_p[X]$ (p, q are prime numbers). Let k be the smallest integer greater than 0 such that $q \mid p^k - 1$ (in other words, k is the order of p $\pmod q$). Then,*

1. *The splitting field of $X^q - 1$ over $\mathbb{F}_p$ is $\mathbb{F}_{p^k}$*
2. *$X^q - 1 = (X-1)g_1(X)g_2(X)\cdots g_{\frac{q-1}{k}}(X)$ where $g_i(X) \in \mathbb{F}_p[X]$ are irreducible polynomials of degree k.*

Proof. Let $\mathbb{K}$ be the splitting field of $X^q - 1$ over $\mathbb{F}_p$. The roots of $X^q - 1$ in $\mathbb{K}$ are by definition the q-th roots of unity. Suppose $\omega \in \mathbb{K}$ is some primitive q-th root of unity. Then, $\{1, \omega, \omega^2, \ldots, \omega^{q-1}\}$ are all the q-th roots of unity, and they form a multiplicative subgroup in $\mathbb{K}^*$. In fact, since q is prime, each of $\{\omega, \omega^2, \ldots, \omega^{q-1}\}$ is a primitive q-th root of unity.

Since $\omega^q = 1$, we have $\omega^{p^k} = \omega$ and hence from Theorem 3.2, $\omega \in \mathbb{F}_{p^k}$. By definition of k, k is the smallest integer greater than 0 such that $\omega^{p^k} = \omega$. So from Lemma 3.4, $\mathbb{F}_{p^k}$ is the smallest extension of $\mathbb{F}_p$ that contains ω. $X^q - 1$ splits linearly as:

$$X^q - 1 = (X-1)(X-\omega)(X-\omega^2)\cdots(X-\omega^{q-1})$$

[4] Shoup constructs an irreducible polynomial $g(X) \in \mathbb{F}_p[X]$ such that $\mathbb{F}_p/(g(X))$ is isomorphic to splitting field of $X^q - 1$ over $\mathbb{F}_p$

$\mathbb{F}_{p^k}$ is the smallest extension of $\mathbb{F}_p$ that contains all the roots of $X^q - 1$. Thus, $\mathbb{F}_{p^k}$ is the splitting field of $X^q - 1$. We next consider the factorization pattern of $X^q - 1$ over $\mathbb{F}_p$.

Let G be the multiplicative group of integers modulo q. Since q is prime, elements of G are $\{1, 2, \ldots, q-1\}$. Consider the cyclic subgroup H of G generated by p. The elements of H are $\{1, p, p^2, \ldots, p^{k-1}\}$. The cosets of H partition G. Let a_1H, a_2H, $\ldots$, $a_{(q-1)/k}H$ be the $(q-1)/k$ cosets of H that partition G. Then, $X^q - 1 \in \mathbb{F}_p[X]$ can be factorized as follows:

$$\begin{aligned} X^q - 1 &= (X-1)(X-\omega)(X-\omega^2)\cdots(X-\omega^{q-1}) \\ &= (X-1)\prod_{i=1}^{(q-1)/k}\prod_{j\in a_iH}(X-\omega^j) \\ &= (X-1)\prod_{i=1}^{(q-1)/k}(X-\omega^{a_i})(X-\omega^{a_ip})(X-\omega^{a_ip^2})\cdots(X-\omega^{a_ip^{k-1}}) \\ &:= (X-1)\prod_{i=1}^{(q-1)/k} g_i(X) \end{aligned}$$

$g_i(X)$ has degree k and its roots are conjugates of $\omega^{a_i} \in \mathbb{F}_{p^k}$ with respect to $\mathbb{F}_p$ (which are all distinct). Thus, from Lemma 3.6, $g_i(X) \in \mathbb{F}_p[X]$ is the minimal polynomial of ω^{a_i} over $\mathbb{F}_p$, and hence is irreducible over $\mathbb{F}_p$. ◀

Thus, the splitting field of $X^q - 1$ over $\mathbb{F}_p$ is $\mathbb{F}_{p^k}$. It is easy to see that $\mathbb{F}_{p^k}$ contains a q-th non residue, since the map $\Phi : \alpha \mapsto \alpha^q$ is not surjective in $\mathbb{F}_{p^k}$ (since for every i, $\Phi(\omega^i) = 1$, where ω is some primitive q-th root of unity).

In order to get our hands on a canonical representation of $\mathbb{F}_{p^k}$, we can factorize $(X^q - 1)/(X-1) = X^{q-1} + X^{q-2} + \cdots + X + 1$ over $\mathbb{F}_p$ and pick the lexicographically smallest degree k irreducible factor $h(X)$. Then, $\mathbb{F}_p[X]/h(X)$ is isomorphic to $\mathbb{F}_{p^k}$. Let ω be an element in $\mathbb{F}_{p^k}$ isomorphic to $X \in \mathbb{F}_p[X]/h(X)$. Next, to find a canonical q-th non residue $\alpha \in \mathbb{F}_{p^k}$, we set $\alpha = \omega$ and repeatedly perform $\alpha \leftarrow \sqrt[q]{\alpha}$ (choosing the lexicographically smallest q-th root) until α is a q-th non residue. Algorithm 3 implements the above idea and constructs an irreducible polynomial of degree d. In Line 3 and Line 8, factorization is done using Theorem 3.10. We analyze the correctness and running time of Algorithm 3 in Theorem 5.3.

▶ **Theorem 5.3** (Correctness and Runtime of Algorithm 3). *Algorithm 3 pseudo-deterministically constructs an irreducible polynomial of degree d over $\mathbb{F}_p$ and runs in expected time $\tilde{O}(d^4 \log^3 p)$.*

Proof. We need to show that the for loop in Algorithm 3 correctly computes the splitting field of $X^q - 1$ and finds a q-th non residue in the splitting field. Then, Line 13 will correctly output an irreducible polynomial of degree d over $\mathbb{F}_p$ (from Theorem 5.1).

Let k be the smallest integer greater than 0 such that $q \mid p^k - 1$ (k is the order of $p \pmod q$). From Lemma 5.2, $X^{q-1} + X^{q-2} + \cdots + X + 1$ factorizes as $g_1(X)g_2(X)\cdots g_{\frac{q-1}{k}}(X)$ where $g_i(X)$ are degree k irreducible polynomials. Thus, by choosing the lexicographically smallest degree k irreducible factor $h(X)$ of $X^q - 1$, we ensure that the choice of $h(X)$ is canonical. $\mathbb{F}_{p^k} \cong \mathbb{F}_p[X]/h(X)$ is the splitting field of $X^q - 1$ which contains a q-th non residue.

Let $\omega \in \mathbb{F}_{p^k}$ be some primitive q-th root of unity. Suppose α is a q-th residue, and let $\beta \in \mathbb{F}_{p^k}$ such that $\alpha = \beta^q$ (β is a q-th root of α). Then, $\{\beta, \beta\omega, \beta\omega^2, \ldots, \beta\omega^{q-1}\}$ are all q-th roots of α. Thus, as required in Line 8, $X^q - \alpha$ will factorize into linear factors. By ensuring

Algorithm 3 Pseudo-deterministic construction of irreducible polynomials over $\mathbb{F}_p$.

```
    Input: Degree $d$
    Output: Irreducible polynomial of degree $d$ over $\mathbb{F}_p$
 1: Initialize arrays $H \leftarrow [\,], \Lambda \leftarrow [\,]$
 2: for prime $q \mid d, q \neq p$ do
 3:     Factorize $X^{q-1} + X^{q-2} + \cdots + X + 1 = g_1(X)g_2(X)\cdots g_{\frac{q-1}{k}}(X)$ over $\mathbb{F}_p$
 4:     $h(X) \leftarrow$ lexicographically smallest degree $k$ factor among $g_1(X), g_2(X), \ldots, g_{\frac{q-1}{k}}(X)$

 5:     Field arithmetic over $\mathbb{F}_{p^k}$ will henceforth be performed over $\mathbb{F}_p[X]/h(X)$.
 6:     $\alpha \leftarrow$ element in $\mathbb{F}_{p^k}$ isomorphic to $X$ in $\mathbb{F}_p[X]/h(X)$
 7:     while $\alpha$ is a $q$-th residue do
 8:         Factorize $X^q - \alpha = (X - \beta_1)(X - \beta_2)\cdots(X - \beta_q)$ over $\mathbb{F}_{p^k}$
 9:         $\alpha \leftarrow$ lexicographically smallest element among $\beta_1, \beta_2, \ldots, \beta_q$ in $\mathbb{F}_{p^k}$
10:     end while
11:     Append $h(X)$ to array $H$ and $\alpha$ to array $\Lambda$
12: end for
13: Using arrays $H$ and $\Lambda$ and Theorem 5.1, deterministically construct an irreducible
    polynomial of degree $d$ over $\mathbb{F}_p$.
```

that we pick the lexicographically smallest q-th root of α, we "canonize" the computation of q-th non residue. This "canonization" process is akin to the one Gat and Goldwasser [6, Section 5] used to compute q-th non residue in $\mathbb{F}_p$.

But we still need to ensure that the while loop eventually terminates. Let $p^k - 1 = q^\ell r$, where r is not divisible by q. Note that $\ell \leq k \log p$. In each iteration of the while loop, the order of α in $\mathbb{F}_{p^k}^*$ increases by a factor of q. Since the order of α divides $\left|\mathbb{F}_{p^k}^*\right| = q^\ell r$ (by Lagrange's theorem), the while loop will terminate in at most ℓ steps. Thus, for each prime $q \mid d, q \neq p$, the for loop at Line 2 pseudo-deterministically constructs the splitting field $\mathbb{F}_{p^k}$ of $X^q - 1$ and a q-th non residue in $\mathbb{F}_{p^k}$.

Now we analyze the runtime. From Theorem 3.10, equal degree factorization in Line 3 takes $\tilde{O}(q \log^2 p)$. From Lemma 3.9, lexicographically smallest $h(X)$ in Line 4 can be chosen in $O(q \log p)$. The while loop at Line 7 runs at most ℓ times. The factoring step at Line 8 takes $\tilde{O}(qk^2 \log^2 p)$ (using Theorem 3.10) and the lexicographically smallest q-th root can be picked in $O(qk \log p)$ time. Thus, the while loop takes $\tilde{O}(\ell q k^2 \log^2 p)$. Since $\ell \leq k \log p$ and $k < q$, the running time of the while loop can be upper bounded by $\tilde{O}(q^4 \log^3 p)$. Thus the overall running time of each iteration of the for loop is $\tilde{O}(q^4 \log^3 p)$. So we can upper bound the running time of the entire for loop by $\tilde{O}(d^4 \log^3 p)$. Since the running time of Line 13 is also upper bounded by $\tilde{O}(d^4 \log^3 p)$ (from Theorem 5.1), the overall running time of the algorithm is $\tilde{O}(d^4 \log^3 p)$. ◀

We end this section by completing the proof of Theorem 1.3.

Proof of Theorem 1.3. Algorithm 2 pseudo-deterministically constructs irreducible polynomial of degree d over $\mathbb{F}_{p^k}$. From Theorem 4.1, it takes time $T(dk, p) + \tilde{O}(dk^3 \log p)$, where $T(dk, p)$ is time taken for the sub-routine which constructs degree dk irreducible polynomial over $\mathbb{F}_p$. We use Algorithm 3 to implement this sub-routine, which from Theorem 5.3 takes time $\tilde{O}(d^4 k^4 \log^3 p)$. Thus, the overall running time is $\tilde{O}(d^4 k^4 \log^3 p)$. Let $q = p^k$. Thus, we have given a pseudo-deterministic algorithm for constructing irreducible polynomials of degree d over $\mathbb{F}_q$ in expected time $\tilde{O}(d^4 \log^4 q)$. ◀

6 Conclusion

We have shown an efficient pseudo-deterministic algorithm for constructing irreducible polynomials of degree d over finite field $\mathbb{F}_q$. It is natural to ask if this algorithm can be derandomized to get a fully deterministic algorithm. Since our approach heavily relies on fast randomized polynomial factoring algorithms, and no efficient deterministic factoring algorithms are known, it is unclear how to derandomize it using the above approach. In fact, we don't even know how to deterministically construct a quadratic non residue modulo p (p is prime).

Another interesting question is to compare the hardness of deterministically factoring polynomials and deterministically constructing irreducible polynomials over finite fields. As mentioned earlier, Shoup [12] had showed that constructing irreducible polynomials over finite fields can be efficiently (and deterministically) reduced to factoring polynomials. This suggests that factoring polynomials is as hard as constructing irreducible polynomials. But what about the other direction? Would we be able to factor polynomials efficiently if we could construct irreducible polynomials?

The answer is affirmative in the quadratic case. Suppose we are given a quadratic non residue β modulo p. Then we can compute the square roots of any quadratic residue α module $\mathbb{F}_p$. In other words, given an irreducible polynomial $X^2 - \beta$, we can factorize $X^2 - \alpha$. This can be achieved using the Tonelli-Shanks [11, 14] algorithm for computing square roots modulo p. However, this technique does not easily generalize to higher degrees d, so there isn't enough evidence to confirm that constructing irreducible polynomials is as hard as factoring polynomials in general. We believe this is an interesting open question that can shine more light on the complexity of both these problems.

Gat and Goldwasser [6] highlighted the open problem of pseudo-deterministically constructing n-bit prime numbers, which still remains unsolved. Chen et al. [4] solved this problem but with the caveat that their algorithm works in the infinitely often regime. Their algorithm is based on complexity theoretic ideas. In this paper, we gave a pseudo-deterministic algorithm for constructing irreducible polynomials, which leverages the structure of irreducible polynomials. Perhaps similarly one could hope to get an efficient pseudo-deterministic algorithm for constructing primes using some number theoretic approaches.

References

1 Leonard M. Adleman and Hendrik W. Lenstra Jr. Finding irreducible polynomials over finite fields. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, STOC '86, pages 350–355, New York, NY, USA, 1986. ACM. doi:10.1145/12130.12166.

2 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. URL: http://www.jstor.org/stable/3597229.

3 David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981. URL: http://www.jstor.org/stable/2007663.

4 Lijie Chen, Zhenjian Lu, Igor C. Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-time pseudodeterministic construction of primes. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1261–1270, Los Alamitos, CA, USA, November 2023. IEEE. doi:10.1109/FOCS57990.2023.00074.

5 Jean-Marc Couveignes and Reynald Lercier. Fast construction of irreducible polynomials over finite fields. *Israel Journal of Mathematics*, 194(1):77–105, March 2013. doi:10.1007/s11856-012-0070-8.

6 Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, TR11-136(TR11-136), October 2011. URL: `https://eccc.weizmann.ac.il/report/2011/136`.

7 Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011. `doi:10.1137/08073408X`.

8 H. W. Lenstra. Finding isomorphisms between finite fields. *Mathematics of Computation*, 56(193):329–347, 1991. `doi:10.1090/S0025-5718-1991-1052099-2`.

9 Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, Cambridge, 2 edition, 1994. `doi:10.1017/CBO9781139172769`.

10 Michael O. Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput.*, 9(2):273–280, 1980. `doi:10.1137/0209024`.

11 Daniel Shanks. Five number-theoretic algorithms. In *Proceedings of the Second Manitoba Conference on Numerical Mathematics (Univ. Manitoba, Winnipeg, Man., 1972)*, volume No. VII of *Congress. Numer.*, pages 51–70. Utilitas Math., Winnipeg, MB, 1973.

12 Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 283–290. IEEE Computer Society, 1988. `doi:10.1109/SFCS.1988.21944`.

13 Terence Tao, Ernest Croot III, and Harald Helfgott. Deterministic methods to find primes. *Math. Comput.*, 81(278):1233–1246, 2012. `doi:10.1090/S0025-5718-2011-02542-1`.

14 Alberto Tonelli. Bemerkung über die auflösung quadratischer congruenzen. *Nachrichten von der Königl. Gesellschaft der Wissenschaften und der Georg-Augusts-Universität zu Göttingen*, 1891:344–346, 1891. URL: `http://eudml.org/doc/180329`.

15 Joachim von zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials. *Comput. Complex.*, 2(3):187–224, September 1992. `doi:10.1007/BF01272074`.

A Quadratic Upper Bound on the Reset Thresholds of Synchronizing Automata Containing a Transitive Permutation Group

Yinfeng Zhu

Institute of Natural Sciences and Mathematics, Ural Federal University, Ekaterinburg, Russia

Abstract

For any synchronizing n-state deterministic automaton, Černý conjectures the existence of a synchronizing word of length at most $(n-1)^2$. We prove that there exists a synchronizing word of length at most $2n^2 - 7n + 7$ for every synchronizing n-state deterministic automaton that satisfies the following two properties: 1. The image of the action of each letter contains at least $n-1$ states; 2. The actions of bijective letters generate a transitive permutation group on the state set.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Černý conjecture, deterministic finite automaton, permutation group, reset threshold, synchronizing automaton

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.34

Related Version *Previous Version*: https://arxiv.org/abs/2407.08135

Acknowledgements I thank Prof. Mikhail V. Volkov for valuable discussions, feedback and research suggestions.

1 Introduction

1.1 Synchronizing automata and Černý Conjecture

Let Q be a set. Denote the set of all mappings from Q to itself by $\mathsf{T}(Q)$. For the purposes of this article, an *automaton* $\mathcal{A}$ is a triple (Q, Σ, δ) where Q and Σ are two finite sets, δ is a mapping from Σ to $\mathsf{T}(Q)$. The elements of Q are called *states* of $\mathcal{A}$; the elements of Σ are called *letters* of $\mathcal{A}$; and δ is called the *transition function* of $\mathcal{A}$. For a mapping $f : X \to Y$ and $x \in X$, we denote the value of f at x by $x.f$ or $f(x)$. When the transition function δ is clear from the context, to simplify notations, $q.(\delta(a))$ will be shortened to $q.a$ where $q \in Q$ and $a \in \Sigma$. For subsets $P \subseteq Q$ and $A \subseteq \Sigma$, write $P.A$, or $P.a$ if $A = \{a\}$, for the set $\{p.a : p \in P, a \in A\}$.

Let X be a set. Finite sequences over X (including the empty sequence denoted by ϵ) are called *words*. For each nonnegative integer i, write X^i ($X^{\leq i}$, respectively) for the set of words of length i (at most i, respectively). Denote the set of all words over X by X^*.

The transition function δ extends to the mapping of the set of finite words Σ^* on $\mathsf{T}(Q)$ (still denoted by δ) via the recursion: $q.\epsilon = q$ and $q.(wa) = (q.w).a$ for every $w \in \Sigma^*$, $a \in \Sigma$ and $q \in Q$.

Let $\mathcal{A} = (Q, \Sigma, \delta)$ be an automaton. A word $w \in \Sigma^*$ is a *reset word* if $|Q.w| = 1$. An automaton that admits a reset word is called a *synchronizing* automaton. The minimum length of reset words for $\mathcal{A}$ is called the *reset threshold* of $\mathcal{A}$, denoted $\mathsf{rt}(\mathcal{A})$. For example, Figure 1 shows a synchronizing automaton $\mathcal{C}_4$ with the state set $\{1, 2, 3, 4\}$ and two letters a and b and transition function δ such that

$$\delta(i, a) = i.a = \begin{cases} 1 & \text{if } i = 4, \\ i & \text{otherwise;} \end{cases} \quad \text{and} \quad \delta(i, b) = i.b = \begin{cases} 1 & \text{if } i = 4, \\ i + 1 & \text{otherwise,} \end{cases}$$

for $i \in \{1, 2, 3, 4\}$. The shortest reset word of $\mathcal{C}_4$ is ab^3ab^3a and $\mathsf{rt}(\mathcal{C}_4) = 9$.

44th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2024).
Editors: Siddharth Barman and Sławomir Lasota; Article No. 34; pp. 34:1–34:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

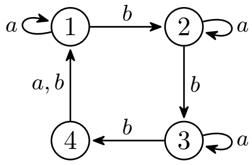


Figure 1 The automaton $\mathcal{C}_4$.

The following conjecture is the most famous conjecture of synchronizing automata.

▶ **Conjecture 1** (Černý-Starke). *Let $\mathcal{A}$ be an n-state synchronizing automaton. Then* $\mathsf{rt}(\mathcal{A}) \leq (n-1)^2$.

This conjecture is usually called Černý Conjecture [6], although it was first published in 1966 by Starke [21]. Regarding the history of Conjecture 1, we recommend [27, Section 3.1].

Černý [5] showed that there exists an n-state automaton with the reset threshold equal to $(n-1)^2$ for every n. That means the upper bound in Conjecture 1 is optimal.

For a long time, the best upper bound of reset thresholds was $\frac{n^3-n}{6}$, obtained by Pin and Frankl [9, 14]. In 2018, Szykuła [24] improved the Pin-Frankl bound. Based on Szykuła's method, Shitov [20] made a further improvement and obtained the new upper bound $cn^3 + o(n^3)$, where the coefficient c is close to 0.1654.

Although Černý Conjecture is widely open in general, it has been shown to be true in many special classes e.g. [7, 11, 25, 26]. For a summary of the state-of-the art around the Černý Conjecture, we recommend the two surveys [12, 27].

1.2 Automata containing transitive groups and our contribution

Let $\mathcal{A} = (Q, \Sigma, \delta)$ be an automaton. The *defect* of a word $w \in \Sigma^*$ is the integer $|Q| - |Q.w|$. For a non-negative integer i, write Σ_i for the set of letters of defect i.

Let $A \subseteq \Sigma_0$. Observe that δ induces a homomorphism from the free monoid A^* to the *symmetric group* $\mathsf{Sym}(Q)$. We say that $\mathcal{A}$ *contains* the permutation group $\delta(A^*)$ with the generating set $\delta(A)$. A subgroup G of $\mathsf{Sym}(Q)$ is *transitive* if $q.G = \{q.g : g \in G\} = Q$ for each $q \in Q$. We use $\mathbb{ST}$ to denote the family of synchronizing automata that contain a transitive permutation group on its state set. Note that the automaton $\mathcal{C}_4$ displayed in Figure 1 belongs $\mathbb{ST}$.

In this article, we focus on automata in $\mathbb{ST}$. Many subfamilies of $\mathbb{ST}$ have been studied in detail [1, 16, 17, 18, 19, 15, 22]. We introduce two important results that are strongly relevant to this article.

For any $A \subseteq \Sigma_0$, the minimum integer d such that $\delta(A^d) = \delta(A^*)$ is denoted by $\mathsf{d}_{\mathcal{A}}(A)$. Observe that $\mathsf{d}_{\mathcal{A}}(A)$ is the diameter of the Cayley digraph of the group $\delta(A^*)$ with the generating set $\delta(A)$. As an example, a Cayley graph of the symmetric group $\mathsf{Sym}(\{1, 2, 3\})$ is depicted in Figure 2.

The following theorem is essentially contained in the results of Rystsov [16].

▶ **Theorem 2** (Rystsov). *Let $\mathcal{A} = (Q, \Sigma, \delta) \in \mathbb{ST}$ be an n-state automaton. Then* $\mathsf{rt}(\mathcal{A}) \leq 1 + (n-2)(n-1+\mathsf{d}_{\mathcal{A}}(A))$, *where $A \subseteq \Sigma_0$ such that $\delta(A^*)$ is transitive.*

Araújo, Cameron and Steinberg [1, Theorem 9.2], using representation theory over the field of rationals $\mathbb{Q}$, have improved Rystsov's bound as displayed in Theorem 3. It is worth mentioning that a similar result can be also found in [22, Theorem 3.4].

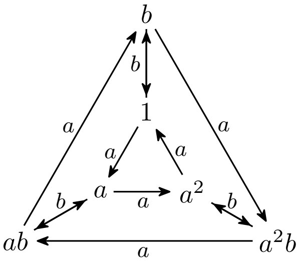


Figure 2 The Cayley digraph of $\mathsf{Sym}(\{1, 2, 3\})$ with the generating set $\{a, b\}$, where $a = (123)$ and $b = (12)$. Its vertex set is $\mathsf{Sym}(\{1, 2, 3\})$. For any two vertices x, y and a generator $g \in \{a, b\}$, there exists an arc with label g from x to y if $xg = y$.

▶ **Theorem 3** (Araújo-Cameron-Steinberg). *Let $\mathcal{A} = (Q, \Sigma, \delta) \in \mathbb{ST}$ be an n-state automaton. Then $\mathsf{rt}(\mathcal{A}) \leq 1 + (n - 2)(n - m + \mathsf{d}_{\mathcal{A}}(A))$, where $A \subseteq \Sigma_0$ such that $\delta(A^*)$ is transitive and m is the maximum dimension of an irreducible $\mathbb{Q}(\delta(A^*))$-module of $\mathbb{Q}^Q$*

In the case that $\mathsf{d}_{\mathcal{A}}(A)$ is small (a linear function of n), Theorems 2 and 3 bound $\mathsf{rt}(\mathcal{A})$ from above by a quadratic function of n, or even verify Černý Conjecture [16, 1, 22]. However, generally, $\mathsf{d}_{\mathcal{A}}(A)$ is not a linear function of n: in the case that $A = \{(12), (12, \dots, n)\}$, we have $\mathsf{d}_{\mathcal{A}}(A) \approx \frac{3}{4}n^2$ (asymptotically) [28].

In this article, we obtain Theorem 13 which improves Theorem 2 in a different way. As an application of Theorem 13, we obtain the following result.

▶ **Theorem 4.** *Let $\mathcal{A} \in \mathbb{ST}$ be an n-state automaton. If $\Sigma = \Sigma_0 \cup \Sigma_1$, then $\mathsf{rt}(\mathcal{A}) \leq 2n^2 - 7n + 7$.*

1.3 Approach and layout

To prove Theorems 4 and 13, we use the so-called *extension method* which is based on Proposition 5. The proof of Proposition 5 can be found in many papers (e.g. [27, Section 3.4]).

Let $\mathcal{A} = (Q, \Sigma, \delta)$ is an automaton. For a subset $S \subseteq Q$ and a word $w \in \Sigma^*$, write $S.w^{-1}$ for the set $\{q \in Q : q.w \in S\}$, that is, the set of states from which upon reading w, the automaton reaches a state in S. A subset $S \subseteq Q$ is *extended* by a word $w \in \Sigma^*$ if $|S.w^{-1}| > |S|$. A subset $S \subseteq Q$ is called *m-extensible*, if S is extended by a word of length at most m.

▶ **Proposition 5.** *Let $\mathcal{A} = (Q, \Sigma, \delta)$ be a synchronizing automaton. If every nonempty proper subset S of Q is m-extensible, then $\mathsf{rt}(\mathcal{A}) \leq 1 + (n - 2)m$.*

The remaining of this article will proceed as follows. In Section 2, combining the extension method and a dimensional argument for a linear structure, we establish a upper bound for the reset thresholds of automata in $\mathbb{ST}$, see Theorem 13. In Section 3, using some graph theoretical techniques, we present a proof of Theorem 4. Using these graph theoretical techniques, we can slightly improve some known results about reset thresholds. At the end, we summarize our results in Section 4.

2 Linear structure

In this section, we will encode some information of an n-state synchronizing automaton into some objects in $\mathbb{Q}^n$. Using the linear structure of $\mathbb{Q}^n$, we will obtain a upper bound for its reset threshold.

Let $\mathcal{A} = (Q, \Sigma, \delta)$ be an n-state automaton. We always assume that $Q = \{1, \ldots, n\}$. Fix a subset A of Σ_0 and denote $\delta(A^*)$ by G.

Firstly, let us recall some concepts in linear space. Let $X \subseteq \mathbb{Q}^n$. The linear subspace *spanned* by X, denoted by $\mathsf{span}(X)$, is defined as

$$\mathsf{span}(X) := \left\{ \sum_{x \in X} c_x x : c_x \in \mathbb{Q} \right\}.$$

The *cone* generated by X, denoted $\mathsf{cone}(X)$, is the set

$$\mathsf{cone}(X) := \left\{ \sum_{x \in X} c_x x : c_x \in \mathbb{Q}_{\geq 0} \right\}$$

where $\mathbb{Q}_{\geq 0}$ is the set of non-negative rationals. Write $\langle \cdot, \cdot \rangle$ for the *standard inner product* of $\mathbb{Q}^n$, that is the map such that $\langle x, y \rangle = \sum_{i=1}^{n} x(i)y(i)$ for every $x, y \in \mathbb{Q}^n$. The *polar cone* of X, denoted X°, is the set

$$X^\circ := \{y \in \mathbb{Q}^n : \langle x, y \rangle \leq 0, \forall x \in X\}.$$

If X is a linear subspace of $\mathbb{Q}^n$, the *orthogonal complement* of X, denoted $X^\perp$, is defined as

$$X^\perp = \{y : \langle x, y \rangle = 0, \forall x \in X\}.$$

It clearly holds $X^\circ = X^\perp$ in the case that X is a linear subspace.

Next we will encode some information of an n-state synchronizing automaton into some objects in $\mathbb{Q}^n$. For a subset $S \subseteq Q$, define $\mathbf{1}_S$ to be the $(1 \times n)$-vector over $\mathbb{Q}$ such that its i-th coordinate is

$$\mathbf{1}_S(i) = \begin{cases} 1 & \text{if } i \in S, \\ 0 & \text{otherwise.} \end{cases}$$

For any $q \in Q$, to simplify notation, we write $\mathbf{1}_q$ for $\mathbf{1}_{\{q\}}$. Let w be an arbitrary word in Σ^*. Define $[w]$ to be the $(n \times n)$-matrix over $\mathbb{Q}$ such that $\mathbf{1}_q[w] = \mathbf{1}_{q.w^{-1}}$ for all $q \in Q$. It is clear that $\mathbf{1}_S[w] = \mathbf{1}_{S.w^{-1}}$ for all $S \subseteq Q$. Define $\mathbf{k}_w$ for the $(1 \times n)$-vector over $\mathbb{Q}$ such that its i-th coordinate is

$$\mathbf{k}_w(i) = |i.w^{-1}| - 1.$$

For every $g \in G$, set $\mathbf{k}_g$ to be $\mathbf{k}_w$ and $[g]$ to be $[w]$, where $w \in \Sigma^*$ is an arbitrary word such that $\delta(w) = g$.

▶ **Example 6.** Consider the automata $\mathcal{C}_4$ (see Figure 1) and the words a, b and ab. One can calculate that

$$\begin{gathered} 1.a^{-1} = \{1,4\}, \quad 2.a^{-1} = \{2\}, \quad 3.a^{-1} = \{3\}, \quad 4.a^{-1} = \emptyset; \\ 1.b^{-1} = \{4\}, \quad 2.b^{-1} = \{1\}, \quad 3.b^{-1} = \{2\}, \quad 4.b^{-1} = \{1\}; \\ 1.ab^{-1} = \emptyset, \quad 2.ab^{-1} = \{1,4\}, \quad 3.ab^{-1} = \{2\}, \quad 4.ab^{-1} = \{3\}. \end{gathered}$$

And then $\mathbf{k}_a = (1, 0, 0, -1)$, $\mathbf{k}_b = (0, 0, 0, 0)$, and $\mathbf{k}_{ab} = (-1, 1, 0, 0)$.

A sequence $(X_i)_{i \geq 0}$ is called *eventually constant* if there exists an integer $j \geq 0$ such that for all $k > j$, $X_k = X_j$. For an eventually constant sequence $\mathcal{X} = (X_i)_{i \geq 0}$, the minimum integer j such that for all $k > j$, $X_k = X_j$ is called *transient length* of $\mathcal{X}$, denoted by $\mathsf{len}(\mathcal{X})$

and we denote the *limit* of the sequence $\mathcal{X}$ by $\lim \mathcal{X}$ which clearly equals $X_{\mathsf{len}(\mathcal{X})}$. In the following, we will define some eventually constant sequences which play a crucial role in our proof.

Define $\mathcal{T}(\mathcal{A}, A) = (T_i)_{i\geq 0}$ and $\mathcal{K}(\mathcal{A}, A) = (K_i)_{i\geq 0}$, to be the two sequences such that

$$T_i := \left\{\mathbf{k}_w : w \in (\Sigma \setminus \Sigma_0)A^{\leq i}\right\} \quad \text{and} \quad K_i := \mathsf{cone}(T_i),$$

for every $i \geq 0$. To simplify notations, without ambiguity, we write $\mathcal{T}$ and $\mathcal{K}$ for $\mathcal{T}(\mathcal{A}, A)$ and $\mathcal{K}(\mathcal{A}, A)$, respectively.

Since $\langle A\rangle$ is a finite group, both $\mathcal{T}$ and $\mathcal{K}$ are eventually constant. Denote $\lim \mathcal{T}$ and $\lim \mathcal{K}$ by T_∞ and K_∞, respectively. Observe that $K_\infty = \mathsf{cone}(T_\infty)$ and then

$$\mathsf{len}(\mathcal{T}) \geq \mathsf{len}(\mathcal{K}). \tag{1}$$

We begin with some elementary results. According to the definition, it clearly holds that

$$|S.w^{-1}| - |S| = \langle 1_S, \mathbf{k}_w\rangle \tag{2}$$

for every $S \subseteq Q$ and $w \in \Sigma^*$. As a consequence, we have the following lemma.

▶ **Lemma 7.** *Let $S \subseteq Q$.*

1. *The subset S is extended by w if and only if $\langle 1_S, \mathbf{k}_w\rangle > 0$.*
2. *If $1_S \in (K_\infty)^\circ$ then $|S.a^{-1}| = |S|$ for all $a \in \Sigma$.*
3. *For every vector $x \in K_\infty$, $\sum_{i=1}^{n} x(i) = \langle 1_Q, x\rangle = 0$.*

We say that $\mathcal{A}$ is *strongly connected* if for every two states p and q, there exists a word $w \in \Sigma^*$ such that $p.w = q$.

▶ **Lemma 8.** *Assume that $\mathcal{A} = (Q, \Sigma, \delta)$ is synchronizing and strongly connected. Let S be a nonempty proper subset of Q. If $1_S \in (K_\infty)^\circ$, then there exists a word $w \in \Sigma^*$ such that $1_{S.w^{-1}} \notin K^\circ$.*

Proof. Since $\mathcal{A}$ is synchronizing and strongly connected, there exists a word

$$u = u_1 u_2 \cdots u_t \in \Sigma^*$$

such that $S.u^{-1} = Q$. Since $|S| < |Q|$, by Lemma 7 Item 2, there exists an integer $t' < t$ such that $1_{S.v^{-1}} \notin (K_\infty)^\circ$, where $v = u_1 u_2 \cdots u_{t'}$. ◀

Due to Lemma 8, we can define $\ell(S)$ to be the length of a shortest word w such that $1_{S.w^{-1}} \notin (K_\infty)^\circ$ for every nonempty proper subset $S \subsetneq Q$.

▶ **Proposition 9.** *Assume that $\mathcal{A}$ is synchronizing and strongly connected. Let S be a nonempty proper subset of Q. Then S is $(\mathsf{len}(\mathcal{K}) + \ell(S) + 1)$-extensible.*

Proof. Let $k = \mathsf{len}(\mathcal{K})$ and $\ell = \ell(S)$. By Lemma 8, there exists an ℓ-length word $w = (w_1, \ldots, w_\ell) \in \Sigma^*$ such that $1_S\,.w^{-1} \notin (K_\infty)^\circ$. Since

$$S.(w_2, \ldots, w_\ell)^{-1} \in (K_\infty)^\circ,$$

by Lemma 7 Item 2, $|S.w^{-1}| = |S|$.

Let $P = S.w^{-1}$. Since $1_P \notin (K_\infty)^\circ$, there exists a vector $x \in K_\infty$ such that $\langle x, 1_P\rangle > 0$. Since $K_\infty = \mathsf{cone}(T_k)$, there exists a vector $y \in T_k$ such that $\langle y, 1_P\rangle > 0$. By the definition of T_k, we can find a word $u \in (\Sigma \setminus \Sigma_0)A^{\leq k}$ such that $y = \mathbf{k}_u$. By Lemma 7 Item 1, $|P.u^{-1}| > |P| = |S|$. Hence, uw extends S and then S is $(k + \ell + 1)$-extensible. ◀

If $\mathsf{len}(\mathcal{K})$ and $\ell(S)$ can be bounded by a linear function of n, using Proposition 5, one can bound $\mathsf{rt}(\mathcal{A})$ by a quadratic function of n. In general, it is hard to estimate $\mathsf{len}(\mathcal{K})$ and $\ell(S)$ of an automaton. However, in the next section, we will establish some linear bounds for $\mathsf{len}(\mathcal{K})$ with the assumption $\mathcal{A} \in \mathbb{ST}$ and $\Sigma = \Sigma_0 \cup \Sigma_1$. And, in the rest of this section, we will establish the following linear bound for $\ell(S)$ in the case that K_∞ is a linear subspace of $\mathbb{Q}^n$.

▶ **Proposition 10.** *Assume that $\mathcal{A}$ is synchronizing and strongly connected. If K_∞ is a linear subspace of $\mathbb{Q}^n$, then $\ell(S) \le n - 1 - \mathsf{dim}(K_\infty)$ for every nonempty proper subset $S \subsetneq Q$.*

Before proving Proposition 10, we show that "G is transitive" implies "K_∞ is a linear subspace of $\mathbb{Q}^n$".

▶ **Lemma 11.** *If G is transitive, then K_∞ is the linear subspace spanned by T_∞.*

Proof. It is sufficient to prove $-x \in K_\infty = \mathsf{cone}(T_\infty)$ for each $x \in T_\infty$. Take an arbitrary $x \in T_\infty$ and let

$$y := \sum_{g \in G} x[g].$$

It is clear that $y \in K_\infty$. Let i and j be two arbitrary integers in $\{1, \ldots, n\}$. Since G is transitive, there exists $h \in G$ such that $i.h = j$. Note that $y[h](i) = y(j)$ and

$$y[h] = \sum_{g \in G} x[g][h] = \sum_{g \in G} x[g] = y.$$

Then $y(i) = y(j)$. By the arbitrariness of i and j, it holds that $y = c\,\mathbb{1}_Q$ for some $c \in \mathbb{Q}$. Since $y \in K_\infty$, by Lemma 7 Item 3, we have $\sum_{i=1}^{n} y(i) = 0$. This implies $c = 0$ and then

$$-x = \sum_{g \in G \setminus \{\mathsf{id}\}} x[g],$$

where id is the identity map in $\mathsf{Sym}(Q)$. Since $x[g] \in T_\infty$ for all $g \in G$, we obtain that $-x \in K_\infty$ as wanted. ◀

Now, we go back to prove Proposition 10. The following dimension argument plays a crucial role in our proof.

▶ **Lemma 12.** *Let L be a subspace of $\mathbb{Q}^n$ and a non-zero vector $x \in L$. If there exists a word $w \in \Sigma^*$ such that $x[w] \notin L$ then there exists a word $w' \in \Sigma^*$ such that $x[w'] \notin L$ and $|w'| \le \mathsf{dim}(L)$.*

Proof. For every nonnegative integer i, define $L_i := \mathsf{span}(xv : v \in \Sigma^{\le i})$. Observe that there exists a unique integer j such that

$$L_0 \subsetneq L_1 \subsetneq \cdots \subsetneq L_j = L_{j+1} = \cdots$$

Let t be the minimum integer such that $L_t \not\subseteq L$. Observing that $t \le j$, we have

$$t = \mathsf{dim}(L_0) + t - 1 \le \mathsf{dim}(L_{t-1}) \le \mathsf{dim}(L).$$

Then there exists a word w' of length $\le \mathsf{dim}(L)$ such that $x[w'] \notin L$. ◀

Proof of Proposition 10. Since K_∞ is a linear subspace of $\mathbb{Q}^n$, it holds that $(K_\infty)^\circ = (K_\infty)^\perp$ is also a linear subspace of $\mathbb{Q}^n$. Observe that $1_Q \in (K_\infty)^\circ$. Then we can decompose $(K_\infty)^\circ$ as $(K_\infty)^\circ = V_0 \oplus V_1$, where

$$V_0 = \{x \in (K_\infty)^\circ : \langle 1_Q, \mathbf{k}_w \rangle = 0\} \quad \text{and} \quad V_1 = \mathsf{span}(1_Q).$$

For every subset $R \subseteq Q$, define $\mathbf{p}_R := 1_R - \frac{|R|}{n} 1_Q$. For each $R \subseteq Q$, observe that $1_R \in (K_\infty)^\circ$ if and only if $\mathbf{p}_R \in V_0$.

Let S be a nonempty proper subset of Q such that $1_S \in (K_\infty)^\circ$. Since $\mathcal{A}$ is synchronizing and strongly connected, let w' be a reset word such that $Q.w' \subseteq S$. Then

$$\mathbf{p}_S[w'] = 1_S[w'] - \frac{|S|}{n} 1_Q[w'] = \left(1 - \frac{|S|}{n}\right) 1_Q \notin V_0.$$

Let $w = va$ be a shortest word such that $\mathbf{p}_S[w] \notin V_0$. Lemma 12 provides that the length of w is at most $\mathsf{dim}(V_0) = n - 1 - \mathsf{dim}(K_\infty)$.

We will complete the proof by showing $1_{S.w^{-1}} \notin (K_\infty)^\circ$. Note that $\mathbf{p}_S[v] = 1_{S.v^{-1}} - \frac{|S|}{n} 1_Q$. Since $\mathbf{p}_S[v], 1_Q \in (K_\infty)^\circ$, we have $1_{S.v^{-1}} \in (K_\infty)^\circ$. By Lemma 7 Item 2, $|S| = |S.v^{-1}| = |S.w^{-1}|$. Hence,

$$\mathbf{p}_{S.w^{-1}} = 1_{S.w^{-1}} - \frac{|S.w^{-1}|}{n} 1_Q = 1_{S.w^{-1}} - \frac{|S|}{n} 1_Q = \mathbf{p}_S[w] \notin V_0$$

which is equivalent to $1_{S.w^{-1}} \notin (K_\infty)^\circ$. ◀

Combining Propositions 5, 9, and 10 and Lemma 11, we establish the following bound.

▶ **Theorem 13.** *Let $\mathcal{A} = (Q, \Sigma, \delta) \in \mathbb{ST}$ be an n-state automaton. Then $\mathsf{rt}(\mathcal{A}) \leq 1 + (n - 2)(n - \mathsf{dim}(K_\infty) + \mathsf{len}(\mathcal{K}(\mathcal{A}, A)))$, where $A \subseteq \Sigma_0$ such that $\delta(A^*)$ is transitive.*

▶ Remark 14. Note that $\mathsf{dim}(K_\infty) \geq 1$ and $\mathsf{d}_{\mathcal{A}}(A) \geq \mathsf{len}(\mathcal{K}(\mathcal{A}, A))$.

1. Theorem 13 improves Theorem 2.
2. One of Theorem 3 and Theorem 13 cannot deduce the other one. If we only want to establish a quadratic upper bound for reset thresholds of a special class of automata, it is easier to establish a linear upper bound for $\mathsf{len}(\mathcal{K}(\mathcal{A}, A))$ than for $\mathsf{d}_{\mathcal{A}}(A)$. In this sense, Theorem 13 may have more advantages.

3 Rystsov digraphs

This section is divided into two parts:

- In Section 3.1, we establish some results for digraphs.
- In Section 3.2, we derive some directed graphs from automata. Using the results in Section 3.1 and Theorem 13, we prove Theorem 4.

3.1 Digraphs

Firstly, we recall some notations of digraphs. A *digraph* $\Gamma = (V, E)$ is an ordered pair of sets such that $E \subseteq V \times V$. The set V is called the *vertex set* of Γ and the E is called the *arc set* of Γ. We assume that the digraphs in this section are *loop-free*, that is, (v, v) is not an arc for every vertex v. Let u and v be two vertices of Γ. A sequence of vertices $(v_1, \ldots, v_t)$ is called a *path* from u to v if $v_1 = u$, $v_t = v$ and $(v_i, v_{i+1}) \in E$ for every $1 \leq i \leq t - 1$. We say that u and v are *connected* if there exists a sequence of vertices $(v_1, \ldots, v_t)$ such that $v_1 = u$, $v_t = v$ and either $(v_i, v_{i+1}) \in E$ or $(v_{i+1}, v_i) \in E$ for every $1 \le i \le t-1$. The *strong connectivity* of Γ, denoted $\mathsf{sc}(\Gamma)$, is the equivalence relation such that $(u,v) \in \mathsf{sc}(\Gamma)$ if and only if there exist a path from u to v and a path from v to u. The *weak connectivity* of Γ, denoted $\mathsf{wc}(\Gamma)$, is the equivalence relation such that $(u,v) \in \mathsf{wc}(\Gamma)$ if and only if u, v are connected. A strongly connected component C of Γ is called a *sink component* of Γ if there is no arc $(u,v) \in E$ such that $u \in C$ and $v \notin C$; is called a *source component* of Γ if there is no arc $(u,v) \in E$ such that $u \notin C$ and $v \in C$. A strongly connected component is *non-sink* (*non-source*, resp.) if it is not a sink (source, resp.) component of Γ. Write $\mathsf{SCC}(\Gamma)$ ($\mathsf{WCC}(\Gamma)$, $\mathsf{SinkC}(\Gamma)$, resp.) for the set of strongly connected components (weakly connected components, sink components, resp.) of Γ. Equivalence classes of $\mathsf{sc}(\Gamma)$ ($\mathsf{wc}(\Gamma)$ resp.,) are called a *strongly connected components* (*weakly connected components* resp.,) of Γ.

Let $V = \{1, \dots, n\}$ and $A \subseteq \mathsf{Sym}(V)$. We will say that the sequence $\Gamma_0, \Gamma_1, \dots$ is an *A-growth* of Γ_0 if

- $\Gamma_0 = (V, E_0)$ is a digraph with vertex set V;
- for every positive integer i, $\Gamma_i = (V, E_i)$ is the digraph such that $E_i = \{(p.w, q.w) : (p,q) \in E_0, w \in A^{\le i}\}$.

This concept, also called Rystsov Digraphs, was firstly used in [17], and appears widely in the research of synchronizing automata [3, 4, 10, 17, 19].

In the rest of Section 3.1, we set $\Gamma_0, \Gamma_1, \dots$ is an A-growth of Γ_0. Write $G = \langle A \rangle$. Since G is finite, the sequence $\Gamma_0, \Gamma_1, \dots$ is eventually constant. Denote $\mathsf{lim}(\Gamma_0, \Gamma_1, \dots)$ by Γ_∞.

▶ **Lemma 15.** *If G is transitive,* $\mathsf{wc}(\Gamma_\infty) = \mathsf{sc}(\Gamma_\infty)$.

Proof. For any element $g \in G$ and two vertices $u, v \in V$, observe that $(u.g, v.g)$ is an arc of Γ_∞ if and only if (u,v) is an arc of Γ_∞. And then every element of G is a graph automorphism of Γ_∞.

For any vertex $v \in V$, let $R(v)$ be the subset of vertices such that $u \in R(v)$ if and only if there exists a path from v to u.

Let (u,v) be an arc of Γ_∞. It is clear that $R(v) \subseteq R(u)$. Since G is transitive, we can pick $g \in G$ such that $u.g = v$. Since g is a graph automorphism, it induces a bijection from $R(u)$ to $R(v)$. Then $R(u) = R(v)$. Hence, if two vertices x and y belong to the same weakly connected component of Γ_∞, it holds $R(x) = R(y)$. This completes the proof. ◀

For all $(p,q) \in V \times V$, the *incidence vector* of (p,q) is the vector $1_p - 1_q$, denoted $\chi_{(p,q)}$. For a digraph $\Gamma = (V, E)$, write $L(\Gamma)$ for the subspace $\mathsf{span}(\chi_e : e \in E)$ of $\mathbb{Q}^n$. The following result is well-known in algebraic graph theory (see [2, Chapter 4]).

▶ **Lemma 16.** *Assume that $W_1, \dots, W_d$ are all weakly connected components of a digraph $\Gamma = (V, E)$. Then the subspace $(L(\Gamma))^\perp$ is the d-dimensional subspace* $\mathsf{span}(1_{W_1}, \dots, 1_{W_d})$.

▶ **Lemma 17.** *Let $d = |\mathsf{SCC}(\Gamma_\infty)|$. If the arc set of Γ_0 is nonempty and G is transitive, then* $\mathsf{wc}(\Gamma_{n-d-1}) = \mathsf{wc}(\Gamma_\infty) = \mathsf{sc}(\Gamma_\infty)$.

Proof. By Lemma 15, $\mathsf{wc}(\Gamma_\infty) = \mathsf{sc}(\Gamma_\infty)$. It is sufficient to show $\mathsf{wc}(\Gamma_{n-d-1}) = \mathsf{wc}(\Gamma_\infty)$.

Define $\mathcal{L} = (L_i)_{i \ge 0}$ to be the sequence of linear spaces, where $L_i := L(\Gamma_i)$ for every $i \ge 0$. Write $L_\infty = \mathsf{lim}\,\mathcal{L}$. By Lemma 16, $\mathsf{dim}((L_\infty)^\perp) = d$ and then $\mathsf{dim}(L_\infty) = n - d$. For every $i < \mathsf{len}(\mathcal{L})$, since L_i and L_{i+1} are linear subspaces with $L_i \subsetneq L_{i+1}$, we have $\mathsf{dim}(L_i) < \mathsf{dim}(L_{i+1})$. Due to $E_0 \ne \emptyset$, it holds that $\mathsf{dim}(L_0) \ge 1$ and then $L_{n-d-1} = L_\infty$. By Lemma 16, $\mathsf{wc}(\Gamma_{n-d-1}) = \mathsf{wc}(\Gamma_\infty)$. ◀

▶ **Lemma 18.** *Assume that $E_0 \ne \emptyset$ and G is transitive. For every $v \in V$, there exist two vertices $p, q \in V$ such that $(v,p), (q,v) \in E_{n-1}$.*

Proof. Since $E_0 \neq \emptyset$, let x, y be two vertices such that $(x, y) \in E_0$. Let v be an arbitrary vertex in V. Since G is transitive, there exist two words $w, w' \in A^{\leq n-1}$ such that $x.w = y.w' = v$. Then $(v, y.w), (x.w', v) \in E_{n-1}$. ◀

▶ **Lemma 19.** *Let i be a positive integer. If $\mathsf{sc}(\Gamma_{i+1}) = \mathsf{sc}(\Gamma_i) \neq \mathsf{sc}(\Gamma_\infty)$ and G is transitive, then there exists a non-sink strongly connected component C of Γ_i and $a \in A$ such that $C.a$ is a sink component of Γ_i.*

Proof. Since $\mathsf{sc}(\Gamma_{i+1}) = \mathsf{sc}(\Gamma_i)$, every $a \in A$ induces a permutation on $\mathsf{SCC}(\Gamma_i)$, denoted by a', such that $X.a' = X.a \in \mathsf{SCC}(\Gamma_i)$.

Case 1: There exists a non-sink strongly connected component of Γ_i. Assume, for a contradiction, that every non-sink strongly connected component C and $a \in A$ satisfy $C.a \in \mathsf{SCC}(\Gamma_i) \setminus \mathsf{SinkC}(\Gamma_i)$. Let A' be the set $\{a' : a \in A\}$. Since $G = \langle A \rangle$ is transitive on V, the permutation group $\langle A' \rangle$ is transitive on $\mathsf{SCC}(\Gamma_i)$. Then there exist $C \in \mathsf{SCC}(\Gamma_i) \setminus \mathsf{SinkC}(\Gamma_i)$ and $a \in A$ such that $C.a \in \mathsf{SinkC}(\Gamma_i)$.

Case 2: There is no non-sink strongly connected component of Γ_i. In this case, we have $\mathsf{wc}(\Gamma_i) = \mathsf{sc}(\Gamma_i)$. For every $a \in A$ and $X \in \mathsf{SCC}(\Gamma_i)$, since $X.a \in \mathsf{SCC}(\Gamma_i)$, there are no arcs outside strongly connected components of Γ_{i+1}. Then $\mathsf{wc}(\Gamma_{i+1}) = \mathsf{sc}(\Gamma_{i+1})$. Using this argument repeatedly, we have $\mathsf{sc}(\Gamma_{i+1}) = \mathsf{sc}(\Gamma_{i+2}) = \cdots = \mathsf{sc}(\Gamma_\infty)$ which contradicts with $\mathsf{sc}(\Gamma_{i+1}) \neq \mathsf{sc}(\Gamma_\infty)$. ◀

▶ **Lemma 20.** *Let i be a non-negative integer such that $\mathsf{sc}(\Gamma_i) \neq \mathsf{sc}(\Gamma_\infty)$. If G is transitive, then either*

$$|\mathsf{SCC}(\Gamma_i)| > |\mathsf{SCC}(\Gamma_{i+1})|$$

or

$$|\mathsf{SinkC}(\Gamma_i)| > |\mathsf{SinkC}(\Gamma_{i+1})|.$$

Proof. If $|\mathsf{SCC}(\Gamma_i)| > |\mathsf{SCC}(\Gamma_{i+1})|$, we are done. Otherwise, since $\mathsf{sc}(\Gamma_i) \subseteq \mathsf{sc}(\Gamma_{i+1})$, we have $\mathsf{sc}(\Gamma_i) = \mathsf{sc}(\Gamma_{i+1})$ and

$$\mathsf{SinkC}(\Gamma_i) \supseteq \mathsf{SinkC}(\Gamma_{i+1}). \tag{3}$$

By Lemma 19, there exist $C \in \mathsf{SCC}(\Gamma_i) \setminus \mathsf{SinkC}(\Gamma_i)$ and $a \in A$ such that $C.a \in \mathsf{SinkC}(\Gamma_i)$. Since $C \in \mathsf{SCC}(\Gamma_i) \setminus \mathsf{SinkC}(\Gamma_i)$, there exists an arc $(p, q) \in E_i$ such that $p \in C$ and $q \notin C$. Note that $(p.a, q.a) \in E_{i+1}$. Since $p.a \in C.a$ and $q.a \notin C.a$, it holds that $C.a$ is a non-sink component of Γ_{i+1}. By Equation (3), we have $|\mathsf{SinkC}(\Gamma_i)| > |\mathsf{SinkC}(\Gamma_{i+1})|$. ◀

▶ **Lemma 21.** *Let $d = |\mathsf{SCC}(\Gamma_\infty)|$. If G is transitive and $d > \frac{n}{3}$, then $\mathsf{sc}(\Gamma_n) = \mathsf{sc}(\Gamma_\infty)$.*

Proof. Since G is transitive, d divides n. Then $d \in \{\frac{n}{2}, n\}$. Noting that, by Lemma 15, every weakly connected component of Γ_∞ is strongly connected. Since each strongly connected component of Γ_∞ has at most 2 vertices, it is clear that there exist at most n arcs in Γ_∞. Note that $|E_i| < |E_{i+1}|$ for every integer i such that $\Gamma_i \neq \Gamma_\infty$. Then $\mathsf{sc}(\Gamma_n) = \mathsf{sc}(\Gamma_\infty)$. ◀

▶ **Lemma 22.** *Let $d = |\mathsf{SCC}(\Gamma_\infty)|$. If G is transitive and $d \leq \frac{n}{3}$, then $\mathsf{sc}(\Gamma_{2n-3d-1}) = \mathsf{sc}(\Gamma_\infty)$.*

Proof. Let m be the minimum integer such that $\mathsf{sc}(\Gamma_m) = \mathsf{sc}(\Gamma_\infty)$. Define

$$f(i) = |\mathsf{SCC}(\Gamma_i)| + |\mathsf{SinkC}(\Gamma_i)|,$$

for each $i \geq 0$.

Consider Γ_{n-1}. By Lemma 17, Γ_{n-1} has d weakly connected components and $\mathsf{wc}(\Gamma_{n-1}) = \mathsf{sc}(\Gamma_m)$. In the case that $m \leq n-1$, then $m \leq 2n-3d-1$ and we are done.

Now we assume that $m > n-1$. Let C be a weakly connected component of Γ_{n-1} but not a strongly connected component of Γ_{n-1}. Define

$a :=$ the number of source components of Γ_{n-1} in C;

$b :=$ the number of non-source non-sink components of Γ_{n-1} in C;

$c :=$ the number of sink components of Γ_{n-1} in C.

Let D be either a source component or a sink component of Γ_{n-1} in C. By Lemma 18, there exists an arc in D and then there are at least two vertices in D. This implies

$$2a + b + 2c \leq |C| = \frac{n}{d}.$$

Since $a \geq 1$, we have

$$(a + b + c) + c \leq \frac{n}{d} - 1. \tag{4}$$

The left hand side of Equation (4) is the sum of the number of strongly connected components and the number of sink components of Γ_{n-1} in C. Since $d \leq \frac{n}{3}$, we have

$$\begin{aligned} f(n-1) &\leq (\frac{n}{d} - 1)x + 2(d - x) && \text{(by Equation (4))} \\ &\leq (\frac{n}{d} - 1)d && \text{(by } n/d - 1 \geq 2) \\ &= n - d, \end{aligned}$$

where $x = |\,\mathsf{WCC}(\Gamma_{n-1}) \setminus \mathsf{SCC}(\Gamma_{n-1})|$.

Since $f(m) = 2d$, using Lemma 20, we have

$$m - (n-1) \leq f(n-1) - f(m) \leq (n-d) - 2d = n - 3d.$$

Hence, $m \leq 2n - 3d - 1$. ◀

▶ Remark 23.

1. In the case that Γ_∞ is strongly connected and $n \geq 3$, Lemmas 21 and 22 show that $\mathsf{sc}(\Gamma_{2n-4}) = \mathsf{sc}(\Gamma_\infty)$. This slightly improves [17, Theorem 2], [10, Lemma 6] and [19, Theorem 2]. And then one can slightly improve the bounds of reset thresholds in [17, Theorem 3], [10, Theorem 7] and [19, Theorem 4].
2. Assume G is transitive. Lemmas 21 and 22 show that $\mathsf{sc}(\Gamma_{O(n)}) = \mathsf{sc}(\Gamma_\infty)$. The following example shows that $\Gamma_{O(n)} = \Gamma_\infty$ is not true.
   A permutation group $G \subseteq \mathsf{Sym}(Q)$ is called 2-*homogeneous* if for every 2-element subsets $X, Y \subseteq Q$, there exists $g \in G$ such that $X.g = Y$. Observe that a 2-homogeneous permutation group is transitive.
   In [10, Section 3], for every odd integer $n \geq 7$, Gonze, Gusev, Gerencsér, Jungers and Volkov constructed two permutations $a, b \in \mathsf{Sym}(n)$ such that
   - $\langle a, b \rangle$ is 2-homogeneous;
   - for any word $w \in \{a, b\}^*$, if $\{2, 4\}.w = \{\frac{n-1}{2}, \frac{n+3}{2}\}$, then $|w| \geq \frac{n^2}{4} + O(n)$.

   Let $\Gamma_0 = (V, E)$ be a digraph such that $V = \{1, \ldots, n\}$ and $E = \{(2, 4)\}$. Let $\Gamma_0, \Gamma_1, \ldots$ be the $\{a, b\}$-growth of Γ_0. Since $\langle a, b \rangle$ is 2-homogeneous, Γ_∞ is a complete digraph. By the second property of these two permutation, if $\Gamma_m = \Gamma_\infty$, then $m \geq \frac{n^2}{4} + O(n)$.
   Meanwhile, the above two permutations also provide a negative answer for [1, Problem 12.39].

3.2 Automata

In this subsection, we will define a sequence of digraphs with respect to an automaton $\mathcal{A} = (Q, \Sigma, \delta)$ and $A \subseteq \Sigma_0$.

For a 1-defect word $w \in \Sigma^*$, the *excluded state* of w is the state such that $\mathsf{excl}(w) \notin Q.w$, denoted $\mathsf{excl}(w)$; the *duplicate state* is the state q such that $|q.w^{-1}| > 1$, denoted $\mathsf{dupl}(w)$.

For $i \geq 0$, define $\Gamma_i := (Q, E_i)$ to be the digraph where

$$E_i := \left\{(\mathsf{excl}(w), \mathsf{dupl}(w)) : w \in \Sigma_1 A^{\leq i}\right\}.$$

▶ **Lemma 24.** *The sequence $(\Gamma_0, \Gamma_1, \dots)$ is the $\delta(A)$-growth of Γ_0.*

Proof. We need to prove $E_{i+1} = E_i \cup E_i.\Sigma_0$ for every $i \geq 0$. Let i be an arbitrary nonnegative integer.

Let $(p, q) \in E_i$. Take $w \in \Sigma_1 \Sigma_0^{\leq i}$ such that $p = \mathsf{excl}(w)$ and $q = \mathsf{dupl}(w)$. By directly computing, we have $\mathsf{excl}(wa) = \mathsf{excl}(w).a$ and $\mathsf{dupl}(wa) = \mathsf{dupl}(w).a$ for all $a \in \Sigma_0$. Then $(p.a, q.a) \in E_{i+1}$ which implies $E_{i+1} \supseteq E_i \cup E_i.\Sigma_0$.

Let $(x, y) \in E_{i+1}$. Take $w' \in \Sigma_1 \Sigma_0^{\leq i+1}$ such that $x = \mathsf{excl}(w')$ and $y = \mathsf{dupl}(w')$. If the length of w' is less than $i + 2$, then $(x, y) \in E_i$. Otherwise, $w' = wa$ where $w \in \Sigma_1 \Sigma_0^i$ and $a \in \Sigma_0$. It is clear that $(\mathsf{excl}(w), \mathsf{dupl}(w)) \in E_i$ and $(\mathsf{excl}(w).a, \mathsf{dupl}(w).a) = (x, y)$. Then $E_{i+1} \subseteq E_i \cup E_i.\Sigma_0$. ◀

▶ **Proposition 25.** *Assume that $\mathcal{A} = (Q, \Sigma, \delta) \in \mathbb{ST}$. If K_∞ is a linear subspace of $\mathbb{Q}^n$ and $\Sigma = \Sigma_0 \cup \Sigma_1$, then*

$$\mathsf{len}(\mathcal{K}) \leq \begin{cases} n & \text{if } \mathsf{dim}(K_\infty) = \frac{n}{2}, \\ 3\,\mathsf{dim}(K_\infty) - n - 1 & \text{otherwise.} \end{cases}$$

Proof. Recall the definition of T_i that $T_i := \{\mathbf{k}_w : w \in \Sigma_{\geq 1} A^{\leq i}\}$ for $i \geq 0$. Since $\Sigma = \Sigma_0 \cup \Sigma_1$, we have $T_i = \{-\chi_e : e \in E_i\}$ for $i \geq 0$. Let m be the minimal integer such that $\mathsf{sc}(\Gamma_m) = \mathsf{sc}(\Gamma_\infty)$. By Lemmas 15 and 24, $\mathsf{sc}(\Gamma_\infty) = \mathsf{wc}(\Gamma_\infty)$ and $\mathsf{sc}(\Gamma_m) = \mathsf{wc}(\Gamma_m)$. This implies that $\mathsf{span}(T_\infty) = \mathsf{cone}(T_\infty)$ and $\mathsf{span}(T_m) = \mathsf{cone}(T_m)$. Let $C_1, \dots, C_d$ be the strongly connected components of Γ_m. Since $\delta(\Sigma_0)$ is transitive, using Lemma 16,

$$\mathsf{span}(1_{C_1}, \dots, 1_{C_d}) = \mathsf{span}(T_m)^\perp = \mathsf{cone}(T_m)^\perp = K_m^\circ$$

and

$$\mathsf{span}(1_{C_1}, \dots, 1_{C_d}) = \mathsf{span}(T_\infty)^\perp = \mathsf{cone}(T_\infty)^\perp = K_\infty^\circ.$$

Then $K_m = K_\infty$ and $\dim(K_\infty) = n - d$. By Lemmas 21 and 22,

$$\mathsf{len}(\mathcal{K}) \leq \begin{cases} n & \text{if } \mathsf{dim}(K_\infty) = \frac{n}{2}, \\ 3\,\mathsf{dim}(K_\infty) - n - 1 & \text{otherwise.} \end{cases}$$ ◀

Proof of Theorem 4. Computer experiments confirmed Černý conjecture for any synchronizing automata with at most 5 states (see [13, Table 2]). One can check directly that $\mathsf{rt}(\mathcal{A}) \leq (n-1)^2 \leq n^2 - 7n + 7$ for $n \leq 5$.

Now, we assume that $n \geq 6$. Using Lemmas 15 and 16, $\frac{n}{2} \leq \mathsf{dim}(K_\infty) \leq n - 1$. Let S be a nonempty proper subset of Q. By Propositions 10 and 25, if $\mathsf{dim}(K_\infty) = \frac{n}{2}$ then

$$\begin{aligned} \mathsf{len}(\mathcal{K}) + \ell(S) + 1 &\leq n + (n - 1 - \frac{n}{2}) + 1 \\ &= 2n - 3 - (\frac{n}{2} - 3) \leq 2n - 3; \end{aligned} \tag{5}$$

if $\mathsf{dim}(K_\infty) > \frac{n}{2}$,

$$\begin{aligned}\mathsf{len}(\mathcal{K}) + \ell(S) + 1 &\leq (3\,\mathsf{dim}(K_\infty) - n - 1) + (n - 1 - \mathsf{dim}(K_\infty)) + 1 \\ &= 2\,\mathsf{dim}(K_\infty) - 1 \leq 2(n-1) - 1 = 2n - 3. \qquad (6)\end{aligned}$$

Combining Proposition 9 and Equations (5) and (6), every nonempty proper subset of Q is $(2n-3)$-extensible. Using Proposition 5, we obtain that

$$\mathsf{rt}(\mathcal{A}) \leq 1 + (n-2)(2n-3) = 2n^2 - 7n + 7. \qquad \blacktriangleleft$$

4 Conclusion and discussions

We obtain an upper bound for the reset thresholds of $\mathbb{ST}$-automata which improves Rystsov's bound. Using this upper bound, we prove that there exists a synchronizing word of length at most $2n^2 - 7n + 7$ for every synchronizing n-state $\mathbb{ST}$-automata whose letters of defect at most 1.

While Theorem 4 is about a specific class of automata, the lemmas presented in Section 2 may be useful tools for the broader study of synchronizing words. We conclude the article by discussing two classes of automata for which these tools have potential applications.

4.1 One-cluster automata

An automaton (Q, Σ, δ) is called *one-cluster* if it has a letter with only one simple cycle on the set of states, more precisely, there exists a letter $a \in \Sigma$ which acts on P as a cyclic permutation where $P = Q.a^{|Q|-1}$. Write $\mathbb{OC}$ for the family of one-cluster automata. It is clear that one of $\mathbb{OC}$ and $\mathbb{ST}$ do not include the other one. Meanwhile, $\mathbb{OC}$ and $\mathbb{ST}$ have nonempty intersection (e.g. the automaton $\mathcal{C}_4$, see Figure 1).

Steinberg [23] proved Černý Conjecture for one-cluster automata with prime length cycles. Béal, Berlinkov and Perrin showed the reset threshold of n-state one-cluster automata is at most $2n^2 - 7n + 7$. To establish this upper bound, Béal, Berlinkov and Perrin use a linear algebra approach which is different from the approach in Section 2. Observing that the two upper bounds are the same, this may not be a coincidence and it is worth unifying these two proofs. It is also interesting to obtain a better upper bound by combining these two approach.

4.2 Completely reachable automata

An automaton (Q, Σ, δ) is called *completely reachable* if for every nonempty subset $P \subseteq Q$, there exists a word $w \in \Sigma^*$ such that $P = Q.w$. Ferens and Szykuła [8, Corollary 31] proved that the reset threshold of n-state completely reachable automata is at most $2n^2 - n \ln n - 4n + 2$. It is clear that every completely reachable automaton has at least one letter of defect 1, since subsets of $(n-1)$ states are reachable. It is not hard to show if a completely reachable automaton has exactly one letter of defect 1, then it contains a transitive permutation group. Hence, the overlap of completely reachable automata and $\mathbb{ST}$-automata with letters of defect ≤ 1 is quite substantial. Therefore, we believe that the tools presented in this article may also useful for studying completely reachable automata.

References

1 João Araújo, Peter J. Cameron, and Benjamin Steinberg. Between primitive and 2-transitive: synchronization and its friends. *EMS Surv. Math. Sci.*, 4(2):101–184, 2017. doi:10.4171/EMSS/4-2-1.

2 Norman Biggs. *Algebraic graph theory*, volume No. 67 of *Cambridge Tracts in Mathematics*. Cambridge University Press, London, 1974. doi:10.1017/CBO9780511608704.

3 Eugenija A. Bondar, David Casas, and Mikhail V. Volkov. Completely reachable automata: an interplay between automata, graphs, and trees. *Internat. J. Found. Comput. Sci.*, 34(6):655–690, 2023. doi:10.1142/s0129054123450053.

4 Eugenija A. Bondar and Mikhail V. Volkov. Completely reachable automata. In *Descriptional complexity of formal systems*, volume 9777 of *Lecture Notes in Comput. Sci.*, pages 1–17. Springer, [Cham], 2016. doi:10.1007/978-3-319-41114-9_1.

5 Ján Černý. A remark on homogeneous experiments with finite automata. *Mat.-Fyz. Časopis. Sloven. Akad. Vied.*, 14:208–216, 1964.

6 Ján Černý, Alica Pirická, and Blanka Rosenauerová. On directable automata. *Kybernetika (Prague)*, 7:289–298, 1971. URL: http://www.kybernetika.cz/content/1971/4/289.

7 L. Dubuc. Sur les automates circulaires et la conjecture de Černý. *RAIRO Inform. Théor. Appl.*, 32(1-3):21–34, 1998. doi:10.1051/ita/1998321-300211.

8 Robert Ferens and Marek Szykuła. Completely Reachable Automata: A Polynomial Algorithm and Quadratic Upper Bounds. 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 59:1–59:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2023.59.

9 P. Frankl. An extremal problem for two families of sets. *European J. Combin.*, 3(2):125–127, 1982. doi:10.1016/S0195-6698(82)80025-5.

10 François Gonze, Vladimir V. Gusev, Raphaël M. Jungers, Balázs Gerencsér, and Mikhail V. Volkov. On the interplay between Černý and Babai's conjectures. *Internat. J. Found. Comput. Sci.*, 30(1):93–114, 2019. doi:10.1142/S0129054119400057.

11 Jarkko Kari. Synchronizing finite automata on Eulerian digraphs. In *Mathematical foundations of computer science, 2001 (Mariánské Lázně)*, volume 2136 of *Lecture Notes in Comput. Sci.*, pages 432–438. Springer, Berlin, 2001. doi:10.1007/3-540-44683-4_38.

12 Jarkko Kari and Mikhail Volkov. Černý's conjecture and the road colouring problem. In *Handbook of Automata Theory. Vol. I. Theoretical Foundations*, pages 525–565. EMS Press, Berlin, 2021. doi:10.4171/AUTOMATA-1/15.

13 Andrzej Kisielewicz, Jakub Kowalski, and Marek Szykuła. Experiments with synchronizing automata. In *Implementation and application of automata*, volume 9705 of *Lecture Notes in Comput. Sci.*, pages 176–188. Springer, [Cham], 2016. doi:10.1007/978-3-319-40946-7_15.

14 J.-E. Pin. On two combinatorial problems arising from automata theory. In *Combinatorial mathematics (Marseille-Luminy, 1981)*, volume 75 of *North-Holland Math. Stud.*, pages 535–548. North-Holland, Amsterdam, 1983. doi:10.1016/S0304-0208(08)73432-7.

15 Jakub Ruszil. Synchronizing automata with coinciding cycles. In *Developments in language theory*, volume 13911 of *Lecture Notes in Comput. Sci.*, pages 208–218. Springer, Cham, 2023. doi:10.1007/978-3-031-33264-7_17.

16 I. K. Rystsov. Quasioptimal bound for the length of reset words for regular automata. *Acta Cybernet.*, 12(2):145–152, 1995. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3453.

17 I. K. Rystsov. On the length of reset words for automata with simple idempotents. *Kibernet. Sistem. Anal.*, 36(3):32–39, 187, 2000. doi:10.1007/BF02732984.

18 I. K. Rystsov. On the Cerny problem for automata with simple idempotents. *Kibernet. Sistem. Anal.*, 58(1):3–10, 2022.

19 Igor Rystsov and Marek Szykuła. Reset thresholds of transformation monoids. *Cybernetics and Systems Analysis*, pages 1–9, 2024. `doi:10.1007/s10559-024-00660-z`.

20 Yaroslav Shitov. An improvement to a recent upper bound for synchronizing words of finite automata. *J. Autom. Lang. Comb.*, 24(2-4):367–373, 2019. `doi:10.15388/na.2019.3.3`.

21 P. H. Starke. Eine Bemerkung über homogene Experimente. *Elektron. Informationsverarbeitung Kybernetik*, 2:257–259, 1966.

22 Benjamin Steinberg. Černý's conjecture and group representation theory. *J. Algebraic Combin.*, 31(1):83–109, 2010. `doi:10.1007/s10801-009-0185-0`.

23 Benjamin Steinberg. The černý conjecture for one-cluster automata with prime length cycle. *Theoret. Comput. Sci.*, 412(39):5487–5491, 2011. `doi:10.1016/j.tcs.2011.06.012`.

24 Marek Szykuła. Improving the Upper Bound on the Length of the Shortest Reset Word. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.STACS.2018.56`.

25 A. N. Trahtman. The Černý conjecture for aperiodic automata. *Discrete Math. Theor. Comput. Sci.*, 9(2):3–10, 2007. `doi:10.46298/dmtcs.395`.

26 M. V. Volkov. Synchronizing automata preserving a chain of partial orders. *Theoret. Comput. Sci.*, 410(37):3513–3519, 2009. `doi:10.1016/j.tcs.2009.03.021`.

27 M. V. Volkov. Synchronization of finite automata. *Russian Mathematical Surveys*, 77(5):819–891, 2022. `doi:10.4213/rm10005e`.

28 A. Yu. Zubov. On the diameter of the group S_N with respect to a system of generators consisting of a complete cycle and a transposition. In *Proceedings in discrete mathematics, Vol. 2 (Russian)*, volume 2 of *Tr. Diskretn. Mat.*, pages 112–150. Nauchn. Izd. TVP, Moscow, 1998.